UNIVERSITÀ CA' FOSCARI - VENEZIA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica (Triennale)

Tesi di Laurea

Laureando: Perinato Max Perry

iCa'Foscari: Progettazione e sviluppo di un'applicazione per iOS

Relatore: Prof. Celentano Augusto

Sessione invernale
Anno accademico 2009-2010

Ad Alvise Marotta, che oggi 16 dicembre 2010 avrebbe compiuto 33 anni.

Riconoscimenti

Vorrei ringraziare tutte le persone che mi hanno sostenuto nella strada intrapresa per raggiungere questa Laurea in Scienze Informatiche: i miei familiari, per il loro incoraggiamento e la loro pazienza, i docenti del DSI, per le conoscenze apprese dai loro insegnamenti, e i miei amici con i quali ho condiviso questi bellissimi anni di Università.

Un ringraziamento speciale è rivolto a due colleghi di IT4B, Cristian De Zotti e Vadym Boychuk, la cui collaborazione è stata (e sarà) di fondamentale importanza per la riuscita di questo progetto.

16/12/2010

Abstract

La recente diffusione dei dispositivi iOS di Apple ha permesso a questi di affermarsi con successo in molti ambiti applicativi. Uno di questi corrisponde ai servizi mobili per le Università. La Tesi si propone di analizzare le potenzialità della piattaforma offerta da Apple nel suddetto ambito di applicazione, con la progettazione e lo sviluppo di un software che si occupi di offrire mobilità ai servizi comunicativi e informativi dell'Ateneo già esistenti. Il lavoro di progettazione e sviluppo, svolto secondo le pratiche di Ingegneria del Software, ha come suo prodotto un sistema client (device) - server basato sulle tecnologie seguenti: iOS SDK, linguaggi di markup e data interchange (XML e JSON), wrapper HTML e database relazionali.

Prefazione

Lo scopo del presente elaborato è quello di analizzare le attuali necessità dell'Università in termini di distribuzione dei propri contenuti informativi sulla piattaforma mobile ad oggi più interessante e diffusa, e di elaborarne una soluzione soddisfacente e competitiva.

La tesi dunque è stata suddivisa in tre sezioni principali: Inizialmente viene effettuata un'analisi dell'attuale offerta applicativa di altri Atenei sparsi in tutto il mondo. Poi sempre in questa sezione e a partire dall'analisi precedente, sono fissati i requisiti funzionali specifici per l'Università Ca' Foscari di Venezia. La seconda parte del documento riguarda, invece, la progettazione completa dell'applicazione, l'architettura dell'infrastruttura di servizio e la descrizione dell'implementazione di una parte dei requisiti funzionali. Infine, l'ultima sezione riguarda i *future work* e gli obiettivi perseguibili al di fuori dell'ambito di tesi.

A quest'ultimo proposito è dovere precisare che, per motivi tecnici e organizzativi, la progettazione documentata in questa tesi rappresenta esclusivamente una versione iniziale da rivedere e riadattare in conformità con il Sistema Informativo di Ateneo.

Indice dei Contenuti

RICONOSCIMENTI ..... **II**

ABSTRACT ..... **IV**

PREFAZIONE ..... **V**

INDICE DEI CONTENUTI ..... **VI**

Indice delle Tabelle xiv

Indice delle Figure xvii

Indice dei Listati xxii

PARTE PRIMA **23**

Capitolo 1: Introduzione 23

Capitolo 2: Analisi Pre-Progettuale 26

- Ambito di analisi 26
- Criterio di analisi 27
- Breve panoramica sull'offerta di stanford 29
- Elenco completo delle applicazioni analizzate 32
- Analisi esemplificativa di tre applicazioni 36
- Esito dell'analisi esemplificativa 37
- L'offerta delle università italiane 39
- Conclusioni 40

Capitolo 3: Definizione del Progetto 41

- Mission 41
- Scopo e finalità 41
- Principi generali 42
- Motivazioni 43
- Modello di progettazione 45

Capitolo 4: Requisiti 47
Utenza 47
Requisiti funzionali 48
Elenco delle funzionalità 48
Descrizione funzionalità News 49
Descrizione funzionalità Eventi 50
Descrizione funzionalità Directory 51
Descrizione funzionalità Insegnamenti 53
Descrizione funzionalità Mappa 55
Descrizione funzionalità Media 56
Descrizione funzionalità Emergenza 57
Descrizione funzionalità Social 57
Descrizione funzionalità iTunes 58
Descrizione funzionalità Radio Ca' Foscari 58
Descrizione funzionalità Risorse 58
Descrizione funzionalità Avvisi 58
Requisiti non funzionali 59
Performance 59
Latenza 60
Capacità 60
Qualità 61
Compatibilità 61
Riusabilità 62
Robustezza 62
Safety 63
Scalabilità 63
Validità 64
Sicurezza 64
Integrità 65
Privacy 65

Accesso ad aree protette....66
Usabilità....66
Apple Human Interface Guidelines....66
Curva di apprendimento....67
Accessibilità....67
Internazionalizzazione e localizzazione....67
Behaviour....68
Ultimo stato e impostazioni utente....68
Scorciatoie e ricerche....69
Leggibilità dei contenuti....69
Documentazione....69
Conclusioni....70

PARTE SECONDA....71

Capitolo 5: Metodologie e strumenti....71
Metodologie....71
Strumenti....72
Librerie e framework aggiuntivi....73
Versione dell'ambiente di sviluppo del prototipo....75
Conclusioni....76

Capitolo 6: Progettazione di sistema....77
Design pattern....77
Model-View-Controller (MVC)....78
Object Modeling....83
Architettura dell'applicazione....85
Ruoli specifici degli oggetti chiave in iCa'Foscari....88
Scomposizione in sottosistemi....89
Sottosistema News....90
Sottosistema Eventi....92
Sottosistema Directory....94

Sottosistema Insegnamenti....96
Sottosistema Mappa....98
Sottosistema Media....100
Sottosistema Social....102
Sottosistema Radio Ca' Foscari....103
Sottosistema Risorse....105
Sottosistema Avvisi....106
Conclusioni....107
Capitolo 7: Diagrammi dei casi d'uso....109
Legenda dei diagrammi use case....109
Diagrammi uml....111
Casi d'uso News....111
Casi d'uso Eventi....112
Casi d'uso Directory....113
Casi d'uso Insegnamenti....114
Casi d'uso Mappa....115
Casi d'uso Media....116
Casi d'uso Avvisi....117
Conclusioni....117
Capitolo 8: Progettazione della Struttura dati....118
Diagramma entità-relazione....118
Descrizione delle entità....124
Struttura dati News....125
Entità News....125
Entità News_Source....126
Struttura dati Eventi....126
Entità Assignment_Event....127
Entità Calendar....128
Entità Event....128
Entità Exam_Event....129

Entità Lecture_Event 130
Entità Personal_Event 130
Entità University_Event 131
Struttura dati Directory 133
Entità Directory_People 133
Struttura dati Insegnamenti 134
Entità Course 134
Entità Degree 135
Entità Ordinary_Course 136
Struttura dati Mappa 137
Entità Building 137
Entità Building_Area 138
Entità Building_Type 139
Entità Building_Details 140
Struttura dati Social 141
Entità Social_Category 141
Entità Social_Media 141
Struttura dati Risorse 142
Entità Resource_Category 142
Struttura dati Avvisi 143
Entità Alert 143
Entità Alert_Category 144
Strutture dati comuni 144
Entità Address 145
Entità Contact_Number 145
Entità Email 146
Entità Hours 146
Entità Image 147
Entità Link 148
Entità Link_Category 148

Osservazioni sull'eliminazione e il caching dei dati 149
Supporto alla migrazione dei dati 150
Conclusioni 151

Capitolo 9: Progettazione della GUI 152
Specifiche per il design dell'ui 152
Consistenza 152
Risoluzioni dello schermo 154
Autorotation 155
Accorgimenti prestazionali 155
Icona dell'applicazione 155
Colori e stile dell'applicazione 157
Wireframe e diagrammi di collegamento 158
Home screen 159
News 162
Eventi 165
Directory 173
Insegnamenti 177
Mappa 183
Media 187
Emergenza 190
Social 191
iTunesU 194
Radio Ca' Foscari 195
Risorse 197
Avvisi 199
Conclusioni 201

Capitolo 10: Ambiente e infrastruttura di scambio dati 202
Data interchange: xml vs json 202
Sintesi dell'infrastruttura di servizio 204
Procedimento di scambio dei dati 207

Profilo di configurazione dei dispositivi 208
Conclusioni 209

Capitolo 11: Implementazione del sottosistema Directory 210
Header delle classi 210
UNIVEDirectorySearch 210
Header 210
Scheda descrittiva 211
Protocolli e oggetti delegati 212
DirectoryRoot 213
Header 213
Scheda descrittiva 214
DirectorySearch 216
Header 216
Scheda descrittiva 217
DirectoryRecents 219
Header 219
Scheda descrittiva 220
Protocolli e oggetti delegati 221
DirectoryFavorites 222
Header 222
Scheda descrittiva 223
Protocolli e oggetti delegati 224
PersonInfo 224
Header 224
Scheda descrittiva 225
Diagrammi di sequenza 226
Ricerca di un docente nella directory 228
Visualizzazione di un risultato della ricerca 233
Aggiunta ed eliminazione di un preferito 240
Eliminazione della cronologia dei recenti 244

Best practices e ottimizzazioni....245
Lazy loading....245
Memory leak....246
Autorelease e autorelease pool....247
Concorrenza....248
Conclusioni....249
PARTE TERZA....251
Capitolo 12: Future works....251
Completamento del progetto....251
Requisiti funzionali aggiuntivi....252
Requisiti non funzionali e aspetti progettuali da approfondire....254
Distribuzione dell'applicazione....256
Mobile web per l'università ca' foscari....256
Il concetto di web mobile....257
Il progetto UNIVE Mobile Web....259
Conclusioni....261
CONCLUSIONI....262
Glossario....264
Riferimenti....269

Indice delle Tabelle

Tabella 2.1: Analisi pre-progettuale.....37

Tabella 5.1: Librerie utilizzate nel prototipo di iCa'Foscari.....75

Tabella 6.1: Ruoli degli oggetti in un'applicazione *iOS*.....87

Tabella 6.2: Ruoli degli oggetti controller nel sottosistema News. 91

Tabella 6.3: Ruoli degli oggetti model nel sottosistema News.....91

Tabella 6.3: Ruoli degli oggetti controller nel sottosistema Eventi.....93

Tabella 6.4: Ruoli degli oggetti model nel sottosistema Eventi. 93

Tabella 6.5: Ruoli degli oggetti controller nel sottosistema Directory. 95

Tabella 6.6: Ruoli degli oggetti model nel sottosistema Directory.....95

Tabella 6.7: Ruoli degli oggetti controller nel sottosistema Insegnamenti.....97

Tabella 6.8: Ruoli degli oggetti model nel sottosistema Insegnamenti. 97

Tabella 6.9: Ruoli degli oggetti controller nel sottosistema Mappa. 99

Tabella 6.10: Ruoli degli oggetti model nel sottosistema Mappa.....99

Tabella 6.11: Ruoli degli oggetti controller nel sottosistema Media. 101

Tabella 6.12: Ruoli degli oggetti model nel sottosistema Media.....101

Tabella 6.13: Ruoli degli oggetti controller nel sottosistema Social. 102

Tabella 6.14: Ruoli degli oggetti model nel sottosistema Social.....103

Tabella 6.15: Ruoli degli oggetti controller nel sottosistema RCF.....104

Tabella 6.16: Ruoli degli oggetti model nel sottosistema RCF. 104

Tabella 6.17: Ruoli degli oggetti controller nel sottosistema Risorse. 105

Tabella 6.18: Ruoli degli oggetti model nel sottosistema Risorse.....106

Tabella 6.19: Ruoli degli oggetti controller nel sottosistema Avvisi.....107

Tabella 6.20: Ruoli degli oggetti model nel sottosistema Avvisi. 107

Tabella 8.1: Scheda entità generica....124
Tabella 8.2: Proprietà entità News....126
Tabella 8.3: Proprietà entità News_Source....126
Tabella 8.4: Proprietà entità Assignment_Event....127
Tabella 8.5: Proprietà entità Calendar....128
Tabella 8.6: Proprietà entità Event....129
Tabella 8.7: Proprietà entità Exam_Event. 130
Tabella 8.8: Proprietà entità Lecture_Event. 130
Tabella 8.9: Proprietà entità Personal_Event. 131
Tabella 8.10: Proprietà entità University_Event. 132
Tabella 8.11: Proprietà entità Directory_People. 134
Tabella 8.12: Proprietà entità Course. 135
Tabella .8.13: Proprietà entità Degree. 136
Tabella 8.14: Proprietà entità Ordinary_Course. 137
Tabella 8.15: Proprietà entità Building. 138
Tabella 8.16: Proprietà entità Building_Area. 139
Tabella 8.17: Proprietà entità Building_Type. 140
Tabella 8.18: Proprietà entità Building_Details. 140
Tabella 8.19: Proprietà entità Social_Category. 141
Tabella 8.20: Proprietà entità Social_Category. 142
Tabella 8.21: Proprietà entità Social_Category. 143
Tabella 8.22: Proprietà entità Alert. 144
Tabella 8.23: Proprietà entità Alert_Category. 144
Tabella 8.24: Proprietà entità Address. 145
Tabella 8.25: Proprietà entità Contact_Number. 146

Tabella 8.26: Proprietà entità Email. 146
Tabella 8.27: Proprietà entità Hours. 147
Tabella 8.28: Proprietà entità Image. 147
Tabella 8.29: Proprietà entità Link. 148
Tabella 8.30: Proprietà entità Link_Category. 149
Tabella 9.1: Set di icone per l'applicazione. 156
Tabella 11.1: Interfaccia UNIVEDirectorySearch. 212
Tabella 11.2: Protocollo UNIVEDirectorySearchDelegate. 213
Tabella 11.3: Interfaccia DirectoryRoot. 215
Tabella 11.4: Interfaccia DirectorySearch. 218
Tabella 11.5: Interfaccia DirectoryRecents. 221
Tabella 11.6: Protocollo DirectoryRecentsDelegate. 221
Tabella 11.7: Interfaccia DirectoryFavorites. 224
Tabella 11.8: Protocollo DirectoryFavorites Delegate. 224
Tabella 11.9: Interfaccia PersonInfo. 226

Indice delle Figure

Figura 2.1: Schermata principale di *iStanford*.....31
Figura 2.2: Lista delle applicazioni analizzate. 35
Figura 2.3: Schermate principali delle tre applicazioni. 37
Figura 3.1: Processo di sviluppo Agile.....45
Figura 5.1: Provisioning Profile del prototipo di iCa'Foscari. 76
Figura 6.1: Versione tradizionale di MVC. 80
Figura 6.2: Versione *Cocoa* di MVC. 81
Figura 6.3: *Coordinating Controller* come proprietario di un file nib. 82
Figura 6.4: Esempio di *object graph*. 85
Figura 6.5: Architettura di un'applicazione *iOS*.....86
Figura 6.6: Scomposizione in sottosistemi. 89
Figura 6.7: Sottosistema News. 90
Figura 6.8: Sottosistema Eventi.....92
Figura 6.9: Sottosistema Directory.....94
Figura 6.10: Sottosistema Insegnamenti.....96
Figura 6.11: Sottosistema Mappa. 98
Figura 6.12: Sottosistema Media. 100
Figura 6.13: Sottosistema Social. 102
Figura 6.14: Sottosistema Radio Ca' Foscari. 103
Figura 6.15: Sottosistema Risorse. 105
Figura 6.16: Sottosistema Avvisi. 106
Figura 7.1: Legenda diagramma casi d'uso.....110
Figura 7.2: Diagramma casi d'uso News. 111

Figura 7.3: Diagramma casi d'uso Eventi. 112
Figura 7.4: Diagramma casi d'uso Directory. 113
Figura 7.5: Diagramma casi d'uso Insegnamenti. 114
Figura 7.6: Diagramma casi d'uso Mappa. 115
Figura 7.7: Diagramma casi d'uso Media. 116
Figura 7.8: Diagramma casi d'uso Avvisi. 117
Figura 8.1: Panoramica layout diagramma. 119
Figura 8.2: Diagramma struttura dati, sezione 1. 120
Figura 8.3: Diagramma struttura dati, sezione 2. 121
Figura 8.4: Diagramma struttura dati, sezione 3. 122
Figura 8.5: Diagramma struttura dati, sezione 4. 123
Figura 9.1: IconadiiCa'Foscari. 156
Figura 9.2: Launch image di iCa'Foscari. 157
Figura9.3: Presentazione con Navigation controller. 158
Figura 9.4: Presentazione con modalViewController. 159
Figura 9.5: Home screen di iCa'Foscari con TickerBar avvisi. 161
Figura 9.6: Visualizzazione di un avviso della TickerBar. 162
Figura 9.7: Funzionalità News, schermata NE1. 162
Figura 9.8: Funzionalità News, schermata NE2. 163
Figura 9.9: Funzionalità News, schermata NE3. 163
Figura 9.10: Funzionalità News, schermata NE4. 164
Figura 9.11: Funzionalità News, diagramma di collegamento. 165
Figura 9.12: Funzionalità Eventi, schermata EV1 165
Figura 9.13: Funzionalità Eventi, schermata EV2 166
Figura 9.14: Funzionalità Eventi, schermata EV2-1. 167

Figura 9.15: Funzionalità Eventi, schermata EV2-2. 168
Figura 9.16: Funzionalità Eventi, schermata EV4.....169
Figura 9.17: Funzionalità Eventi, schermata EV4-1. 170
Figura 9.18: Funzionalità Eventi, schermata EV4-2. 171
Figura 9.19: Funzionalità Eventi, diagramma di collegamento. 172
Figura 9.20: Funzionalità Directory, schermata DI1.....173
Figura 9.21: Funzionalità Directory, schermata DI2.....173
Figura 9.22: Funzionalità Directory, schermata DI3.....174
Figura 9.23: Funzionalità Directory, schermata DI4.....175
Figura 9.24: Funzionalità Directory, diagramma di collegamento.....176
Figura 9.25: Funzionalità Insegnamenti, schermata IN1.....177
Figura 9.26: Funzionalità Insegnamenti, schermata IN2.....178
Figura 9.27: Funzionalità Insegnamenti, schermata IN3.....179
Figura 9.28: Funzionalità Insegnamenti, schermata IN3-1. 180
Figura 9.29: Funzionalità Insegnamenti, schermata IN4.....180
Figura 9.30: Funzionalità Insegnamenti, schermata IN5.....181
Figura 9.31: Funzionalità Insegnamenti, diagramma di collegamento. 182
Figura 9.32: Funzionalità Mappa, schermata MA1.....183
Figura 9.33: Funzionalità Mappa, schermata MA2.....184
Figura 9.34: Funzionalità Mappa, schermata MA3/3-1. 184
Figura 9.35: Funzionalità Mappa, schermata MA4.....185
Figura 9.36: Funzionalità Mappa, schermata MA4-1. 185
Figura 9.37: Funzionalità Mappa, schermata MA4-2. 186
Figura 9.38: Funzionalità Mappa, diagramma di collegamento.....186
Figura 9.39: Funzionalità Media, schermata ME1. 187

Figura 9.40: Funzionalità Media, schermata ME2. 188
Figura 9.41: Funzionalità Media, schermata ME3. 188
Figura 9.42: Funzionalità Media, schermata ME4. 189
Figura 9.43: Funzionalità Media, diagramma di collegamento. 189
Figura 9.44: Funzionalità Emergenza, schermata EM1. 190
Figura 9.45: Funzionalità Social, schermata SO1. 191
Figura 9.46: Funzionalità Social, schermata SO2. 192
Figura 9.46: Funzionalità Social, schermata SO3. 192
Figura 9.47: Funzionalità Social, diagramma di collegamento. 193
Figura 9.48: Funzionalità iTunesU. 194
Figura 9.49: Funzionalità RCF, schermata RA1. 195
Figura 9.50: Funzionalità RCF, schermata RA2. 196
Figura 9.51: Funzionalità RCF, diagramma di collegamento. 196
Figura 9.52: Funzionalità Media, schermata RI1. 197
Figura 9.53: Funzionalità Media, schermata RI2. 197
Figura 9.54: Funzionalità Media, schermata RI3. 198
Figura 9.55: Funzionalità Media, diagramma di collegamento. 198
Figura 9.56: Funzionalità Avvisi, schermata AV1. 199
Figura 9.57: Funzionalità Avvisi, schermata AV2. 200
Figura 9.58: Funzionalità Avvisi, diagramma di collegamento. 200
Figura 10.1: Infrastruttura Servizi iCa'Foscari. 205
Figura 10.2: Procedimento di scambio dei dati. 208
Figura 10.3: Esempio di iPhone Configuration Profile. 209
Figura 11.1: Legenda diagramma di sequenza. 227
Figura 11.2: Sequenza ricerca docente, caso 1. 229

Figura 11.3: Sequenza ricerca docente, caso 2. 230
Figura 11.4: Sequenza ricerca docente, caso 3 parte 1. 231
Figura 11.5: Sequenza ricerca docente, caso 3 parte 2. 232
Figura 11.6: Sequenza visualizzazione risultato, parte 1. 234
Figura 11.7: Sequenza visualizzazione risultato, parte 2. 235
Figura 11.8: Sequenza metodo addRecent( ), parte 1 236
Figura 11.9: Sequenza metodo addRecent( ), parte 2 237
Figura 11.10: Sequenza metodo fetchExistent( ). 238
Figura 11.11: Sequenza metodo loadRecents( ). 239
Figura 11.12: Sequenza aggiunta preferito. 241
Figura 11.13: Sequenza metodo loadFavorites( ). 242
Figura 11.14: Sequenza eliminazione preferito. 243
Figura 11.15: Sequenza metodo clearRecents( ). 244
Figura 11.16: Trovare i memory leak con Instruments. 247
Figura 12.1: Indicatore di previsione della marea di Venezia. 254
Figura 12.2: Architettura UNIVE Mobile Web. 260

Indice dei Listati

Listato 11.1: UNIVEDirectorySearch.h 211

Listato 11.2: DirectoryRoot.h 214

Listato 11.3: DirectorySearch.h 216

Listato 11.4: DirectoryRecents.h.....219

Listato 11.5: DirectoryFavorites.h 222

Listato 11.6: PersonInfo.h 225

PARTE PRIMA

Capitolo 1: Introduzione

La necessità di distribuire una parte dei servizi e dei contenuti presenti sul sito dell'Ateneo su una piattaforma mobile nasce soprattutto da un'esigenza personale, riscontrata anche da molti altri studenti, che è quella di avere sempre a portata di mano informazioni importanti quali orari dei corsi, contatti e orari di ricevimento dei professori, il calendario accademico, collegamenti a pagine utili, ecc., ma non solo. Spesso sarebbe comodo avere con sé una mappa delle sedi, degli uffici e delle biblioteche, oppure consultare gli eventi organizzati nei vari dipartimenti e inserirli nel proprio calendario, e perché no? Seguire la vita universitaria anche dai *Social Network*! Il tutto visto in qualsiasi luogo e momento da un dispositivo mobile collegato ad *Internet*.

Tuttavia qui sorge un dubbio: gran parte di questi dispositivi (o per essere più precisi gli *smartphone*) sono dotati di un browser per la visualizzazione dei siti web, allora l'utente non dovrebbe far altro che aprire il browser e indirizzarlo al sito dell'Ateneo. Infatti, da un lato è vero che molti *smartphone* sono in grado di visualizzare le pagine web senza alterarne il layout e utilizzando gli stessi fogli di stile *CSS* dei browser desktop, tuttavia dall'altro è anche vero che il layout del sito web dell'Università fu concepito ancora quando tali apparecchi non esistevano nel mercato, perciò a maggior ragione non era stato previsto il supporto ai display *touch screen* di piccole dimensioni. Allo stesso modo la navigabilità e la gerarchia dei contenuti erano stati concepiti per una consultazione distensiva del sito su un comune PC, piuttosto che per una reperibilità immediata delle informazioni su un dispositivo mobile e una fruizione sempre più *on demand*.

Tra gli studenti, gli *smartphone* con browser web integrale più diffusi sono gli iPhone di Apple, i sistemi Windows Phone e Android OS, rispettivamente di Microsoft e Google, e i BlackBerry (alcuni modelli) di RIM. A questi poi si aggiungono, in numero addirittura maggiore, i cosiddetti *Post-PC device*, che comprendono lettori multimediali, come Apple iPod Touch, e *tablet PC,* come Apple iPad e altri basati su Windows e Android.

Dunque, vista l'incapacità del sito web di assolvere da solo alle necessità di un utente mobile, è nato lo stimolo di realizzare una soluzione per fornire a tutti gli studenti in possesso di uno di questi dispositivi tutte le informazioni di cui possono aver bisogno nelle forme *on demand* e *on the go*.

A questo punto però, poiché vi sono, se si può dire, un'infinità di modelli di dispositivi delle classi appena discusse, risultava d'obbligo scegliere un target ben preciso al quale destinare l'attività di progettazione e di sviluppo della soluzione da realizzare, considerato anche che non esistono metodologie e strumenti universali di sviluppo su tutte le piattaforme. Nemmeno i browser si comportano tutti allo stesso modo e le dimensioni degli schermi variano di modello in modello.

E' stato per cui necessario indagare su quali fossero le soluzioni già adottate da altri Atenei e condurre, come vedremo più avanti, un'analisi per determinare allo stesso tempo il target tecnologico migliore e i casi d'uso più frequenti. Da questa si è concluso che la maggior parte delle Università nel mondo che offrono una forma di consultazione mobile possiedono non solo una versione adeguata del proprio sito internet, ma anche un'applicazione specifica per i dispositivi *iOS* di Apple.

I prodotti *iOS* di Apple, fin dall'annuncio nel 2007 dell'iPhone, hanno riscosso un notevole successo, soprattutto in termini di diffusione e usabilità, e al momento rappresentano il punto di riferimento per il mercato. Pertanto, si è ritenuto opportuno concepire una soluzione corrispondente ad una *mobile application* capace di ottenere il massimo dai sistemi *iOS*; che possa inoltre costituire un esempio per la realizzazione di una versione ridotta del sito internet per dispositivi mobili lasciando le porte aperte ad un eventuale *porting* dell'applicazione verso altre piattaforme.

Lo scopo di questa tesi è quello in primo luogo di formalizzare il progetto di tale applicazione, fissandone gli obiettivi e dettandone le linee guida, le specifiche e i requisiti, e in secondo luogo di avviarne la progettazione costruendo un modello iniziale da mettere in opera e ampliare in futuro. Infine, essa si propone di porre le basi dell'implementazione generale del sistema applicativo, sul quale verranno poi sviluppate le singole funzionalità prefissate, una parte delle quali è anch'essa sviluppata nell'ambito di questa tesi a titolo di esempio.

Capitolo 2: Analisi Pre-Progettuale

L'analisi pre-progettuale è di fondamentale importanza al fine di una buona progettazione. Infatti, essa consente di prevenire già da subito alcune lacune nel design che potrebbero presentarsi solamente a prodotto già finito, e in generale di ottenere un prodotto più utile e competitivo senza perdite di tempo.

Questo tipo di analisi consiste nell'esplorare l'ambito di utenza di interesse per il progetto, in questo caso i servizi di tipo mobile offerti dalle Università, quindi individuare i prodotti da recensire ed infine raccogliere alcuni dati sui quali effettuare a posteriori uno studio oppure un confronto.

In questo capitolo, viene per cui descritto il procedimento di analisi, riportando i dati raccolti, per poi trarre le dovute conclusioni.

AMBITO DI ANALISI

Il principale ambito sul quale si è svolta l'analisi, come già premesso, è quello dei servizi mobili di Ateneo. Essa è stata condotta a partire da un gruppo ristretto di Università, estendendosi a mano a mano su una rosa più vasta. Di seguito sono riportati gli Atenei con i quali ha avuto inizio il processo di analisi:

- Carnegie Mellon University, Pittsburgh, Pennsylvania;
- Harvard University, Cambridge, Massachusetts;
- Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts;
- Stanford University, Palo Alto, California.

Questi ultimi sono stati selezionati, non solo per il proprio prestigio, ma soprattutto per il fatto di aver stabilito come requisito iniziale per l'analisi dei servizi quello di avere un sito web appositamente realizzato per i dispositivi mobili, raggiungibile ad un indirizzo del tipo *m.nomeAteneo.edu*.

Tuttavia proseguendo con questo primo campione di analisi si è verificato che per ciascuna di queste Università esisteva sull'App Store di Apple un'applicazione specifica per iPhone e iPod, corrispondente a una versione arricchita degli stessi servizi offerti dal sito web mobile.

Vista la diffusione dei dispositivi *iOS* di Apple è stato ritenuto opportuno modificare il metodo di selezione dei successivi campioni da analizzare e concentrare la ricerca alle sole Università presenti sull'App Store. Ciò ha reso possibile di semplificare i tempi di ricerca, in quanto, essendo l'App Store suddiviso in categorie, è stato in seguito sufficiente sfogliare tutte le applicazioni presenti nella categoria "Istruzione".

Nel corso dell'analisi è stato inoltre rilevato che alcune Università avevano sviluppato applicazioni analoghe anche per sistemi BlackBerry e Android OS. Tuttavia, essendo queste solamente una piccola parte rispetto a tutte quelle presenti, invece, sull'App Store di Apple, non sono state prese in considerazione in questo ambito di analisi. Ulteriori motivazioni sulla scelta di *iOS* verranno discusse nei capitoli successivi.

CRITERIO DI ANALISI

Per stabilire il criterio di analisi è stato preso in considerazione, come modello applicativo, *iStanford*.

iStanford è l'applicazione ufficiale dell'omonima Università. Essa costituisce un valido modello in quanto è stata la prima ad essere sviluppata nel suo genere. L'applicazione, giunta alla terza versione, venne realizzata nel 2007 da una spin-off

proprio dell'Università di Stanford, la *TerriblyClever Design*, poi acquisita da *BlackBoard Inc*.. Quest'ultima sviluppa e concede in licenza applicazioni software e servizi correlati ad oltre 2200 istituzioni in più di 60 paesi.

I parametri di analisi ritenuti validi per eseguire un confronto delle altre applicazioni con questo modello sono risultati i seguenti:

- Usabilità: riguarda la facilità di utilizzo dell'applicativo e la curva di apprendimento delle sue funzioni;
- Reperibilità e leggibilità delle informazioni: indica la velocità nel reperire le informazioni desiderate e la qualità delle stesse in termini di sintesi, utilità e chiarezza;
- Caratteristiche funzionali: prende in considerazione l'insieme delle funzionalità proposte, valutandone efficacia e completezza nell'assolvere ai casi d'uso più frequenti;
- Scalabilità e prestazioni: valuta i tempi di caricamento dell'applicazione e la capacità di gestire nuovi contenuti e funzionalità senza compromettere il design iniziale;
- Interfaccia grafica: capacità di trovare un compromesso tra originalità e *responsiveness* negli elementi dell'*UI*, pur mantenendo lo stile proprio di un'applicazione *iOS*;
- Integrazione dei servizi web: capacità del software nell'integrare i servizi web dell'Ateneo senza trascurarne i più importanti.

Per poter effettuare l'analisi per mezzo di questi parametri è necessario, però, stabilire un sistema di valutazione.

La valutazione consisterà dunque nel fissare un punteggio su una scala da 1 a 5, dove:

- Un punteggio uguale a 5 significa che l'applicazione soddisfa i requisiti del parametro osservato in modo superiore rispetto ad *iStanford*;
- Se il punteggio è uguale a 3 o 4 vuol dire che l'applicazione soddisfa i requisiti del parametro osservato in modo analogo ad *iStanford*;
- Infine, un punteggio inferiore a 3 indica che il software non soddisfa quanto osservato e non è in alcun modo comparabile ad *iStanford*.

Il voto complessivo sarà espresso in centesimi e corrisponderà alla somma dei voti ottenuti moltiplicati per i seguenti fattori relativamente al parametro valutato:

- Usabilità: 4
- Reperibilità e leggibilità delle informazioni: 5
- Caratteristiche funzionali: 3
- Scalabilità e prestazioni: 3
- Interfaccia grafica: 2
- Integrazione dei servizi web: 3

Nei paragrafi seguenti verranno messe a confronto solamente tre applicazioni a titolo esemplificativo.

BREVE PANORAMICA SULL'OFFERTA DI STANFORD

In questo paragrafo vengono fatte alcune considerazioni sul modello su cui si è basata l'analisi.

L'applicazione offerta dall'Università di Stanford si presenta in maniera molto semplice e ordinata. Fin dalla schermata principale, una simulazione della *Spring Board* di *iOS*, si nota che è stata sviluppata in modo strettamente coerente con le specifiche di Apple. Dagli elementi grafici di *UIKit* fino allo stile di navigazione stesso, l'utente si

trova subito a suo agio, come se stesse usando un applicativo in dotazione col proprio dispositivo. Le funzionalità proposte sono riconoscibili fin da subito:

- Atletica: aggiornamenti, punteggi e calendari di ogni attività sportiva di Ateneo;
- Corsi: ricerca dei corsi, degli orari delle lezioni, contatto coi docenti, iscrizione ai corsi e visualizzazioni dei propri esiti;
- Directory: funzionalità di ricerca di studenti, docenti e staff nella directory di Stanford, inserimento dei contatti nella propria rubrica;
- Emergenza: accesso istantaneo ai numeri telefonici di emergenza;
- Eventi: calendario degli eventi universitari e localizzazione sulla mappa;
- Immagini: visualizzazione e condivisione di foto riguardanti l'Università;
- GoTourIt: tour virtuale del campus;
- iTunes U: accesso ai contenuti su iTunes U;
- Biblioteca: ricerca dei contenuti disponibili nelle biblioteche con informazioni sulla disponibilità, luogo e contatti della biblioteca;
- Mappe: ricerca di edifici e luoghi nel campus, posizionamento in tempo reale del bus e visualizzazione dei tragitti, localizzazione GPS;
- News: articoli dai principali notiziari di Ateneo;
- Video: filmati dei corsi e altri contenuti dal canale YouTube;
- 5-Sure: richiesta di un trasporto sicuro all'interno del campus nelle ore notturne.

Figura 2.1: Schermata principale di *iStanford*.

Le caratteristiche appena elencate, sulle quali non andremo maggiormente in dettaglio, soddisfano in maniera rapida ed esaustiva le richieste dell'utente, costituendo una valida estensione su piattaforma mobile dei servizi offerti dal sito web. Per di più, le funzionalità sono coerenti con le pagine di *m.stanford.edu*, rispetto alle quali presentano le informazioni in modo più ricco, su cui l'utente può svolgere delle semplici, ma spesso utili, elaborazioni.

I tempi di caricamento sono complessivamente accettabili, pur tenendo conto che le informazioni sono prelevate da una connessione con un server esterno, e l'interfaccia grafica è sempre reattiva.

L'utente ha a sua disposizione, in quasi tutta l'applicazione, una barra di ricerca, grazie alla quale è possibile trovare direttamente le informazioni desiderate senza dover necessariamente navigare tra le categorie. Ulteriormente, è possibile anche consultare una lista delle schede visualizzate di recente, oppure salvarle tra i preferiti.

Inoltre, per concludere questa sezione del capitolo, l'applicativo sfrutta al meglio l'hardware a disposizione e il sistema operativo che lo ospita. Infatti, è possibile accedere a tutte le funzioni di base del dispositivo, come ad esempio il GPS interno e le mappe, il telefono, la rubrica, il calendario, il browser, l'invio di e-mail, ecc.

ELENCO COMPLETO DELLE APPLICAZIONI ANALIZZATE

L'analisi complessivamente ha coinvolto 107 applicazioni disponibili sull'App Store nella sezione "Istruzione" al mese di Agosto 2010. Tra queste è stato possibile individuare le principali compagnie che hanno sviluppato una parte di queste applicazioni, e sono le seguenti:

- Blackboard, Inc. – www.blackboard.com (24 applicazioni);
- Straxis, LLC – www.straxis.com (9 applicazioni);
- Amuzu, Inc. – www.amuzu.com (5 applicazioni);

Le restanti 69 applicazioni sono state sviluppate internamente o da ditte di sviluppo software specifiche. Inoltre, è necessario precisare che una parte dei software analizzati (17) svolgevano solamente una funzione di marketing per l'Università, fornendo informazioni sui corsi di laurea o un tour virtuale dell'Ateneo.

Altri 19 applicativi, invece, richiedevano dei dati di accesso per essere eseguiti, oppure presentavano dei problemi di connessione al server. Nelle pagine seguenti è riportata la lista completa delle applicazioni.


Academy
AUSB
Ball State University Ca...
Biola Mobile
bMobi
CampusLink
CanooNet
CGU
CMU
Corllins University
CornellCast Mobile
Coventry University
Curtin University iPortfolio
DePauw University
Deutsche Gebärdenspra...
Discover Northeastern
DukeMobile
FPU
FSU Mobile
Harding University App
Harvard Mobile
HKUCC
iBarry
iECPI
iEKU
iHusker
IlliniMobile
Institut Montaigne
InTouch
iPoli
iStanford
iTCU
iTour UNC Wilmington
iUni
iUsask
iWKU
iWVU
Kean Mobile
KHCU-LMS
m.UW

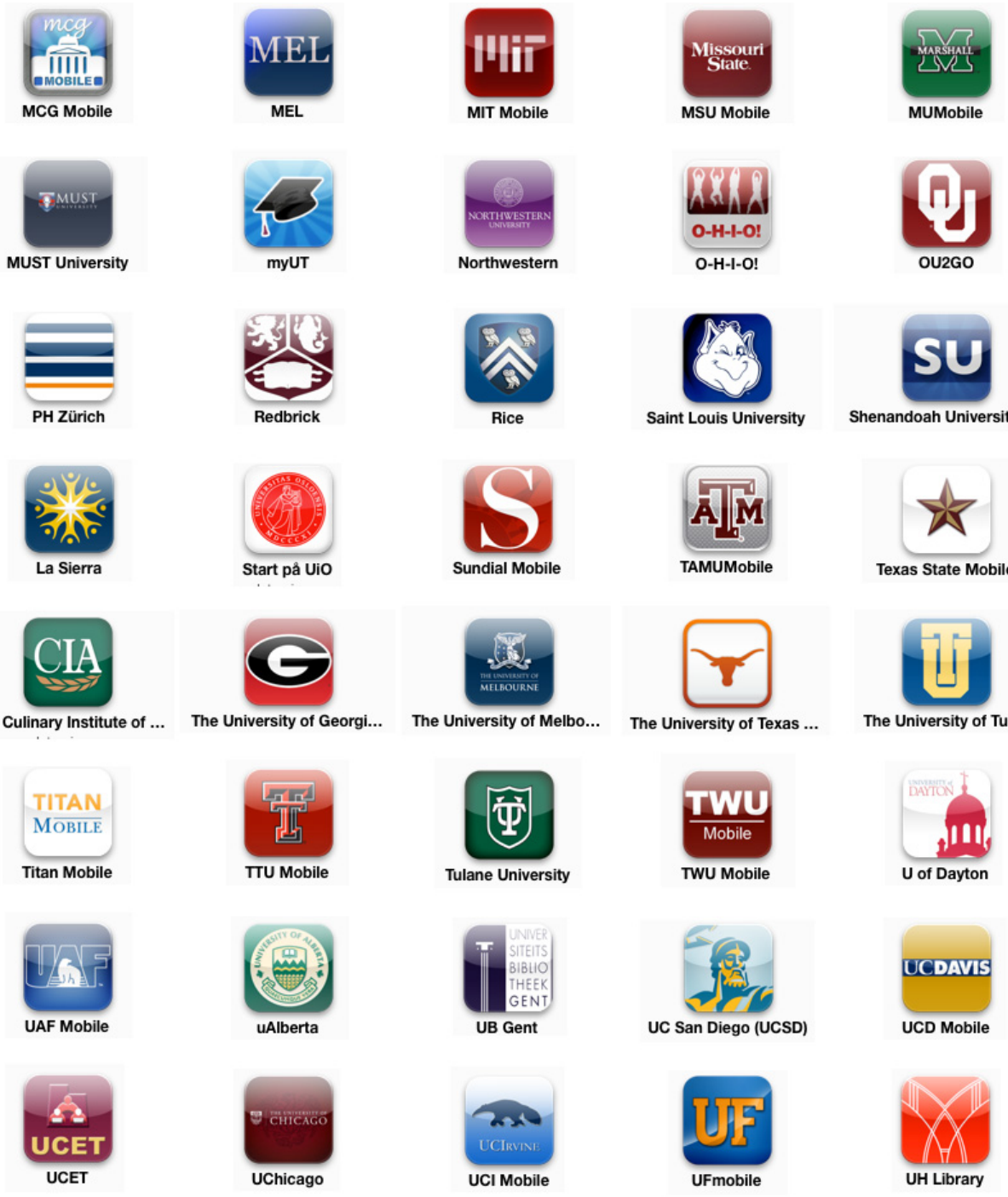
MCG Mobile
MEL
MIT Mobile
MSU Mobile
MUMobile
MUST University
myUT
Northwestern
O-H-I-O!
OU2GO
PH Zürich
Redbrick
Rice
Saint Louis University
Shenandoah University ...
La Sierra
Start på UiO
Sundial Mobile
TAMUMobile
Texas State Mobile
The Culinary Institute of ...
The University of Georgi...
The University of Melbo...
The University of Texas ...
The University of Tulsa
Titan Mobile
TTU Mobile
Tulane University
TWU Mobile
U of Dayton
UAF Mobile
uAlberta
UB Gent
UC San Diego (UCSD)
UCD Mobile
UCET
UChicago
UCI Mobile
UFmobile
UH Library

Figura 2.2: Lista delle applicazioni analizzate.

Sempre nella categoria “Istruzione” sono state trovate oltre 50 applicazioni di Scuole Superiori americane, in buona parte realizzate da *Blackboard*. Sebbene questi applicativi siano molto simili a quelli realizzati per le Università, essi sono stati esclusi dall’analisi.

ANALISI ESEMPLIFICATIVA DI TRE APPLICAZIONI

A titolo esemplificativo di quanto svolto nell'analisi pre-progettuale, nella tabella a seguire sono messe a confronto con il modello di Stanford tre applicazioni:

- Harvard Mobile, Harvard University, Cambridge, Massachusetts;
- MIT Mobile, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts;
- Texas, The University of Texas, Austin, Texas.

La valutazione delle caratteristiche verte esclusivamente su quelle ritenute più importanti in comune tra le tre applicazioni.

	Harvard Mobile	MIT Mobile	Texas
Usabilità	5	3	3
Reperibilità e leggibilità delle informazioni	5	5	3
Caratteristiche Funzionali			
News	4	5	3
Mappa	3	3	1
Directory	3	3	3
Eventi	2	2	1
Punteggio caratteristiche (media aritmetica dei punti)	3	3	2
Scalabilità e prestazioni	3	3	3
Interfaccia grafica	3	4	3
Integrazione dei servizi web	3	3	2
Voto complessivo	78	72	54

Tabella 2.1: Analisi pre-progettuale.

Figura 2.3: Schermate principali delle tre applicazioni.

ESITO DELL'ANALISI ESEMPLIFICATIVA

Dall'analisi emergono alcune importanti considerazioni che giustificano la valutazione finale, in particolare si evince che:

- Harvard Mobile ha un'interfaccia grafica analoga a quella di *iStanford*, ma presenta una facilità considerevole nel reperire le informazioni, con risvolti importanti per l'usabilità. Infatti è possibile già dalla schermata principale effettuare una ricerca su tutte le funzionalità disponibili. Tra queste ultime "Eventi", al contrario del software di Stanford, non è realizzata come un vero e proprio calendario, tuttavia le informazioni sono comunque discretamente reperibili e rappresentate con qualità.

- MIT Mobile utilizza, invece, una struttura grafica diversa (come visibile in figura), costituita da una *Tab Bar* sempre presente, che nasconde però, per motivi di spazio, delle funzionalità importanti. L'usabilità rimane comunque alla pari dell'offerta di Stanford (tranne per lo spazio disponibile sullo schermo molto minore) a cui si aggiungono delle interessanti *Scrolling Toolbar* che permettono all'utente, ad esempio, di scegliere categorie ed effettuare ricerche, senza dover ricorrere ad una navigazione tra tabelle. Per quanto riguarda le caratteristiche funzionali, "News" è stata realizzata in modo eccellente, mentre "Eventi" presenta gli stessi problemi di Harvard Mobile. Infine, mentre la *Tab Bar* permette di ridurre i tempi di caricamento delle singole funzionalità, essa costituisce un impedimento dal punto di vista della scalabilità, in quanto secondo le specifiche di Apple non è possibile aggiungere più di 5 elementi sulla *Tab Bar*, limitando l'aggiunta di eventuali nuove caratteristiche.

- Texas, infine, si presenta in modo molto simile ad *iStanford*. La possibilità di aggiungere alla schermata principale delle scorciatoie che puntano a dei casi d'uso specifici sembra utile per velocizzare l'accesso alle informazioni, ma forse in realtà non è molto efficace. Dal punto di vista dell'usabilità, invece, si nota la mancanza di poter visualizzare una lista dei recenti, oppure di aggiungere dei preferiti. Inoltre, un grave errore è stato quello di ripristinare lo stato precedente dell'applicazione già dalla schermata principale, piuttosto che dalle singole funzionalità come su *iStanford*, costituendo alla fine una perdita di tempo per l'utente, con la possibilità che l'applicazione non riesca più a ripartire. Per quanto concerne la qualità di presentazione delle informazioni, se si guarda la

funzionalità “Directory” non si notano differenze con il modello dell’analisi, ciò non è vero però per “Mappa”, dove per i punti di interesse disponibili è presente solo una mera immagine, senza considerare che i *POI* non possono essere né filtrati né consultati da una lista. Anche “Eventi” non solo presenta gli stessi problemi delle altre due proposte (pur facendo uso di un calendario), ma si aggiungono inoltre dei seri difetti nell’interfaccia e alcune opzioni, come ad esempio la visualizzazione sulla mappa degli eventi. Per di più, la possibilità di consultare i corsi tenuti dall’Università è assente. Ciò giustifica il voto assegnato alla voce Integrazione dei servizi web.

Traendo delle conclusioni, possiamo affermare che l’offerta di Harvard per alcuni aspetti è addirittura migliore di quella di Stanford, mentre la proposta del MIT perde alcuni punti per le limitazioni imposte dalla struttura dell’*UI*. Dietro a queste si posiziona quanto sviluppato per l’Università di Austin, che pur presentandosi molto bene stilisticamente, presenta delle serie trascuranze nei contenuti, cioè quanto di più importante per soddisfare le aspettative dell’utente.

L’OFFERTA DELLE UNIVERSITÀ ITALIANE

Al mese di Agosto 2010, periodo in cui si è svolta l’analisi, l’unica applicazione presente sull’App Store per un Ateneo italiano è iPoli del Politecnico di Milano. Tuttavia questa non è un’applicazione ufficiale. Infatti, il software gratuito sviluppato da una coppia di studenti del PoliMi, è un *wrapper* dei dati raccolti dalle pagine del sito web.

Nel frattempo però sono da poco state sviluppate delle nuove applicazioni per le Università italiane, di cui solo una è la prima e l’unica ad essere ufficiale:

- INFOSTUD: permette agli studenti dell'Università La Sapienza di Roma di gestire e prenotare i propri esami direttamente, e comodamente, dall'iPhone e dall'iPod Touch (€ 1,59);
- Genial Sapienza: dedicata agli studenti dell'Università La Sapienza di Roma. Si tratta di un programma grazie al quale è possibile gestire e visualizzare i propri esami direttamente da iPhone e iPod Touch (€ 0,79);
- iKore: l'Università di Enna "Kore" è la prima, in Italia, ad avere un'applicazione specifica ed ufficiale per iPhone, disponibile gratuitamente su AppStore;
- iSDAI: permette di ottenere tutte le informazioni del corso di laurea in informatica dell'Università di Catania.

CONCLUSIONI

L'analisi pre-progettuale si è rivelata effettivamente di estrema importanza per porre le basi del progetto che andremo a descrivere di qui a poco; in particolare per i seguenti motivi. Innanzitutto ha permesso di identificare la piattaforma target più adatta sul quale sviluppare il software, poi ha consentito di estrapolare le funzionalità maggiormente offerte e di stabilire dei requisiti funzionali. Per di più ha contribuito alla creazione dei *wireframe* delle interfacce grafiche, svolgendo una selezione tra i campioni analizzati delle *UI* meglio riuscite. Infine, ha permesso di definire delle specifiche di implementazione, soprattutto riguardanti l'usabilità, finalizzate a prevenire alcuni difetti rilevati nelle applicazioni osservate.

Capitolo 3: Definizione del Progetto

Nei paragrafi seguenti viene stabilito e motivato lo scopo del progetto di tesi che prende il nome di iCa'Foscari, descrivendo inoltre i suoi principi generali. Infine, viene definito il modello di progettazione adottato.

MISSION

iCa'Foscari è un'applicazione per dispositivi *iOS* pensata appositamente per l'Università Ca' Foscari di Venezia. Essa, come abbiamo già visto nel capitolo precedente, ha come suo modello di ispirazione *iStanford*, e il suo ruolo fondamentale sarà quello di costituire una nuova piattaforma di *mobile edutainment* per l'Università. La missione di iCa'Foscari, infatti, è quella di

> *"Offrire una varietà di servizi studenteschi e di facoltà in ogni luogo e momento; motivando gli studenti a frequentare i corsi e a partecipare alle iniziative universitarie, attraverso una nuova forma di comunicazione ed educazione, basata sulle tecnologie mobili emergenti."*

SCOPO E FINALITÀ

Lo scopo del presente lavoro di tesi è quello di fornire la progettazione e le specifiche di implementazione "iniziali" per lo sviluppo futuro di un'applicazione da distribuire sull'App Store sotto il nome di iCa'Foscari, mirata alla fornitura dei servizi web di Ateneo sulla piattaforma mobile ad oggi maggiormente riconosciuta sul mercato, ovvero la piattaforma *iOS* di Apple.

Pertanto si precisa che da questa tesi non ci si deve aspettare un prodotto finito, sebbene ne verrà presentato un prototipo avanzato, bensì una base progettuale solida da estendere, integrare e conformare con la struttura reale del sistema informativo dell'Università.

Il risultato della progettazione, essendo questa svolta secondo le comuni pratiche di Ingegneria del Software, potrà inoltre essere riutilizzato per lo sviluppo sulle piattaforme mobili concorrenti.

L'obiettivo finale di iCa'Foscari sarà quello invece di portare a termine la sua missione, il cui perseguimento ha inizio proprio con il lavoro di progettazione presentato in questo elaborato.

PRINCIPI GENERALI

Il progetto iCa'Foscari può altresì essere riassunto nei seguenti principi, in particolare sui quali si baserà la definizione dei requisiti funzionali.

- *Mobile learning and edutainment*: rappresentano una nuova forma di educazione interattiva che permette di accrescere il livello di interesse degli studenti nei corsi universitari e quindi la frequenza ai corsi. Possibili implementazioni comprendono, ad esempio, informazioni sul programma e il contenuto dei corsi, assegnazione di task e compiti per casa, e condivisione di materiali e media da parte di docenti e universitari.

- *Comunicazione*: riguarda la notifica di comunicazioni e avvisi urgenti in tempo reale; come ad esempio, aggiornamenti sugli orari dei corsi, avviso di lezioni sospese, orari di ricevimento, ecc. Comunicazioni specifiche sui corsi (news o reminder) accessibili *on-the-go*. Numeri utili, contatto con la segreteria studenti e i professori; via e-mail o telefono.

- *Mappa e luoghi di interesse*: consiste nel fornire una mappa che indichi come raggiungere le strutture universitarie e fornisca indicazioni su diverse categorie di

luoghi di interesse; come ad esempio biblioteche, uffici, aule, dipartimenti, locali convenzionati, ecc.

- *Vita universitaria ed eventi*: insieme al principio di *comunicazione* corrisponde nell'offrire la possibilità di tenersi sempre aggiornati sulle attività promosse dall'Università. Sia per mezzo di un calendario degli eventi, sia attraverso collegamenti ai social network e ai media, quali immagini e video.

- *News e informazione*: divulgazione e consultazione delle notizie dalle principali fonti di informazione dell'Ateneo.

MOTIVAZIONI

Di seguito, rispondendo ad alcune domande, si danno delle motivazioni che giustificano soprattutto la scelta di progettare per i dispositivi Apple.

- *Perché scegliere come target i dispositivi iOS?*

I dispositivi *iOS* di Apple costituiscono una nuova frontiera tecnologica nella quale molte aziende ed istituzioni stanno investendo. Apple inoltre ha sempre avuto un rapporto solido con il settore educational. Per di più, l'analisi pre-progettuale dimostra come questa sia la tecnologia maggiormente utilizzata in ambiente accademico. Quindi, al fine di offrire all'Università una tecnologia all'avanguardia, investire esclusivamente sulla piattaforma *iOS* rappresenta una scelta necessaria.

- *Limitarsi ad adattare il portale web universitario ai dispositivi iOS può dare gli stessi vantaggi di un'app pubblicata sull'App Store?*

No. Creare dei nuovi fogli di stile CSS specificamente per *WebKit*, sebbene meno impegnativo, non porta al completo raggiungimento degli obiettivi stabiliti. In quanto non si sfrutterebbero appieno le potenzialità dell'SDK fornita da Apple. Piuttosto potrebbe essere oggetto di un altro lavoro l'adattamento del sito web dell'Ateneo ai dispositivi mobili concorrenti.

- *Quale può essere l'utilità di un dispositivo iOS in ambito accademico?*

L'Università, sfruttando i dispositivi *iOS* già molto diffusi tra gli studenti (iPod e iPhone prevalentemente), può ottenere livelli di istruzione e apprendimento migliori. Le analisi condotte dalle Università che hanno già applicato questa forma di *edutainment*, hanno evidenziato infatti che gli universitari utilizzano almeno una volta al giorno i dispositivi a scopi accademici; molti di questi, inoltre, hanno sostituito nelle aule i propri laptop e in alcuni casi i libri di testo stessi coi sistemi *iOS*. Questi non solo vengono utilizzati dagli alunni per prendere appunti e seguire le lezioni, ma sfruttano la connettività esistente per accedere a internet e ai servizi universitari attraverso l'app dedicata. Le tecnologie *iOS* di Apple costituiscono, dunque, un sistema di apprendimento e comunicazione efficace e a basso costo.

- *Infine, cosa si può realmente ottenere da questa progettazione, e quali sarebbero i benefici di un prodotto finito?*

Il lavoro di design presentato in questa tesi non può determinare da solo la realizzazione di un prodotto finito. Infatti, è necessario che venga innanzitutto revisionato e approvato dagli enti organizzatrici, tra cui il Rettorato e il C.S.I.T.A., per poi intraprendere un iter di estensione, integrazione e testing, fino alla pubblicazione sull'App Store. Tuttavia, questa progettazione prevede già da subito un ricco pacchetto funzionale, che da solo potrebbe

costituire un valido strumento di supporto agli universitari, nella scena dei principi precedentemente esposti.

MODELLO DI PROGETTAZIONE

Il modello di progettazione più adeguato alla realizzazione di iCa'Foscari è la metodologia *Agile*. I suoi punti di forza, come i tempi di consegna rapidi, la facilità di comunicazione nel team e l'accoglienza di modifiche ai requisiti, rispecchiano i bisogni di un prodotto in costante evoluzione quale può essere un'applicazione *iOS*. Non per niente è il modello scelto dalla stragrande maggioranza di software house che sviluppano su questa piattaforma.

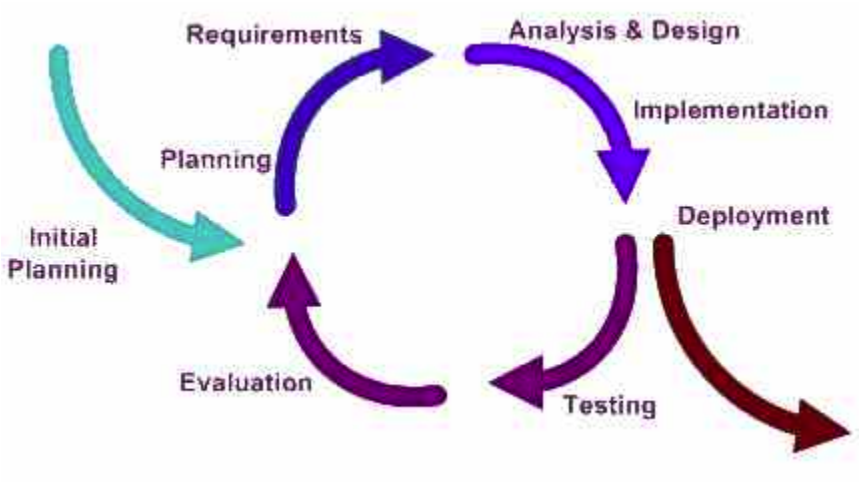

Figura 3.1: Processo di sviluppo Agile.

La metodologia consiste in una serie di iterazioni nelle quali un team lavora su un ciclo di sviluppo completo del software, dall'analisi dei requisiti fino alla fase di testing, da cui risulta un prodotto funzionante da presentare al committente. Ciò permette di ridurre i rischi e di adattare velocemente il progetto ai cambiamenti. Una sola iterazione

però non è sufficiente ad ottenere una vera e propria *release* del prodotto, ma permette comunque di avere una versione utile per monitorare l'andamento del progetto.

Il lavoro presentato in questa tesi rappresenta così l'avvio della prima iterazione di questo metodo di sviluppo incrementale. Nella terza e ultima parte dell'elaborato verranno fatte ulteriori precisazioni sull'organizzazione e il ciclo di vita del progetto, introducendo, inoltre, nuovi requisiti e alcuni future work di particolare rilevanza per il progetto stesso.

Capitolo 4: Requisiti

In questa sezione vengono identificati e definiti i requisiti funzionali e non funzionali relativi all'applicazione iCa'Foscari, specificando il relativo target di utenza.

Lo scopo dell'applicativo, come già annunciato, è quello di fornire a tutti gli studenti dell'Università Ca' Foscari un sistema per consultare sul proprio dispositivo *iOS* tutte le informazioni riguardanti l'Ateneo, dai corsi agli eventi universitari, e ricevere aggiornamenti in tempo reale.

UTENZA

L'utenza che avrà la possibilità di utilizzare iCa'Foscari è rappresentata virtualmente da tutti i possessori di un dispositivo *iOS*, in qualunque parte del mondo. Infatti, poiché non è richiesta alcuna autenticazione, chiunque voglia potrà scaricare iCa'Foscari dall'App Store ed iniziare a fruire dei suoi contenuti.

Tuttavia, i requisiti che andremo a discutere di qui a poco fanno di iCa'Foscari un applicativo espressamente costruito attorno alle esigenze degli studenti dell'Università Ca' Foscari, i quali costituiscono l'utenza vera e propria del sistema.

Secondariamente, l'applicazione è rivolta anche agli stessi docenti, che potranno trovare numerose informazioni dedicate agli studenti anche di loro utilità; sebbene l'estensione a questo tipo di utenza rappresenti più concretamente l'oggetto di un eventuale future work.

Ulteriormente, considerato che la progettazione di iCa'Foscari rispecchia quella propria di un'applicazione mobile, lo sviluppo dovrà tenere in seria considerazione i requisiti funzionali e non funzionali imposti, da un lato, da un'utenza in mobilità, per di più di nicchia, che si aspetta, ad esempio, un numero di funzionalità non troppo elevato,

tempi di risposta rapidi e una grafica esaltante, e dall'altro, da un dispositivo con delle prestazioni limitate.

Inoltre, l'implementazione dovrà supportare la localizzazione di altre lingue oltre a quella italiana, in modo tale da poter essere utilizzata anche da studenti in erasmus. Allo stesso modo un livello minimo di accessibilità dovrà essere garantito per la rispettiva minoranza di utenti.

REQUISITI FUNZIONALI

Nelle pagine seguenti sono elencate e descritte le funzionalità principali del sistema, elaborate sulla base dell'analisi pre-progettuale. Oltre a ciò, per ogni funzionalità è riportata anche una lista delle informazioni attualmente disponibili nelle pagine del sito www.unive.it, al fine di avere un'indicazione pressappoco realistica dei dati consultabili dall'utente nell'applicazione; in preparazione ad una futura fase di conformazione e integrazione dell'applicazione col Sistema Informativo di Ca' Foscari.

I diagrammi UML, che illustrano i casi d'uso specifici di ogni macrofunzionalità, saranno trattati nella seconda parte di questa tesi.

Elenco delle funzionalità

Di seguito è riportato l'elenco completo delle funzionalità di iCa'Foscari (in ordine arbitrario).

1. News;
2. Eventi;
3. Directory;
4. Insegnamenti;
5. Mappa;
6. Media;

7. Emergenza;
8. Social;
9. Radio Ca' Foscari;
10. iTunesU;
11. Risorse;
12. Avvisi.

Descrizione funzionalità News

La funzionalità News permette all'utente di sfogliare e leggere articoli raccolti da *feed RSS*. Gli articoli sono suddivisi per categoria/origine e possono essere consultati in versione integrale mediante il browser *Safari* oppure in forma *embedded* senza chiudere l'applicazione. Un elenco degli articoli più recenti raccoglie le notizie di tutte le categorie ordinate per data di pubblicazione.

L'utente può aggiungere (e rimuovere) gli articoli ad una lista dei preferiti. Inoltre, le notizie possono essere condivise con altri utenti per mezzo di una e-mail predefinita.

Le origini RSS comprendono i principali feed elencati sul sito Unive.it, a questi si aggiungono feed di altri blog e forum gestiti dagli studenti.

L'elenco seguente riporta i dati disponibili sul sito www.unive.it utili per questa funzionalità:

- Feed RSS
    - Aggiornamenti Infoscari;
    - Blog del Rettore;
    - Bandi e concorsi;

- Blog Radio Ca' Foscari.

Descrizione funzionalità Eventi

Eventi è un insieme di calendari, suddivisi per categoria o tipologia, che riportano eventi e attività dell'Università. I calendari possono essere consultati per giorno o mese. Gli eventi possono essere ricercati semplicemente per nome o a seconda dei parametri disponibili.

L'utente, inoltre, ha a disposizione un calendario accademico, nel quale, oltre agli eventi comuni in tutto l'ateneo, sono presenti quelli specifici della propria facoltà e corso di laurea (specificati dall'utente nell'impostazione del suo profilo). Per di più, non solo l'utente può inserire in questo calendario degli eventi personali, ma può anche consultare gli orari dei corsi aggiunti nella lista "I miei corsi".

Un altro calendario, invece, fornisce le date degli appelli d'esame relativi a tutti gli insegnamenti del proprio corso di laurea.

I calendari sono realizzati come il Calendario nativo di *iOS*. Gli eventi personali inseriti dall'utente nel calendario accademico sono automaticamente sincronizzati col calendario del dispositivo, al fine di poter utilizzare la funzione allarme se richiesta.

Selezionando un evento dal calendario è possibile consultare una scheda coi dettagli dell'evento. Inoltre, è possibile aprire le URL specificate, mettersi in comunicazione coi contatti disponibili (telefono o e-mail) e visualizzare sulla mappa il luogo dell'evento.

L'elenco seguente riporta i dati disponibili sul sito www.unive.it utili per questa funzionalità:

- Evento
    - Titolo;
    - Categoria;
    - Tema;
    - Note;
    - Organizzatore;
    - Relatore;
    - Sede;
    - Data e ora;
    - Durata;
    - Approfondimenti;
    - Allegati (link).

Descrizione funzionalità Directory

Rappresenta una directory di tutti i docenti dell'Università. Ogni docente può essere ricercato semplicemente per nome o eventualmente specificando dei parametri di ricerca avanzata. I risultati selezionati possono aggiunti ad una lista dei preferiti, altrimenti sono reperibili nuovamente da una lista dei recenti.

Una scheda dettagliata fornisce le informazioni disponibili per ciascun professore. L'utente può mettersi in contatto direttamente via mail o telefono, e visitare le URL presenti in modo *embedded* o aprendo *Safari*. Sempre dal profilo del docente, l'utente può visualizzare direttamente la scheda dei corsi per i quali è responsabile.

I docenti dei corsi aggiunti dall'utente nella lista "I miei corsi" sono inseriti automaticamente nella lista dei preferiti.

L'elenco seguente riporta i dati disponibili sul sito www.unive.it utili per questa funzionalità:

- Docente
    - Nome e cognome;
    - Qualifica;
    - E-mail;
    - Ufficio;
        - Dipartimento;
        - Sede;
        - Telefono;
        - Fax.
    - Orario di ricevimento;
    - Foto;
    - Sito web.
- Didattica
    - Corso di laurea (id, nome);
        - Insegnamento (nome, frequentazione, crediti, id).
- Avvisi
- Collegamenti
    - Pubblicazioni;
    - Materiale didattico;
    - Insegnamenti anni precedenti;

- Curriculum.

Descrizione funzionalità Insegnamenti

Svolge le stesse funzioni di Directory, ma orientate agli insegnamenti. Quest'ultimi in particolare, oltre a poter essere consultati tramite una ricerca per nome o avanzata tra tutti gli insegnamenti disponibili, sono suddivisi per facoltà e tipo di laurea (triennale/magistrale), rendendo dunque possibile anche la navigazione gerarchica.

Dalla scheda dettagliata di ogni insegnamento non solo è possibile ottenere informazioni ad esso specifiche, come ad esempio gli orari delle lezioni, ma è anche possibile consultare direttamente la scheda dei docenti responsabili, oppure vedere sulla mappa la sede dove si svolgono le lezioni.

Come già anticipato, ogni insegnamento può essere aggiunto alla lista "I miei corsi". Ciò abilita la funzione di scheduling avanzata. Essa consiste nel visualizzare nel "calendario accademico" gli orari delle lezioni dell'insegnamento in questione. Per di più, l'utente può inserire per il corso selezionato degli eventi personalizzati delle seguenti tipologie: lezione/laboratorio (singola o ordinaria) o esame/task. Agli elementi presenti nella lista "I miei corsi" l'utente può associare un colore per rendere più facile la consultazione del calendario.

La funzione di scheduling avanzata consente per cui di utilizzare l'applicazione anche come personal planner, per organizzare lo studio e la frequentazione dei corsi.

L'elenco seguente riporta i dati disponibili sul sito www.unive.it utili per questa funzionalità:

- Insegnamento
  - Nome;

    - Titolo corso in inglese;
    - Anno accademico;
    - Codice insegnamento;
    - Crediti formativi universitari;
    - Livello laurea;
    - Settore scientifico disciplinare;
    - Periodo;
    - Anno corso;
    - Partizione;
    - Sede.
- Docenti
    - Nome e cognome;
    - Ore di lezione;
    - Materiali didattici (link).
- Corsi di laurea e percorsi
    - Codice e nome.
- Insegnamenti mutuati
    - Nome e codice.
- Orario delle lezioni
    - Giorno;
    - Ora;
    - Sede;
    - Aula;
    - Note.
- Programma

    - Obiettivi formativi;
    - Contenuti;
    - Testi di riferimento;
    - Modalità di verifica dell'apprendimento;
    - Prerequisiti;
    - Metodi didattici;
    - Lingue di insegnamento;
    - Altre informazioni.
- Altre parti di questo insegnamento
    - Nome insegnamento;
    - Tipo frequentazione.

Descrizione funzionalità Mappa

La funzionalità Mappa consiste in una mappa di Google Maps nei quali sono posti dei punti di interesse predefiniti, i quali indicano la posizione delle strutture dell'Ateneo. Quest'ultime possono essere, ad esempio, sedi, facoltà, aule, ecc. A queste si possono aggiungere i locali e le mense convenzionate con l'Università.

L'utente può visualizzare sulla mappa un punto di interesse specifico attraverso una ricerca oppure sfogliando le tipologie e le categorie; analogamente i *POI* possono essere filtrati direttamente sulla mappa.

Sfruttando il dispositivo GPS e l'applicazione Mappe di *iOS*, è possibile informare l'utente sulla sua posizione attuale e sulla distanza dal punto di interesse selezionato, oppure fornire le direzioni per raggiungerlo.

Ogni punto di interesse è accompagnato da una scheda che ne mostra i dettagli specifici, i collegamenti internet, le foto del luogo, ecc. I *POI* possono essere aggiunti ad

una lista dei preferiti, allo stesso l'utente può ritrovare quelli visti di recente in una lista dei recenti. L'utente può inoltre utilizzare i contatti disponibili per effettuare una telefonata o inviare un messaggio di testo.

L'elenco seguente riporta i dati disponibili sul sito www.unive.it utili per questa funzionalità:

- POI
    - Sedi;
    - Aule;
    - Facoltà;
    - Centri;
    - Biblioteche;
    - Dipartimenti;
    - Uffici amministrativi;
    - Wifi;
    - Sale rappresentanza.

Descrizione funzionalità Media

Costituisce un archivio che comprende foto e video. L'utente può consultare l'archivio attraverso una ricerca semplice, oppure scegliendo da una lista la categoria desiderata. In particolare le foto sono raggruppate in album e *feed*, mentre i video in canali. Sia le foto che i video possono avere una breve descrizione e riferirsi ad un evento.

La visualizzazione dei media rispecchia quella nativa del sistema *iOS*. Le funzioni sono supportate dalle *API* di YouTube e Flickr.

La lista dei preferiti e quella dei recenti non sono di importanza rilevante, per questo non sono presenti. Tuttavia, i link ai media possono essere eventualmente condivisi via e-mail.

L'elenco seguente riporta i gli album e i canali presenti su Flickr e YouTube:

- http://www.youtube.com/user/youcafoscari
- http://www.flickr.com/groups/unive/

Descrizione funzionalità Emergenza

E' una lista rapida di numeri di telefono utili o di emergenza. Sulla lista sono riportati i contatti accompagnati da una breve descrizione.

Descrizione funzionalità Social

Permette di visualizzare pagine e *feed* dai profili di Facebook e Twitter relativi all'Università. L'utente può scegliere il tipo di *social media* per avere una lista di tutti i network disponibili, suddivisi per categorie. La pagina del *social network* viene dunque visualizzata attraverso un browser *embedded*.

L'elenco seguente riporta alcuni dei social network individuati:

- Facebook
    - http://www.facebook.com/cafoscari
    - http://is-is.facebook.com/group.php?gid=34754124581&v=wall
    - http://www.facebook.com/informaticafoscari
    - http://www.facebook.com/pages/Venezia-Italy/CA-FOSCARI-ESPOSIZIONI/60587241084
    - http://www.facebook.com/teatrodicafoscari

- Twitter
    - http://mobile.twitter.com/cafoscari

Descrizione funzionalità iTunes

Costituisce un semplice link diretto ai podcast dell'Università. Causa l'apertura di iTunes.

Descrizione funzionalità Radio Ca' Foscari

Offre lo streaming in live e permette di accedere al sito web di Radio Ca' Foscari. Inoltre, consente di scaricare i *podcast* di RCF dall'applicazione iTunes.

Descrizione funzionalità Risorse

Rappresenta un archivio di URL che puntano a risorse sul web utili allo studente. I link possono essere consultati sfogliando delle categorie. Selezionando un elemento dalla lista risultante è possibile leggere una breve descrizione e aprire il link sul browser *Safari*.

Descrizione funzionalità Avvisi

Ultima, ma non meno importante, è la funzionalità Avvisi. Questa permette di ricevere avvisi e notifiche rilasciate dall'Università o in particolare relative al proprio dipartimento.

Per di più, se l'utente ha aggiunto degli insegnamenti alla lista "I miei corsi", verranno notificati tutti gli *alert* provenienti dai corsi e dai relativi docenti.

Gli avvisi sono visualizzati sia in ordine di pubblicazione, che per categorie. L'applicazione si sincronizza automaticamente con le origini delle notifiche. Se sono stati ricevuti degli aggiornamenti, l'applicazione lo fa sapere all'utente indicando il numero di avvisi ricevuti (*badge*).

L'elenco seguente riporta i dati disponibili sul sito www.unive.it utili per questa funzionalità:

- Avviso
    - Oggetto;
    - Data;
    - Posizione;
    - Descrizione.

REQUISITI NON FUNZIONALI

I requisiti non funzionali di un'applicazione sono relativi agli aspetti non direttamente legati alle sue funzionalità, ma che piuttosto riguardano proprietà e vincoli del software applicativo stesso. Infatti, fattori come costi, prestazioni, robustezza e compatibilità, alle volte gravano pesantemente sulla bontà del prodotto finale: trascurarli potrebbe risultare nello sviluppo di un prodotto scadente.

Performance

Questa sottosezione specifica i requisiti associati alla velocità con cui il sistema iCa'Foscari deve funzionare. Le caratteristiche di performance si basano sui tempi di risposta del sistema per una generica transazione; che consiste prima nel richiedere i dati alla sorgente, poi nello scaricarli sempre attraverso la rete ed infine nel presentarli sullo schermo.

I tempi di risposta dipendono particolarmente dalla quantità di dati trasmessi dal server e dalla banda di trasmissione offerta dalla connessione di rete. Tuttavia, si suppone

che l'utente sia a disposizione di una connessione a banda larga e di una buona ricezione del segnale.

Latenza

Specifica i requisiti riguardanti il tempo massimo consentito al sistema per eseguire dei task specifici, oppure per la terminazione di un caso d'uso:

- La ricerca in un qualsiasi sottosistema, per mezzo della barra di ricerca oppure sfogliando le categorie, deve essere completata dall'utente entro un tempo massimo di 10 minuti;
- L'utente deve poter consultare e modificare la lista dei preferiti o dei recenti entro un tempo massimo di 3 minuti;
- L'utente deve essere in grado di creare dei contenuti, ove consentito, in un tempo massimo di 10 minuti;
- Il sistema deve riuscire a caricare una pagina web, un *feed RSS*, oppure i dati da un'*API* esterna, in un tempo inferiore ai 2 minuti;
- Il sistema deve essere in grado di visualizzare un media (immagine o video), oppure avviare uno streaming audio in un tempo inferiore ai 3 minuti;
- Un sottosistema deve consentire all'utente di accedere alle funzionalità di competenza di un altro sottosistema entro un tempo massimo di 1 minuto;
- L'utente deve poter utilizzare la funzionalità emergenza e avviare una chiamata (se il dispositivo lo supporta) in un tempo inferiore ai 30 secondi.

Capacità

Identifica i requisiti riguardanti il numero minimo di oggetti che il sistema può supportare:

- La struttura dati del sistema deve supportare il *caching* di un minimo di 50 oggetti aggregati per ogni suo sottosistema;
- Il sistema non deve necessariamente effettuare il *caching* dei file multimediali;
- Le liste degli oggetti consultati di recente devono poter contare un massimo di 10 oggetti, mentre quelle dei preferiti un massimo di 25.

Qualità

I requisiti di qualità definiscono la misura in cui il software applicativo soddisfa un certo numero di aspettative rispetto, sia al suo funzionamento, sia alla sua struttura interna.

Compatibilità

Identifica i requisiti associati alla compatibilità dell'applicazione con la gamma di dispositivi Apple e la relativa versione di *iOS*:

L'applicazione deve poter essere utilizzata sui seguenti dispositivi *iOS*, presenti al momento sul mercato:

- Apple iPod Touch 2G e superiori;
- Apple iPhone 3G e 3GS;
- Apple iPhone 4 (*retina display*);
- Apple iPad (HD).

Il software dovrà essere sviluppato in ambiente *XCode*, utilizzando la versione più recente degli strumenti inclusi nell'*iOS SDK* rilasciata da Apple. Tuttavia, deve essere garantita la retro compatibilità alla versione 3.2.2 di *iOS*.

Riusabilità

Definisce i requisiti associati al grado con cui il sistema iCa'Foscari può essere utilizzato per scopi diversi da quelli intesi in origine (ad esempio, per la realizzazione di applicazioni analoghe su altre piattaforme):

- Il sistema deve incorporare un livello di database a disponibilità continua;
- Classi ed interfacce grafiche comuni devono essere realizzate nel modo più generico possibile in modo tale da poter essere riutilizzate.

Robustezza

Specifica i requisiti associati al grado con cui l'applicativo deve continuare a funzionare sotto circostanze anomale:

- Il sistema deve gestire input invalido (ad esempio rilevare l'input invalido, richiedere input valido e non bloccarsi);
- Il software deve saper gestire notifiche improvvise del sistema operativo, come "livello della batteria basso" o "memoria virtuale insufficiente";
- L'applicazione deve prevedere la migrazione dei dati esistenti tra una release e l'altra, qualora fossero state apportate delle modifiche alla struttura dati;
- Nel caso di assenza di connettività oppure di una caduta della connessione, il sistema non deve bloccarsi e deve avvisare l'utente dell'errore;
- Nell'eventualità che alcune informazioni non siano al momento disponibili dalla sorgente, oppure nel caso in cui i dati ricevuti siano corrotti, il software deve essere in grado di rilevare l'errore, avvisare l'utente di riprovare più tardi e tornare indietro alla vista precedente;
- Se l'applicazione non dovesse essere in grado di completare correttamente l'avvio, oppure se si dovesse verificare un errore nella persistenza, l'utente deve essere informato della situazione e gli deve essere richiesto di rilanciare

l'applicazione per consentirle di autoripararsi; resettando eventuali impostazioni e cancellando i dati corrotti.

Safety

Definisce i requisiti associati al grado con cui il sistema non deve direttamente o indirettamente (ad esempio, per inattività) causare danni accidentali o perdite di dati:

- Il sistema non deve perdere accidentalmente informazioni e modifiche create dall'utente;
- Il sistema non deve accidentalmente modificare gli oggetti che persistono nella sua struttura dati.

Scalabilità

Specifica i requisiti associati al grado di scalabilità dell'applicazione (ad esempio, gestire nuove funzionalità o memorizzare più informazioni nella sua struttura dati):

- Il sistema deve supportare un incremento nella dimensione del modello dati da gestire senza essere riprogettato;
- I tempi di risposta del sistema devono poter essere migliorati semplicemente fornendo al sistema stesso maggiori risorse di calcolo, memoria o una connessione più veloce;
- Il software deve poter accogliere nuove funzionalità richieste dal committente o nuove tecnologie introdotte nell'*SDK* senza dover essere riprogettato;
- La risoluzione dell'interfaccia grafica del sistema si deve riscalare correttamente a quella del dispositivo in uso e adattarsi alla definizione dei nuovi *retina display*;
- Il sistema deve adeguarsi correttamente nel caso dell'assenza di tecnologie HW nel dispositivo, come il telefono e il GPS.

- Il sistema deve supportare un incremento nella dimensione del modello dati da gestire senza essere riprogettato;

Validità

Indica i requisiti inerenti il livello di validità delle informazioni gestite dal sistema (ad esempio, l'eliminazione di dati obsoleti):

- Le informazioni presentate all'utente devono essere autentiche, ovvero prelevate esclusivamente dalle sorgenti gestite dall'Università Ca' Foscari;
- I dati gestiti dall'applicazione devono essere sincronizzati periodicamente con la sorgente dati, in modo da fornire all'utente informazioni sempre valide e aggiornate;
- I collegamenti alle pagine web e ai media esterni devono puntare ai contenuti prefissati ed essere sempre funzionanti;
- Il sistema deve provvedere automaticamente, secondo delle regole ben precise, all'eliminazione dei dati nella propria cache non sincronizzabili con la sorgente (quindi rimossi) e alla rimozione dei link non funzionanti;

Sicurezza

Questa sottosezione specifica i requisiti di sicurezza che consentono di definire in quale misura il sistema software deve proteggere se stesso, i suoi dati e le comunicazioni da accessi accidentali, dannosi o non autorizzati, e dall'uso, la modifica o la divulgazione non autorizzata dei dati provenienti dalla sorgente o creati dall'utente, compresa la distruzione di quest'ultimi.

Integrità

Di seguito sono definiti i requisiti di integrità che specificano i criteri di protezione dei dati e delle comunicazioni nei confronti di tentativi di corruzione intenzionali attraverso operazioni non autorizzate:

Comunicazioni: il 99,999% delle comunicazioni deve essere protetto da corruzioni intenzionali non autorizzate, incluse le comunicazioni con:

- Sorgente dati e Sistema Informativo centrale;
- *Web services* e *API* pubbliche.

Dati Persistenti: il sistema deve proteggere il 99,999% dei suoi dati persistenti da accessi e corruzioni non autorizzate.

Privacy

Il sistema deve assicurare l'utilizzo confidenziale di tutte le informazioni, sia quelle memorizzate che quelle comunicate, ad eccezione di quelle informazioni rese esplicitamente pubbliche da un requisito funzionale.

L'utente che installa l'applicazione potrà decidere di specificare alcuni dati per migliorare l'utilizzo di alcune funzionalità (per esempio di ricerca). Il sistema dovrà quindi garantire l'anonimità dell'utente rispetto alle informazioni riportate qui di seguito:

- Nome e cognome;
- Numero di matricola;
- Facoltà e corso di laurea.

Il software, inoltre, potrà avvalersi di un servizio di raccolta delle statistiche di utilizzo, come ad esempio *Google Mobile Analytics*. In questo caso, si applicheranno termini e condizioni specifiche per il trattamento di questi dati.

Accesso ad aree protette

Per il sistema iCa'Foscari non è attualmente previsto alcun livello di autenticazione dell'utente per l'accesso all'area riservata del sito www.unive.it, di conseguenza le funzionalità del sistema non hanno il diritto di accedervi.

L'accesso all'area riservata dovrà essere eseguito autonomamente dall'utente attraverso un browser che non dipenda dall'applicazione. Ogni tentativo di accesso dovrà essere rilevato causando la terminazione dell'applicazione e l'apertura del browser *Safari*; alternativamente, modifiche alla licenza d'uso potranno essere apportate per porre eccezioni sull'esecuzione di tale operazione da un browser integrato in iCa'Foscari, limitando le responsabilità del sistema in termini di sicurezza.

Usabilità

L'usabilità è definita dall'*ISO*, come l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono precisati obiettivi in particolari contesti. In pratica definisce il grado di facilità e soddisfazione con cui l'interazione uomo-sistema si compie.

Apple Human Interface Guidelines

Apple ha definito delle linee guida ben precise sulla progettazione delle interfacce umane di applicazioni per *iOS*.

Il *look and feel* di un software per *iOS* dovrebbe essere tale da dare l'impressione all'utente di utilizzare un'applicazione progettata specificatamente per il dispositivo. Gli utenti devono aver esperienza della stessa consistenza e delle stesse modalità di manipolazione che hanno reso i dispositivi *iOS* distinguibili da ogni altro nel loro genere; che sia la disposizione degli elementi grafici sullo schermo o la reazione del sistema alle *gesture* conosciute dagli utenti.

Il software applicativo dovrà dunque aderire alle specifiche definite da Apple[1] al fine di soddisfare le aspettative dell'utente e garantire un livello di usabilità adeguato.

Curva di apprendimento

L'applicazione deve garantire che almeno il 90% di un campione valido di utenti, con una buona esperienza di utilizzo dei dispositivi *iOS*, siano in grado di completare tutti i casi d'uso definiti per ogni funzionalità.

L'utente dovrà essere in grado di riconoscere fin da subito le funzionalità offerte dall'applicazione, senza istruzioni o tutorial iniziali. La curva di apprendimento deve essere immediata e può solamente dipendere dall'esperienza maturata dall'utente nell'utilizzo del dispositivo; senza fare alcuna distinzione tra utente novizio ed esperto.

Accessibilità

Il software iCa'Foscari dovrà supportare un'*API* inclusa in *iOS* chiamata *Universal Access*[2], che, assieme alle istruzioni fornite nelle linee guida sulla *Human Interface* discusse precedentemente, permette di integrare nell'applicazione un set di funzionalità addizionali come l'ingrandimento dello schermo, lo *screen reader*, e molte altre ancora, specificamente progettate per garantire accessibilità agli utenti con bisogni particolari.

Internazionalizzazione e localizzazione

La lingua principale dell'applicazione dovrà essere l'Inglese, mentre la seconda lingua, e anche la più importante, sarà l'Italiano. Il sistema dovrà essere predisposto al supporto di lingue e localizzazioni multiple, secondo le pratiche descritte in questo documento:

---

[1]http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/
[2]http://developer.apple.com/technologies/ios/accessibility.html

http://developer.apple.com/library/ios/#documentation/MacOSX/Conceptual/BPInternational/BPInternational.html

Se disponibile, la lingua del sistema sarà la stessa impostata nelle preferenze del dispositivo, altrimenti verrà scelto di default l'Inglese.

Behaviour

Il comportamento dell'applicazione dovrà rispettare le specifiche definite da Apple relativamente all'utilizzo delle tecnologie incluse nell'*SDK*. Ad esempio, la ricezione di notifiche *PUSH* o l'utilizzo del GPS devono essere espressamente autorizzati dall'utente prima di essere attivati.

Ultimo stato e impostazioni utente

Dovendo rispettare i requisiti di compatibilità, il sistema non supporterà, almeno non fin dall'inizio, il *multi-tasking*. Per ovviare a tale limitazione, l'applicazione dovrà gestire il salvataggio dell'ultimo stato di ogni sottosistema separatamente; ovvero quando l'utente esce da una funzionalità, il sottosistema che la gestisce salverà lo stato in cui si trova prima di essere deallocato. All'avvio dell'applicazione non verrà caricato lo stato dell'ultimo sottosistema utilizzato, bensì la schermata principale. L'ultimo stato di ogni sottosistema è caricato solamente quando l'utente seleziona la funzionalità ad esso corrispondente.

Tutte le impostazioni che devono essere modificate dall'utente ad applicazione avviata dovranno risiedere esclusivamente nell'applicazione stessa, in un'apposita sezione. Una entry nel pannello "Preferenze di sistema" del dispositivo, invece, potrà essere inserita solo nel caso di impostazioni utili al ripristino dell'applicazione, nel caso questa non dovesse riuscire a completare l'avvio.

Scorciatoie e ricerche

Per favorire una rapida reperibilità delle informazioni desiderate dall'utente, il software dovrà disporre di un motore di ricerca in comunicazione con la sorgente dati, presente già dalla schermata principale.

Inoltre, il sistema dovrà dare la possibilità all'utente di utilizzare delle scorciatoie per visualizzare le informazioni di suo maggiore interesse senza effettuare nuovamente da capo una ricerca.

Leggibilità dei contenuti

Ogni forma di contenuto gestito dal sistema dovrà essere presentata all'utente ricoprendo la porzione di schermo necessaria a garantirne un adeguato livello di leggibilità. Gli elementi grafici dell'*UI* di sistema, come ad esempio barre e pulsanti, non devono portar via eccessivo spazio ai contenuti, i quali rappresentano informazioni spesso di notevole importanza per l'utente.

Documentazione

Il prodotto finito dovrà essere corredato da almeno due tipi di documentazione:

- Documentazione di sistema: rappresenta la documentazione di riferimento per il team di progetto e lo staff del C.S.I.T.A.. Include la documentazione completa di tutte le classi, generata con gli appositi strumenti e la descrizione dettagliata di tutte le attività svolte dal sistema;
- Sito web dell'applicazione: rivolto agli utenti, offre una panoramica delle funzionalità e alcuni video tutorial sull'utilizzo dell'applicazione.

In osservanza dei requisiti sull'usabilità, l'utente non dovrà aver bisogno di consultare manuali o guide per poter fruire dell'applicazione; pertanto non è richiesta la fornitura di tale documentazione.

CONCLUSIONI

Nei paragrafi precedenti sono stati stabiliti i requisiti iniziali del progetto, necessari per lo sviluppo di iCa'Foscari.

Nella seconda parte di questo elaborato verranno definite ulteriori specifiche che riguardano, tra gli altri, gli strumenti e le metodologie di sviluppo, i formati e i mezzi di comunicazione con la sorgente dati, e la realizzazione della *GUI*.

PARTE SECONDA

Capitolo 5: Metodologie e strumenti

La seconda parte di questo lavoro di tesi fornisce una spiegazione dettagliata della progettazione di iCa'Foscari. I capitoli seguenti, infatti, descriveranno il design del sistema, della struttura dati e delle interfacce utente dell'applicazione. In questo capitolo, invece, sono brevemente presentate le principali metodologie di progettazione e gli strumenti disponibili, da impiegare nel processo di sviluppo del software applicativo.

METODOLOGIE

I capisaldi della progettazione e dello sviluppo di iCa'Foscari sono l'*Object Oriented Programming*[3], la modellazione *UML*[4] e l'utilizzo dei *design pattern*. Questi principi sono fortemente interconnessi tra loro e sono in parte imposti dall'ambiente di sviluppo stesso.

Infatti, il linguaggio primario dell'*API Cocoa* di Apple, su cui si basa *iOS*, è *Objective-C*. Esso è una versione orientata agli oggetti di *C* fortemente influenzata da *SmallTalk*, da cui ha ereditato lo stile di scambio dei messaggi. Un linguaggio di programmazione si dice orientato agli oggetti quando, appunto, utilizza delle strutture dati, costituite da campi e metodi, che interagiscono tra loro attraverso lo scambio di messaggi, il polimorfismo e l'ereditarietà.

La modellazione *UML*, invece, consiste nel rappresentare graficamente elementi come package, classi e associazioni che descrivono in maniera standard i casi d'uso o la progettazione di un'applicazione. Alcuni esempi di modelli realizzati secondo lo standard

[3]http://en.wikipedia.org/wiki/Object-oriented_programming
[4]http://en.wikipedia.org/wiki/Unified_Modeling_Language

UML 2.0 sono i diagrammi dei casi d'uso, il diagramma delle classi, lo schema relazionale della base di dati e i diagrammi di sequenza.

Infine, la progettazione di un'applicazione per *iOS* verte necessariamente sul paradigma *Model View Controller*. Ciò è dovuto al fatto che lo stesso *framework Cocoa* è stato sviluppato secondo questo *design pattern*. A quest'ultimo si aggiungono altri pattern spesso comuni nella programmazione a oggetti e altri invece che dipendono direttamente dal linguaggio *Objective-C*. Maggiori dettagli sui pattern di progettazioni sono trattati nel capitolo sulla progettazione di sistema.

STRUMENTI

Il *Software Development Kit* per *iOS* rilasciato da Apple[5] include tutti gli strumenti necessari per lo sviluppo di iCa'Foscari. L'ambiente di sviluppo richiesto per l'utilizzo del *SDK* comprende l'ultima versione di Mac OS X e un iPhone sul quale effettuare il *debugging* dell'applicazione.

Di seguito è riportato un elenco degli strumenti più importanti nello sviluppo di un'applicazione per *iOS*:

- **XCode IDE**: integra un editor professionale e un compilatore ottimizzato per il *framework Cocoa*. Esegue il *debugging* sul simulatore oppure direttamente su un dispositivo collegato via USB;
- **Interface Builder**: consente la costruzione visuale delle interfacce grafiche di tutti i dispositivi *iOS*. Integra il *framework UIKit* e *l'API Universal* Access per l'utilizzo degli elementi grafici nativi e l'accesso facilitato. Supporta inoltre la localizzazione del testo inserito nell'interfaccia;

[5]http://developer.apple.com/technologies/

- **Instruments**: rappresenta un'applicazione per l'analisi delle performance, capace di raccogliere in tempo reale dati sull'utilizzo di memoria, CPU e connessioni di rete. I dati raccolti sono poi visualizzati graficamente, consentendo di individuare *memory leak* e *hot spot* nelle sorgenti di codice;
- **iPhone Simulator**: permette di emulare su Mac OS X un dispositivo *iOS* a scelta tra iPhone 3G, iPhone 4 e iPad, sul quale eseguire l'applicazione in fase di sviluppo ed effettuare il *debugging*;
- **genstrings**: eseguito da terminale, si occupa di generare dei file con estensione .strings, contenenti tutte le stringhe dichiarate localizzabili nel codice con la funzione *NSLocalizedString(key, comment)*. Le stringhe contenute nel file possono poi essere tradotte nella lingua specifica.

Lo stile grafico e le icone dell'applicazione dovranno essere realizzati attraverso strumenti di grafica *raster* o preferibilmente *vettoriale*, come ad esempio *Adobe Photoshop*, *Adobe Illustrator* oppure *InkScape*. I prototipi delle interfacce utente possono essere disegnati sempre con *Photoshop*, oppure con *Omnigraffle*, utilizzando degli *stencil* della *GUI* di *iOS* disponibili ai seguenti indirizzi:

- http://www.smashingmagazine.com/2008/11/26/iphone-psd-vector-kit/
- http://www.graffletopia.com

LIBRERIE E FRAMEWORK AGGIUNTIVI

L'elenco seguente rappresenta una lista di librerie, *API* e *framework open source* ritenute utili per lo sviluppo di iCa'Foscari:

- **Facebook Connect**: fornisce un metodo di autenticazione affidabile per i servizi offerti da Facebook, consentendo di integrare Facebook Connect nell'applicazione;
- **FeedParser**: è un *NSXMLParser* basato su un *parser RSS/Atom* per *Cocoa*;
- **FTUtils**: consiste nella classe *FTAnimation* che contiene delle macro per animazioni *Core Graphics*;
- **Google API Client Library**: sono delle *API* scritte in *Objective-C* per l'accesso ai offerti da Google, tra cui Mappe, Docs, YouTube e Analytics;
- **JSON Framework**: implementa un *parser* e un generatore di codice *JSON* in *Objective-C*;
- **OAuth Consumer**: implementa in *Objective-C* il protocollo *OAuth* per l'accesso alle risorse protette di un web service, come Google ad esempio, attraverso un *API*, senza che l'utente riveli le proprie credenziali all'applicazione;
- **ObjectiveFlickr**: permette l'accesso alle *API* di Flickr dai dispositivi *iOS*;
- **Tapku Library**: fornisce un'estensione dei *framework Cocoa Foundation* e *UIKit*. Il *framework* contiene elementi di interfaccia grafica come cover flow, calendari e grafici. Altre funzionalità utili includono nuove *Table View Cell*, la creazione semplificata di *request HTTP Post* e l'estensione di notifiche per il *device shaking*;
- **Three20**: è una libreria che comprende una ricca collezione di classi utilizzabili per vari scopi. Three20 è utilizzata dall'applicazione ufficiale di Facebook per iPhone e da molte altre applicazioni nell'App Store.

Quando si utilizza del codice sorgente sviluppato da altri è necessario accertare che questo non effettui delle chiamate a delle *API* private non autorizzate da Apple. Ciò infatti impedirebbe l'ottenimento dell'approvazione di Apple per la distribuzione sull'App Store. Per ovviare a tale problema, è possibile utilizzare *App Scanner* di *Chimp Studios*, il quale permette di individuare proattivamente eventuali *API* private nel codice sorgente.

VERSIONE DELL'AMBIENTE DI SVILUPPO DEL PROTOTIPO

Il prototipo dimostrativo di iCa'Foscari, presentato assieme a questo lavoro di tesi, è stato sviluppato su Mac OS X 10.6.4, adoperando la versione 3.2.3 di XCode inclusa nel *SDK* di *iOS 4.0.1*. In particolare la versione del compilatore è GCC 4.2, mentre i parametri di *build* hanno *iOS* 4.0 come *Base SDK* e 3.2 come *Deployment Target*. Nella tabella seguente sono riportate alcune informazioni che riguardano invece l'utilizzo delle librerie importate nel progetto:

Libreria	Dettagli	Licenza
FeedParser	Utilizzato dalla TickerBar per il parsing del feed RSS degli avvisi di Ca'Foscari.	MIT
FTUtils	Animazione di TTLauncherView nel caricamento della schermata home e animazione di altri oggetti UIView.	MIT
JSON Framework	Parsing del codice JSON impiegato nella funzionalità Directory.	BSD
TapkuLibrary	Oggetti TKEmptyView e TKLoadingView.	Apache 2.0
Three20	Riproduzione della SpringBoard con TTLauncherView.	Apache 2.0

Tabella 5.1: Librerie utilizzate nel prototipo di iCa'Foscari.

Il prototipo è stato testato su iPhone 3G con la versione 4.0.1 del *firmware*, senza *jailbreak*, mediante un *Provisioning Profile* certificato da Apple[6]:

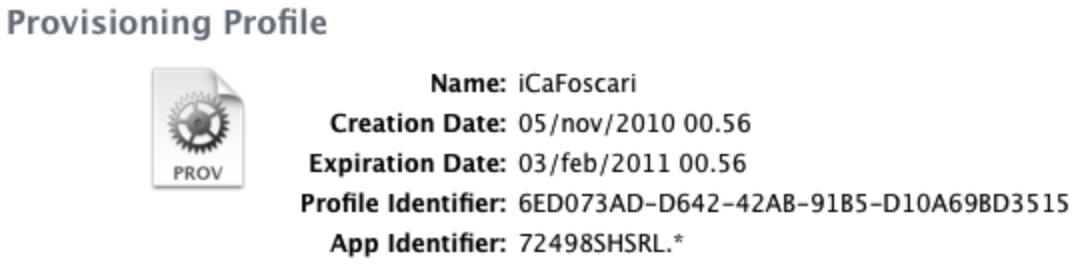

Figura 5.1: Provisioning Profile del prototipo di iCa'Foscari.

CONCLUSIONI

Nei paragrafi precedenti sono state definite sinteticamente le metodologie impiegate nella progettazione presentata in questo elaborato. Per di più è stata fornita una lista degli strumenti che serviranno nella fase di sviluppo dell'applicazione. Ulteriori informazioni su quanto trattato in questo capitolo possono essere reperite dai riferimenti a piè di pagina; specie per quanto riguarda le tecnologie offerte dal *SDK* di *iOS*, le quali ci si è riservati di approfondire.

[6]http://developer.apple.com/library/ios/#documentation/Xcode/Conceptual/iphone_development/128-Managing_Devices_and_Digital_Identities/devices_and_identities.html

Capitolo 6: Progettazione di sistema

In questa sezione viene descritta la progettazione di sistema di iCa'Foscari, definendo i principali design pattern utilizzati e fornendo la scomposizione in sottosistemi.

DESIGN PATTERN

Nell'ambiente di programmazione *Cocoa* esistono la maggior parte dei design pattern introdotti e catalogati nel libro *Design Patterns: Elements of Reusable Object-Oriented Software* pubblicato nel 1994. Ciò nonostante in *Cocoa* questi pattern sono implementati in maniera distinta, in quanto essi sono fortemente influenzati da fattori quali caratteristiche del linguaggio *Objective*-C o architetture esistenti.

Di seguito è riportata una lista completa dei pattern disponibili in *Cocoa* per *iOS*:

- Abstract Factory (Class Cluster);
- Adapter (Protocol);
- Block objects;
- Chain of Responsibility (Responder Chain);
- Command (Invocation Objects);
- Composite (View Hierarchy);
- Decorator (General Comments, Delegation, Categories);
- Facade (NSImage);
- Iterator (Enumerator);
- Managed memory model (sostituisce la Garbage Collection);
- Meccanismo Target-Action;
- Mediator (Controller in Application Kit, View Controller in UIKit);

- Memento (Archiving, Serializzazione Property List, Core Data);
- Observer (Notifiche, Key –Value Observing);
- Proxy (NSProxy);
- Singleton (alcune classi dei Framework);
- Template (Ovverride dei metodi dei Framework).

La programmazione *Cocoa* in particolare è fortemente orientata all'utilizzo di due pattern fondamentali: *Model View Controller* e *Object Modeling*. Il primo è un pattern composto o aggregato, cioè che si basa su un catalogo di pattern. Il secondo invece ha origine dal dominio dei database relazionali.

La progettazione di iCa'Foscari riflette dunque l'utilizzo di questi due principi di progettazione propri di *Cocoa* e dell'*SDK* di *iOS*, che andremo a descrivere nei paragrafi seguenti.

Model-View-Controller (MVC)

Il pattern di progettazione MVC, spesso chiamato "paradigma", è utilizzato sin dai tempi di *Smalltalk*. È un pattern ad alto livello, in quanto rappresenta l'architettura globale di un'applicazione e classifica gli oggetti secondo il proprio ruolo giocato in un'applicazione. È anche un pattern composto, poiché comprende molti altri pattern più elementari. I programmi *Object-oriented* traggono numerosi benefici nell'adottare il paradigma MVC, in particolare gli oggetti tendono ad essere più riutilizzabili, il programma è molto più adattabile alle modifiche nei requisiti, e in generale vi è una maggiore estensibilità rispetto ai programmi che non si basano su MVC.

Il pattern MVC prevede che vi siano tre tipi di oggetti: oggetti modello, oggetti vista e oggetti controllore. Il pattern definisce i ruoli svolti nell'applicazione da questi tipi di oggetti e i loro canali di comunicazione.

- *Oggetti modello, incapsulano i dati e i comportamenti di base*: rappresentano conoscenze e competenze specifiche di un problema. Racchiudono tutte le informazioni importanti di un'applicazione e definiscono la logica che manipola tali informazioni;
- *Oggetti vista, presentano le informazioni all'utente*: un oggetto vista ha le conoscenze necessarie per presentare i dati ed eventualmente consentire all'utente di modificarli. Non è però responsabile di immagazzinare i dati visualizzati. Il suo ruolo fondamentale è quello di visualizzare un intero oggetto modello, oppure solamente una sua parte. Gli oggetti vista tendono ad essere riutilizzabili e configurabili, e forniscono consistenza tra le applicazioni;
- *Oggetti controllore, legano il modello alla vista*: un controllore agisce essenzialmente da intermediario tra gli oggetti vista e gli oggetti modello di un'applicazione. Spesso il loro compito è quello di assicurare che le viste abbiano accesso agli oggetti modello che devono presentare e agiscono da condotto attraverso il quale le viste vengono a conoscenza delle modifiche di un modello. Gli oggetti controllore svolgono, inoltre, altri compiti di set-up e coordinamento; gestiscono il ciclo di vita di altri oggetti e possono essere riutilizzabili o meno, a seconda del tipo di controllore *Cocoa* implementato (in *iOS* sottoclassi di *UIViewController*).

Come già anticipato MVC è un pattern composito. Nella sua versione tradizionale (*Smalltalk)*, infatti, è composto dai pattern Composite, Strategy e Observer, come rappresentato in figura:

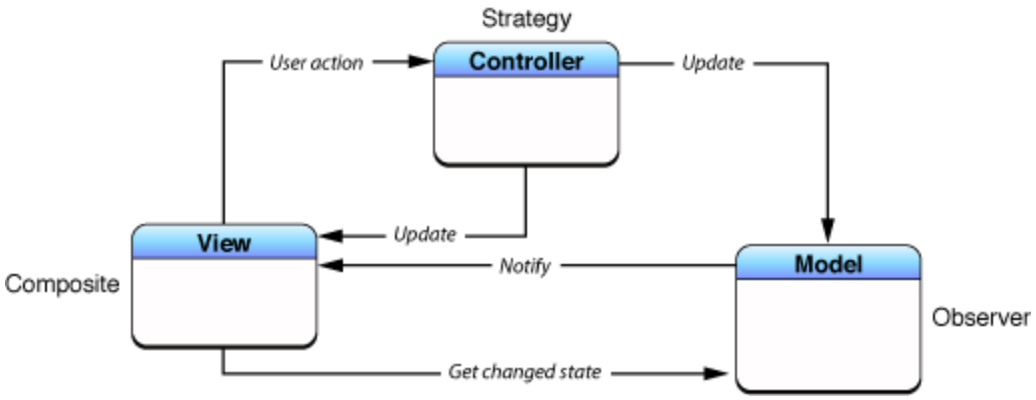

Figura 6.1: Versione tradizionale di MVC.

Tuttavia, la versione *Cocoa* di MVC è diversa da quella tradizionale, nel fatto che è composta da pattern differenti. Nonostante ciò vi sono comunque delle somiglianze con la versione tradizionale ed è di fatto possibile costruire un'applicazione funzionante basata sul diagramma nella figura precedente.

Purtroppo, però, sorge un problema teorico nell'utilizzo di questo design. Gli oggetti vista e modello dovrebbero essere gli oggetti maggiormente riutilizzabili nell'applicazione. Le viste rappresentano il *"look and feel"* di un sistema operativo e le applicazioni da esso supportate; secondo le specifiche Apple in *Cocoa* la consistenza nell'aspetto e nel comportamento tra le applicazioni è essenziale, e richiede oggetti altamente riusabili. Per di più, i modelli per definizione incapsulano i dati associati ad uno specifico problema e svolgono delle operazioni su di essi. A rigor di progettazione dunque, è meglio mantenere gli oggetti vista e modello separati l'uno dall'altro, in quanto ciò ne aumenta la riusabilità.

Nella maggior parte delle applicazioni *Cocoa*, le notifiche sui cambiamenti di stato nei modelli sono comunicate alle viste attraverso i controllori. La figura seguente

mostra questa diversa configurazione, che appare molto più chiara nonostante siano coinvolti altri due pattern elementari (Mediator e Command).

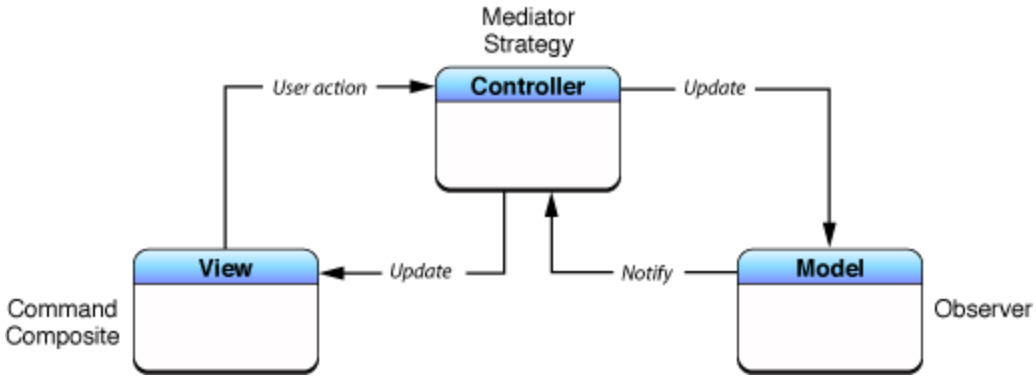

Figura 6.2: Versione *Cocoa* di MVC.

Il controllore in questa versione incorpora il pattern Mediator, così come il pattern Strategy; esso svolge da mediatore del flusso dei dati tra il modello e la vista in entrambe le direzioni. Gli oggetti vista incorporano, invece, il pattern Command nella loro implementazione del meccanismo target-action.

Un'applicazione MVC *Cocoa* ben progettata, fa solitamente uso di un *Coordinating Controller*, il quale "possiede" dei propri controllori di mediazione, archiviati in file *nib*, come rappresentato in figura:

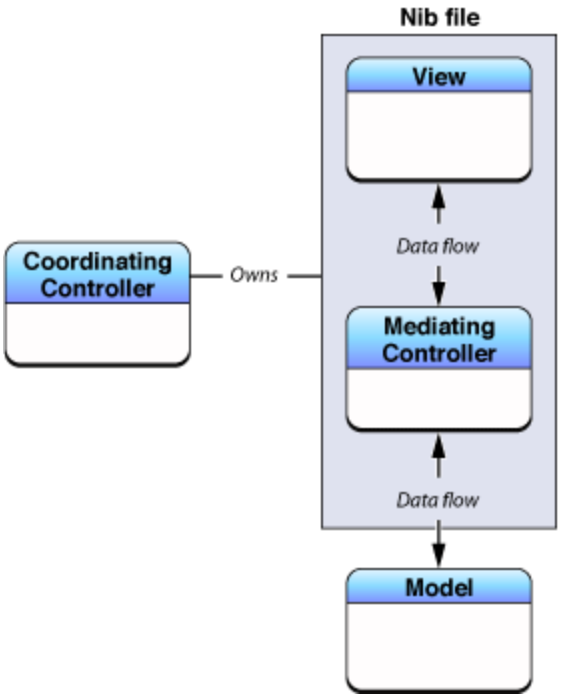

Figura 6.3: *Coordinating Controller* come proprietario di un file nib.

In questo modo è possibile allo stesso tempo implementare il pattern Mediator e sfruttare il meccanismo target-action attraverso la *Responder Chain*; cosa non possibile utilizzando solamente dei controllori di mediazione. Tutto ciò senza creare delle sottoclassi personalizzate di viste per rispondere alle notifiche, ma bensì utilizzando file *nib* creati con *Interface Builder*.

Come abbiamo visto, l'ambiente *Cocoa* impone un approccio differente nella progettazione in MVC. A questo si aggiungono delle importanti linee guida definite da Apple, reperibili al seguente indirizzo:

http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html#//apple_ref/doc/uid/TP40002974-CH6-SW23

Object Modeling

Object Modeling, assieme ad MVC, rappresentano le metafore più importanti e maggiormente pervasive in *Cocoa*, e sono in larga misura strettamente correlate tra loro.

Utilizzando il framework *Core Data* (disponibile su *iOS* dalla versione 3.0), è necessario avere un modo per descrivere i propri oggetti modello che non dipenda sulle viste o i controllori. In una buona progettazione, infatti, le viste e i controllori hanno bisogno di una maniera per accedere le proprietà del modello senza imporre delle dipendenze tra loro. *Core Data* risolve questo problema prendendo in prestito dei concetti e dei termini propri della modellazione entità-relazione.

La modellazione entità-relazione è un modo di rappresentare gli oggetti, non esclusivo a *Cocoa*, tipicamente utilizzato per descrivere le strutture dati di una sorgente dati, come ad esempio un database o un file, in maniera tale da consentire alle strutture dati stesse di essere mappate negli oggetti di un sistema *object-oriented*.

Cocoa, ancora una volta, utilizza una versione modificata delle regole tradizionali della modellazione entità-relazione, che prende il nome di Object Modeling; il quale è spesso utilizzato anche nelle applicazioni MVC più semplici, in quanto i modelli tipicamente sono persistenti. Il pattern Object Modeling, così come il modello entità-relazione, è costituito dagli elementi seguenti:

- *Entità*: rappresentano niente meno che i modelli del pattern MVC. I componenti di un'entità sono chiamati *attributi*, e i riferimenti agli altri modelli sono chiamati *relazioni*. Insieme, questi sono noti anche come *proprietà*;
- *Attributi*: rappresentano strutture che contengono dati; come ad esempio, uno scalare, una struttura C oppure un'istanza di una classe primitiva. In *Cocoa*, un attributo tipicamente corrisponde ad una variabile d'istanza o a un metodo d'accesso di un modello;

- *Relazioni*: un'applicazione solitamente è modellata da un insieme di classi. A runtime, il modello di un'applicazione è una collezione di oggetti in relazione fra loro che formano un *object graph*. Le relazioni tra questi modelli può essere percorsa a tempo di esecuzione per accedere alle proprietà degli oggetti correlati. Le relazioni possono essere riflessive, bidirezionali o unidirezionali (*Core Data*, tuttavia, impone che per ogni relazione esista anche la sua inversa, dunque sono quasi sempre bidirezionali). La cardinalità di una relazione, invece, può essere a-uno oppure a-molti, a seconda del numero di oggetti a destinazione. Le relazioni possono essere dichiarate obbligatorie oppure opzionali.

L'accesso alle proprietà di un'entità è garantito, in maniera indiretta e automatica, dal meccanismo di *key-value-coding*, che agisce utilizzando il nome delle proprietà di un oggetto come chiave per accedere ai rispettivi valori. In particolare, il valore di una relazione a-uno è semplicemente l'oggetto destinatario della relazione, il valore di una relazione a-molti è, invece, un oggetto collezione che contiene gli oggetti destinatari della relazione.

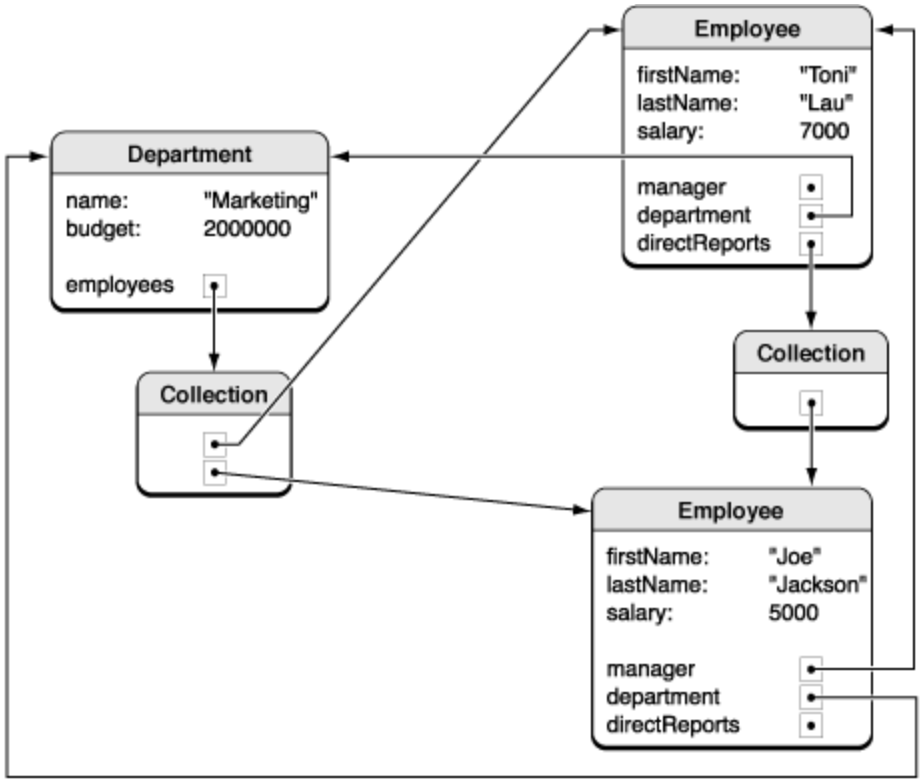

Figura 6.4: Esempio di *object graph*.

Entrambi il pattern MVC e la metafora Object Modeling sono stati impiegati nella progettazione di iCa'Foscari. Nei paragrafi seguenti è illustrata l'architettura e la scomposizione in sottosistemi dell'applicazione, mentre la struttura del modello dati sarà trattata separatamente al capitolo successivo.

ARCHITETTURA DELL'APPLICAZIONE

Ogni applicazione sviluppata per *iOS* presenta degli oggetti chiave che svolgono dei ruoli specifici nel ciclo di vita dell'applicazione. Dal momento in cui l'applicazione viene lanciata dall'utente, fino alla sua chiusura, il framework *UIKit* gestisce gran parte della sua infrastruttura chiave. Tra questi componenti fondamentali ve ne sono alcuni, detti "di sistema", che si occupano, ad esempio, di ricevere continuamente gli eventi dal

sistema stesso, ma è responsabilità del proprio codice di rispondere a queste notifiche. La figura seguente illustra l'architettura standard di un'applicazione *iOS*, adottata anche da iCa'Foscari.

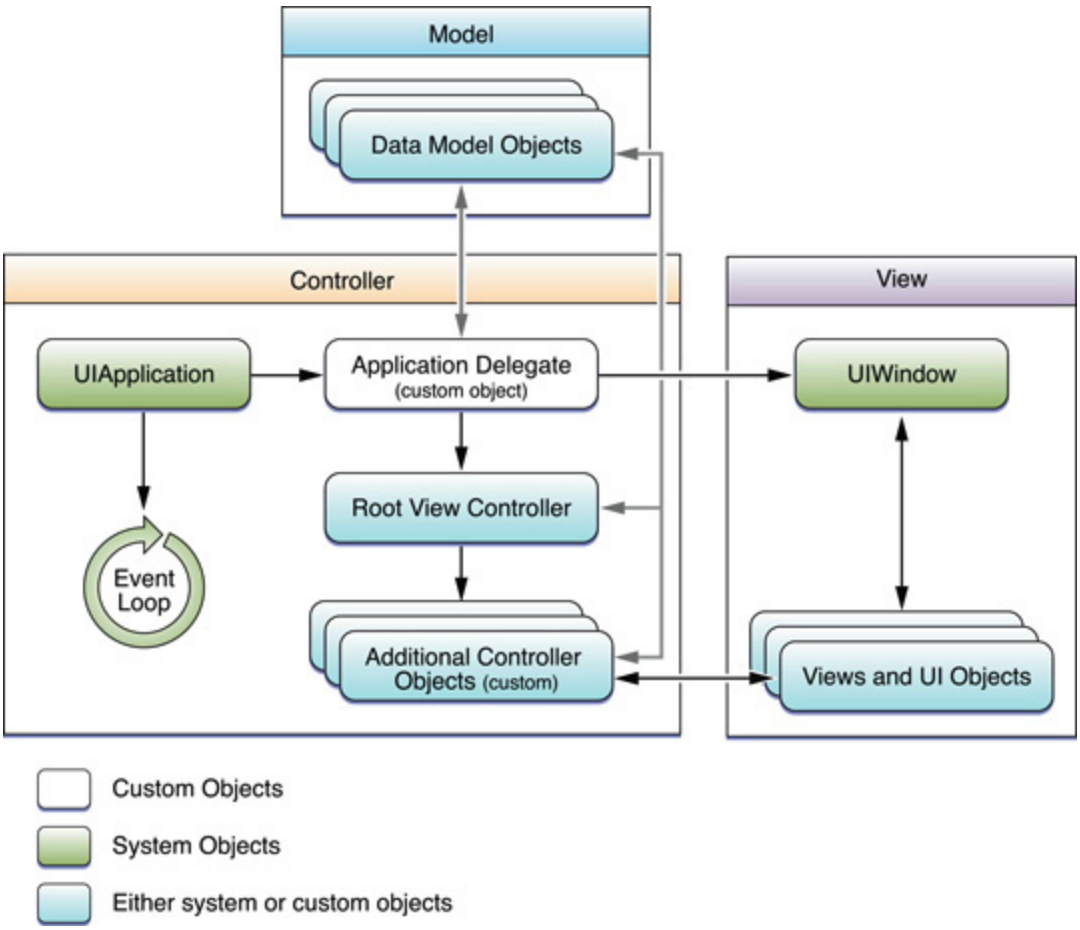

Figura 6.5: Architettura di un'applicazione *iOS*.

Come si nota dalla figura, l'architettura rispetta il paradigma MVC e la sua infrastruttura corrisponde ad un complesso ecosistema di oggetti che hanno delle competenze comuni a tutte le applicazioni *iOS*, la cui descrizione è riportata sinteticamente nella tabella seguente:

Oggetto/i	Descrizione
UIApplication	Gestisce il ciclo degli eventi dell'applicazione e coordina i suoi comportamenti ad alto livello. Viene utilizzato soprattutto per configurare l'aspetto dell'applicazione. È un oggetto di sistema e lavora in coppia con l'Application delegate, nel quale risiede il codice personalizzato vero e proprio.
Application delegate	Rappresenta un oggetto personalizzato da fornire all'applicazione a tempo di esecuzione, integrandolo nel file *nib* principale. Lo scopo principale di quest'oggetto è di inizializzare l'applicazione e presentare su schermo la sua finestra. Riceve notifiche dall'oggetto UIApplication quando si verificano degli eventi specifici (ad esempio, pressione del tasto Home).
Data model	Sono degli oggetti specifici per il modello dell'applicazione e hanno il compito di immagazzinare i suoi contenuti.
View controller	Hanno il compito di caricare le View necessarie per la presentazione dei contenuti sullo schermo, e coordinano le interazioni con gli oggetti data model. La classe *UIViewController* è la classe di base per tutti gli oggetti di questo tipo. Fornisce funzionalità di default come l'animazione delle View e la rotazione dello schermo. Solitamente questa classe viene estesa per svolgere delle funzionalità specifiche.
UIWindow	Amministra la superficie di disegno dell'applicazione, sulla quale viene caricato un nuovo insieme di View, attraverso un View controller. Inoltre, ha anche il compito di distribuire gli eventi a quelle View e ai loro View controller.
View, control e layer	Forniscono una rappresentazione visuale del contenuto dell'applicazione. Le View disegnano una parte del contenuto nell'area designata e rispondono agli eventi che hanno luogo in quell'area. I Control sono tipi specializzati di View responsabili per l'implementazione di oggetti grafici comuni come pulsanti e text field. Oltre agli oggetti standard forniti da *UIKit* è possibile estendere la classe *UIView* per definire delle View personalizzate. I Layer si occupano invece di renderizzare dietro le quinte i contenuti delle View.

Tabella 6.1: Ruoli degli oggetti in un'applicazione *iOS*.

Ruoli specifici degli oggetti chiave in iCa'Foscari

Come annunciato un'applicazione *iOS* deve fornire il proprio codice per rispondere correttamente agli eventi del sistema e presentare opportunamente i contenuti sullo schermo. In questo paragrafo verrà ora discusso il ruolo specifico degli oggetti principali richiesti dal sistema all'avvio di iCa'Foscari.

- *Application delegate (iCaFoscariAppDelegate)*: in iCa'Foscari l'Application delegate svolge delle funzioni comuni in tutta l'applicazione; come ad esempio l'apertura di URL, la composizione di e-mail, le chiamate telefoniche o l'apertura di un'applicazione esterna di sistema. Attività che possono essere invocate da qualunque funzionalità dell'applicazione. L'iCaFoscariAppDelegate, inoltre, si occupa di coordinare lo stack di *Core Data* e passare il contesto di persistenza a tutte le funzionalità. Oltre a ciò, ha la responsabilità di definire lo stile comune per ogni *UITableViewCell* delle *UITableView* utilizzate. Infine, ha la responsabilità di rispondere alle notifiche *PUSH*, di gestire il ciclo di vita dell'oggetto adibito allo streaming audio proveniente dalla web radio e di eseguire eventuali set up della view principale specifici per iPad;
- *RootViewController (MainScreen)*: Instanziato dall'AppDelegate all'invocazione di *applicationDidFinishLaunching*, ha la responsabilità di caricare la *MainScreenLauncherView* (sottoclasse di *TTLauncherView)*, che presenta tutte le funzionalità dell'applicazione emulando lo stile della *SpringBoard*. MainScreen si occupa, inoltre, di avviare la *TickerBar* e aggiungerla alla view. Per di più, essendo il RootViewController, ha il compito di caricare le funzionalità selezionate dall'utente, presentandole attraverso un *Modal View Controller*. Infine, definisce la logica di rotazione dello schermo e lo stile della *Status Bar*.

(Ulteriori dettagli sulla presentazione delle view verranno discussi più avanti nel capitolo inerente i prototipi dell'*UI*).

SCOMPOSIZIONE IN SOTTOSISTEMI

Nelle pagine seguenti sono riportati dei diagrammi che illustrano l'organizzazione, secondo il pattern MVC, delle funzionalità di iCa'Foscari (ad esclusione delle funzionalità Emergenza e iTunesU, ritenute trascurabili). L'insieme dei diagrammi proposti rappresenta per cui la scomposizione in sottosistemi dell'applicazione, riassumibile nel grafico seguente:

Figura 6.6: Scomposizione in sottosistemi.

Per ciascuno dei diagrammi seguenti è riportata una tabella che spiega il ruolo dei controllori e dei modelli illustrati, ad esclusione delle viste, che verranno descritte nel capitolo inerente l'*UI*. Allo stesso modo, la descrizione funzionale e i casi d'uso di ogni sottosistema sono consultabili nella sezione relativa i requisiti funzionali.

Sottosistema News

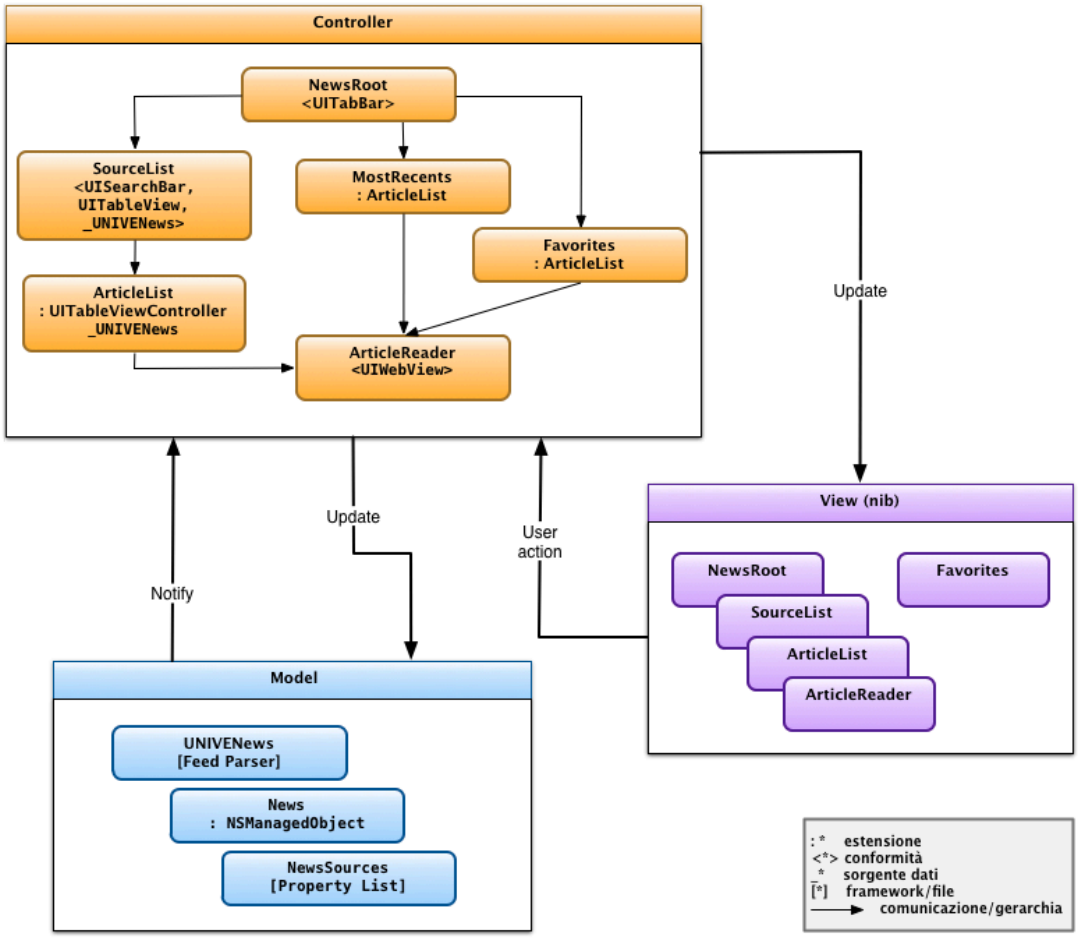

Figura 6.7: Sottosistema News.

Controller	Descrizione
NewsRoot	Rappresenta il RootViewController di questo sottosistema. Gestisce una TabBar che controlla la presentazione delle fonti delle notizie, delle notizie più recenti e degli articoli preferiti dall'utente.
SourceList	Il suo ruolo è quello di richiedere al modello le fonti disponibili per le notizie e di presentarle in una TableView. Gestisce inoltre le ricerche effettuate su tutti i contenuti dalla barra di ricerca. Effettuando una ricerca o selezionando una fonte, si occuperà di caricare il controller responsabile per mostrarne i relativi contenuti.
ArticleList	Sempre attraverso una TableView, elenca gli articoli della fonte selezionata in ordine cronologico. Permette di aggiornare la lista corrente e di caricare gli articoli più vecchi.
ArticleReader	Si occupa di visualizzare la notizia selezionata da una lista di articoli, e di caricare quella precedente o successiva in ordine cronologico. Coordina l'inserimento dell'articolo nei preferiti, e consente di condividerlo o visualizzarlo in forma integrale.
MostRecents	Eredita le funzioni di ArticleList e si occupa di mantenere una lista degli articoli inseriti più di recente nelle fonti disponibili.
Favorites	Mantiene una lista dei preferiti aggiunti dall'utente, consentendone la rimozione.

Tabella 6.2: Ruoli degli oggetti controller nel sottosistema News.

Model	Descrizione
UNIVENews	Utilizza il framework *Feed Parser* per scaricare le news dalle fonti disponibili, instanziandole nelle classi definite nel modello dati dell'applicazione.
News	È il modello vero e proprio di questo sottosistema e rappresenta in memoria un articolo per tutti i controller che lo utilizzano.
NewsSources	Fornisce le informazioni necessarie per collegarsi alle fonti di notizie disponibili. Viene utilizzato al primo avvio del sottosistema da UNIVENews, che si occupa di salvare il suo contenuto nel modello News_Source con *Core Data*, velocizzando gli avvii successivi.

Tabella 6.3: Ruoli degli oggetti model nel sottosistema News.

Sottosistema Eventi

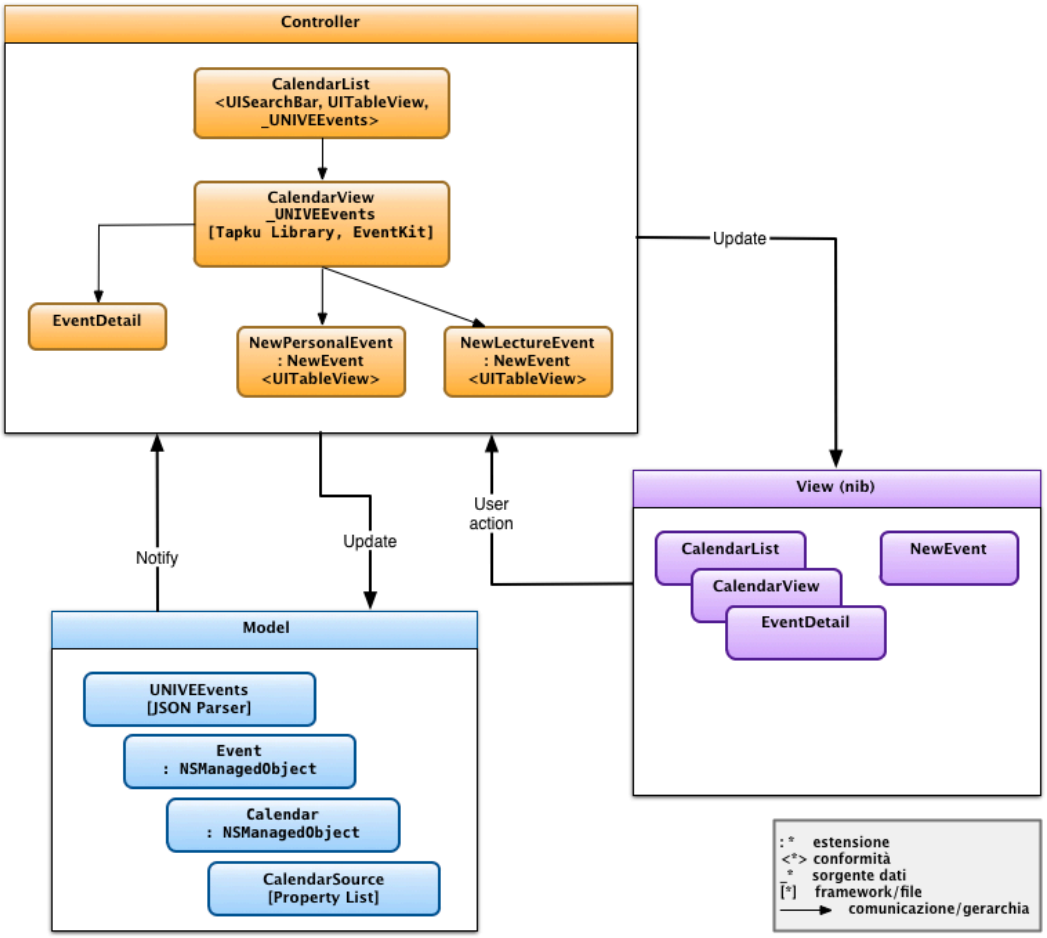

Figura 6.8: Sottosistema Eventi.

Controller	Descrizione
CalendarList	Consente la ricerca di eventi nei calendari disponibili, quest'ultimi elencati in una TableView.
CalendarView	Il suo compito è quello di rappresentare gli eventi del calendario selezionato dall'utente, oppure i risultati di una ricerca, in tre modalità differenti: lista, giorno e mese. Consente di visualizzare gli eventi del giorno corrente e di muoversi avanti e indietro di

	giorni o mesi. Permette, inoltre, di creare un nuovo evento (a seconda del calendario selezionato) o di visualizzare i dettagli di uno esistente, passando il controllo all'oggetto responsabile.
EventDetail	Si occupa di presentare i dettagli dell'evento selezionato. Comunica con altri sottosistemi o applicazioni esterne per aprire URL, visualizzare indirizzi sulla mappa e utilizzare i recapiti disponibili.
NewPersonalEvent	Coordina l'inserimento nel calendario di un nuovo evento rappresentato dal modello Personal_Event.
NewLectureEvent	Analogamente al precedente, coordina l'inserimento nel calendario di un nuovo evento rappresentato dal modello Lecture_Event.

Tabella 6.3: Ruoli degli oggetti controller nel sottosistema Eventi.

Model	Descrizione
UNIVEEvents	Ha la responsabilità di scaricare in formato JSON gli eventi dei calendari, instanziandoli nelle classi definite nel modello dati dell'applicazione.
Event	Rappresenta nell'applicazione un oggetto contenente tutte le informazioni di un evento. Ha delle sottoclassi ed è utilizzato dai controller per presentare i suoi dati.
Calendar	Costituisce in memoria un insieme di eventi correlati e fornisce ai controller le informazioni necessarie per sincronizzarlo.
CalendarSource	Rappresenta una lista dei calendari disponibili e l'indirizzo della sorgente di dati JSON necessaria per sincronizzarli. Viene utilizzato al primo avvio del sottosistema da UNIVEEvents, che si occupa di salvare il suo contenuto nel modello Calendar con *Core Data*, velocizzando gli avvii successivi.

Tabella 6.4: Ruoli degli oggetti model nel sottosistema Eventi.

Sottosistema Directory

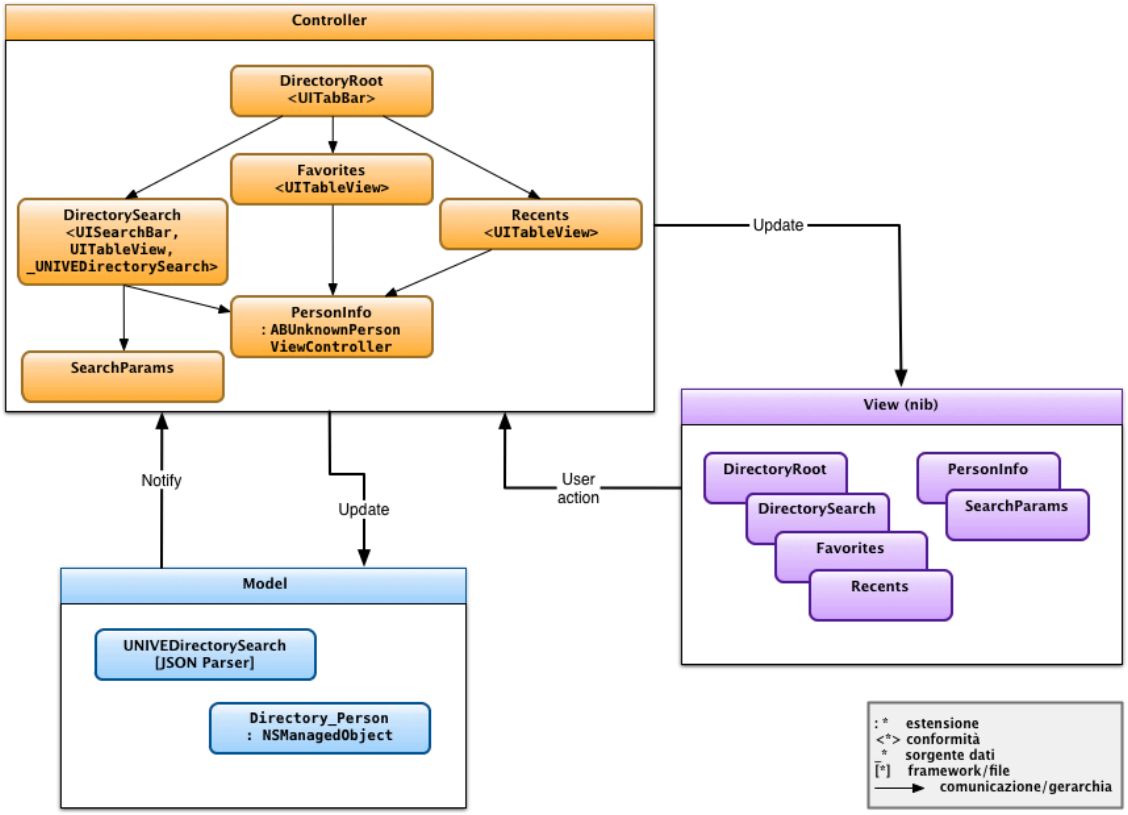

Figura 6.9: Sottosistema Directory.

Controller	Descrizione
DirectoryRoot	Gestice, per mezzo di una TabBar, altri controller per la ricerca nella directory e la visualizzazione di preferiti e recenti. Svolge la funzione di delegato degli altri controller per tutte le operazioni con il model.
DirectorySearch	Collabora con UNIVEDirectorySearch per effettuare ricerche nella directory, elencando i risultati in una TableView.
SearchParams	Consente di specificare dei parametri di ricerca avanzati, occupandosi di fornirli al controller DirectorySearch.

PersonInfo	Mediante il framework *AddressBook* coordina la presentazione delle proprietà del modello Directory_Person in modo coerente con l'applicazione Contatti del sistema operativo. Comunica con applicazioni esterne per l'apertura di URL, la visualizzazione di indirizzi sulla mappa e l'utilizzo di recapiti telefonici o e-mail.
Favorites	Riporta su una TableView i contatti delle persone aggiunte dall'utente nei preferiti, consentendone la rimozione.
Recents	Mantiene un elenco degli ultimi 10 profili consultati dall'utente. Permette in qualsiasi momento di svuotare il contenuto della lista.

Tabella 6.5: Ruoli degli oggetti controller nel sottosistema Directory.

Model	Descrizione
UNIVEDirectorySearch	Ha la responsabilità di scaricare in formato JSON i risultati provenienti dalla directory, instanziandoli nelle classi definite nel modello dati dell'applicazione.
Directory_Person	Rappresenta nell'applicazione un oggetto contenente tutte le informazioni di una persona della directory.

Tabella 6.6: Ruoli degli oggetti model nel sottosistema Directory.

Sottosistema Insegnamenti

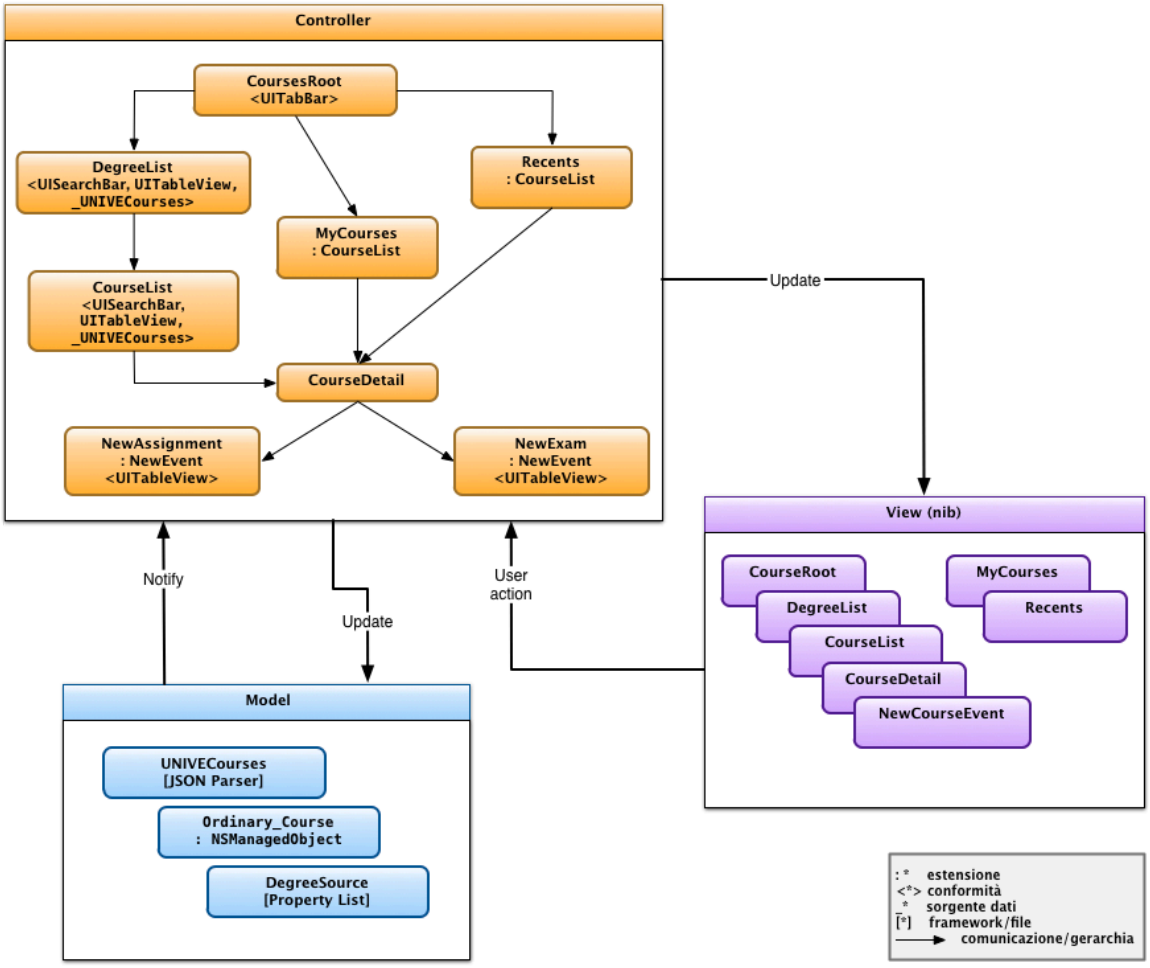

Figura 6.10: Sottosistema Insegnamenti.

Controller	Descrizione
CourseRoot	Coordina, tramite una TabBar, il browsing dei corsi e la visualizzazione di quelli presenti nella lista MyCourses o visti di recente.
DegreeList	Il suo ruolo è quello di fornire una lista dei corsi di laurea tenuti nei vari dipartimenti e suddividerli tra triennali e magistrali. Gestisce, inoltre, una barra per la ricerca dei corsi di laurea, e garantisce il mantenimento della lista dei recenti.

CourseList	Presenta in una TableView gli insegnamenti del corso di laurea selezionato da DegreeList, oppure quelli risultanti dalla ricerca dalla SearchBar da lui gestita.
CourseDetail	Ha lo scopo di presentare in maniera opportuna le proprietà di un oggetto Ordinary_Course. Coordina l'inserimento del corso nella lista personale dell'utente. Collabora con altri sottosistemi o applicazioni esterne per l'apertura di URL, la visualizzazione di indirizzi sulla mappa, l'apertura del profilo del docente o di un corso correlato. Gestisce il controller NewEvent per l'inserimento di un nuovo evento connesso al corso.
NewEvent	La sua responsabilità è quella di preparare i campi necessari per l'inserimento di un nuovo oggetto di tipo Assignment_Event o Exam_Event per il corso selezionato, attraverso le sue rispettive sottoclassi.
MyCourses	Gestisce una lista dei corsi aggiunti dall'utente. Permette la rimozione di un corso dalla lista. Collabora con il sottosistema Eventi per la visualizzazione dello scheduling del corso e l'inserimento di un nuovo oggetto Lecture_Event.
Recents	Elenca i corsi consultati di recente, permettendo di svuotare la lista collezionata.

Tabella 6.7: Ruoli degli oggetti controller nel sottosistema Insegnamenti.

Model	Descrizione
UNIVECourses	Ha la responsabilità di scaricare in formato JSON le informazioni sui corsi richieste dai controller, instanziando gli oggetti definiti nel modello dati dell'applicazione.
Ordinary_Course	Eredita le proprietà della classe Course, e rappresenta nell'applicazione un insegnamento.
DegreeSource	È un file *Property List* contenente una lista dei corsi di laurea e i relativi insegnamenti disponibili e contiene l'indirizzo della sorgente di dati JSON necessaria per sincronizzarli. Viene utilizzato al primo avvio del sottosistema da UNIVECourses, che si occupa di salvare il suo contenuto nei modelli Degree e Course con *Core Data*, velocizzando gli avvii successivi.

Tabella 6.8: Ruoli degli oggetti model nel sottosistema Insegnamenti.

Sottosistema Mappa

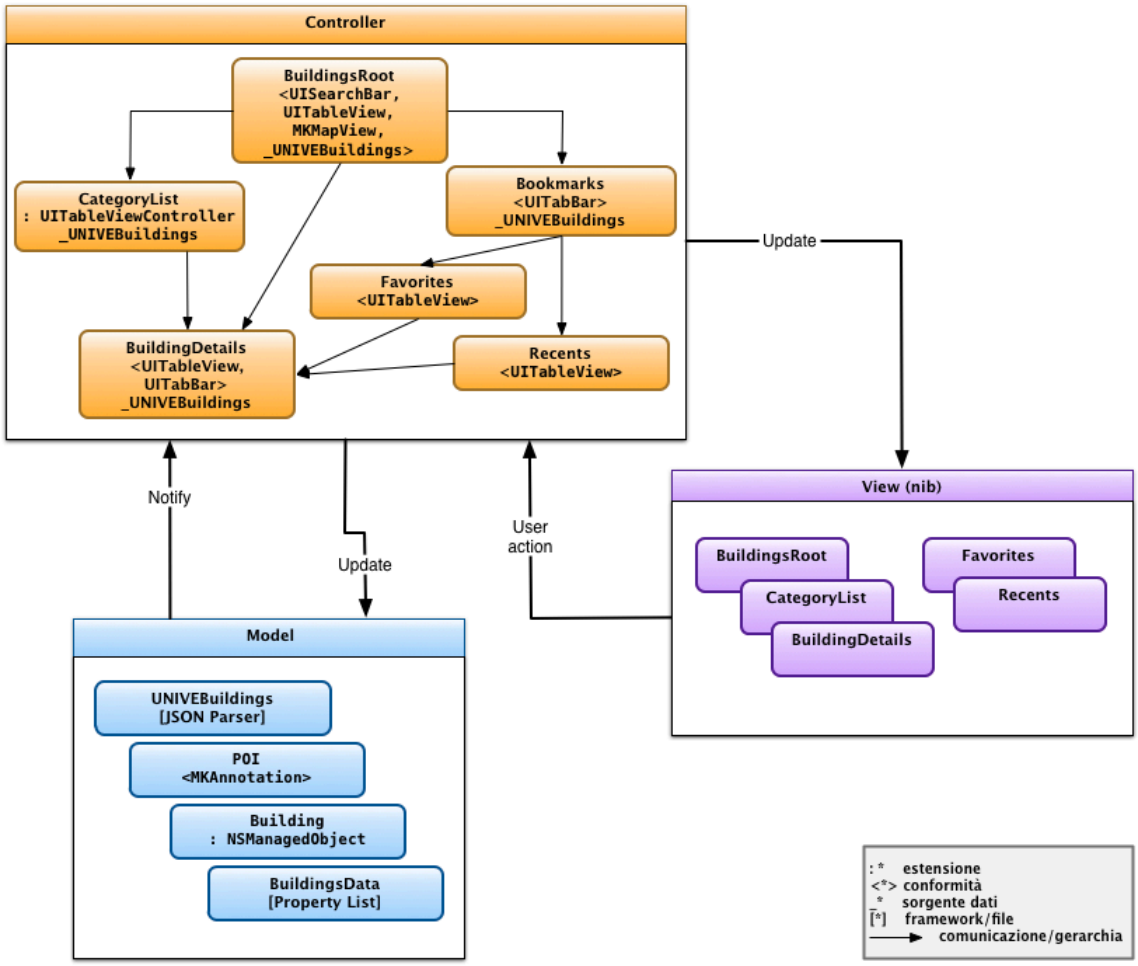

Figura 6.11: Sottosistema Mappa.

Controller	Descrizione
BuildingsRoot	E' responsabile della presentazione della mappa sullo schermo. Risponde agli eventi generati dalla mappa e collabora con gli altri controller per visualizzare le informazioni richieste. Gestisce la ricerca e il filtro dei punti di interesse e provvede alla loro localizzazione sulla mappa. Consente, inoltre, la localizzazione GPS dell'utente.

CategoryList	Consente di sfogliare le tipologie dei *POI* selezionati e di visualizzare in una *TableView* le rispettive località di interesse.
BuildingDetails	Gestisce la presentazione di una view dettagliata delle informazioni di un punto di interesse. Si affida a BuildingsRoot per la visualizzazione della località sulla mappa e collabora con altri sottosistemi o applicazioni esterne per l'utilizzo dei recapiti telefonici o e-mail, l'apertura di URL oppure l'ottenimento delle indicazioni stradali da o verso la località scelta. Permette inoltre l'inserimento nei preferiti.
Bookmarks	Coordina la visualizzazione dei preferiti e dei recenti, attraverso una TabBar in una view presentata in maniera modale.
Favorites	Riporta su una TableView le località aggiunte dall'utente nei preferiti, consentendone la rimozione.
Recents	Elenca le località consultate di recente, permettendo di svuotare la lista collezionata.

Tabella 6.9: Ruoli degli oggetti controller nel sottosistema Mappa.

Model	Descrizione
UNIVEBuildings	Ha la responsabilità di scaricare in formato JSON le informazioni sulla località richieste dai controller, instanziando gli oggetti definiti nel modello dati dell'applicazione.
POI	Corrisponde ad una annotazione sulla mappa che identifica un punto di interesse. Può essere rappresentato visualmente in maniera diversa a seconda dei tipi definiti nel modello Building_Type.
Building	Rappresenta una località listata in BuildingsData. Si appoggia ad altre entità per modellare una serie di informazioni dettagliate, che vanno dalle coordinate GPS agli orari di apertura di un ufficio.
BuildingsData	È un file *Property List* contenente le minime informazioni indispensabili per visualizzare sulla mappa un oggetto POI di ogni località disponibile. Viene utilizzato al primo avvio del sottosistema da UNIVEBuildings, che si occupa di effettuare il *caching* con *Core Data*, velocizzando gli avvii successivi.

Tabella 6.10: Ruoli degli oggetti model nel sottosistema Mappa.

Sottosistema Media

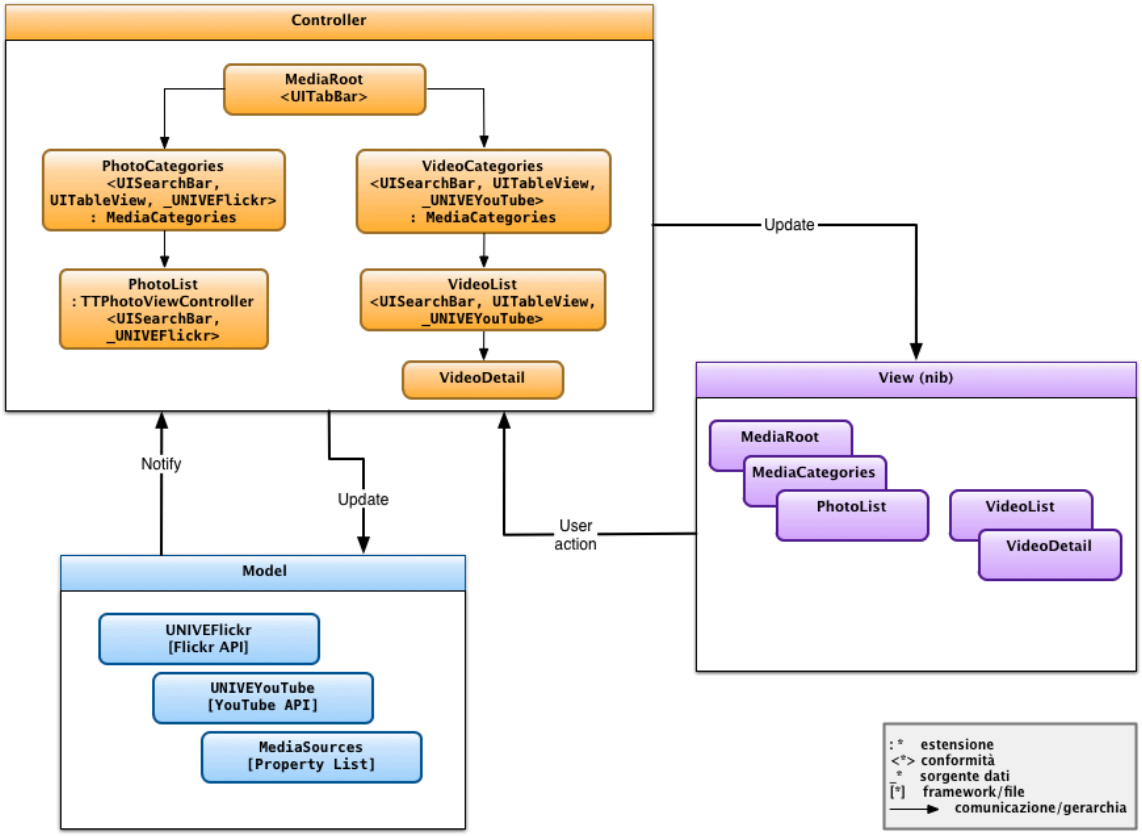

Figura 6.12: Sottosistema Media.

Controller	Descrizione
MediaRoot	Consente di scegliere da una TabBar il tipo di media da consultare. La selezione di una delle opzioni della TabBar comporta il caricamento del rispettivo controller.
MediaCategories	Rappresenta un controller estensibile che si occupa di presentare una lista delle categorie disponibili (album o canali) per ogni tipo di media, permettendo di eseguire una ricerca dei contenuti in tutte le categorie.
PhotoList	Estende la classe TTPhotoController, che permette di rappresentare un'anteprima con *thumbnail* delle immagini

	presenti nella categoria selezionata e di avviare uno slide show delle immagini.
VideoList	Offre anch'esso una visualizzazione a griglia dei contenuti di una categoria, a differenza che la selezione di un'anteprima comporta il caricamento del controller VideoDetail.
VideoDetail	Si occupa di presentare le informazioni riguardanti un video e di effettuarne il playback.

Tabella 6.11: Ruoli degli oggetti controller nel sottosistema Media.

Model	Descrizione
UNIVEFlickr	Svolge il ruolo di data source per i controller della tipologia photo, effettuando delle chiamate alle *API* di Flickr
UNIVEYouTube	Stesso ruolo di UNIVEFlickr, utilizzando le *API* di YouTube.
MediaSources	File *Property List* contenente i collegamenti agli album Flickr e i canali YouTube da cui reperire i media.

Tabella 6.12: Ruoli degli oggetti model nel sottosistema Media.

Sottosistema Social

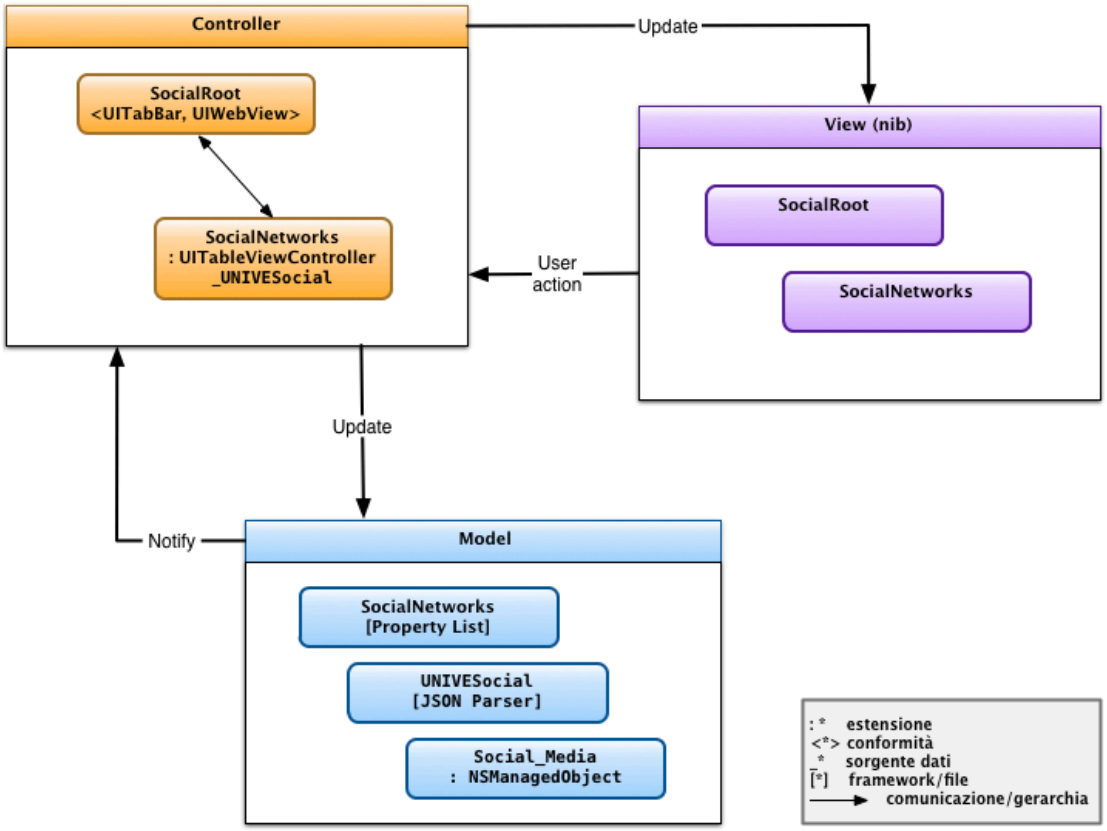

Figura 6.13: Sottosistema Social.

Controller	Descrizione
SocialRoot	Tramite una TabBar coordina la selezione del social media desiderato, e controllando una WebView ne mostra i contenuti.
SocialNeworks	Controller che ha lo scopo di presentare modalmente una view per la selezione di un social network tra quelli disponibili nello stesso social media.

Tabella 6.13: Ruoli degli oggetti controller nel sottosistema Social.

Model	Descrizione
UNIVESocial	Ha il compito di effettuare il *caching* in *Core Data* dei dati contenuti nel file SocialNetworks. Svolge periodicamente la ricerca di nuovi social network, effettuando il parsing di dati JSON scaricati dalla rete.
Social_Media	Rappresenta nell'applicazione un insieme di social network suddivisi per categorie.
SocialNetworks	File *Property List* contenente una lista di tutti i social network disponibili al primo avvio del sottosistema, suddivisi per social media nelle opportune categorie.

Tabella 6.14: Ruoli degli oggetti model nel sottosistema Social.

Sottosistema Radio Ca' Foscari

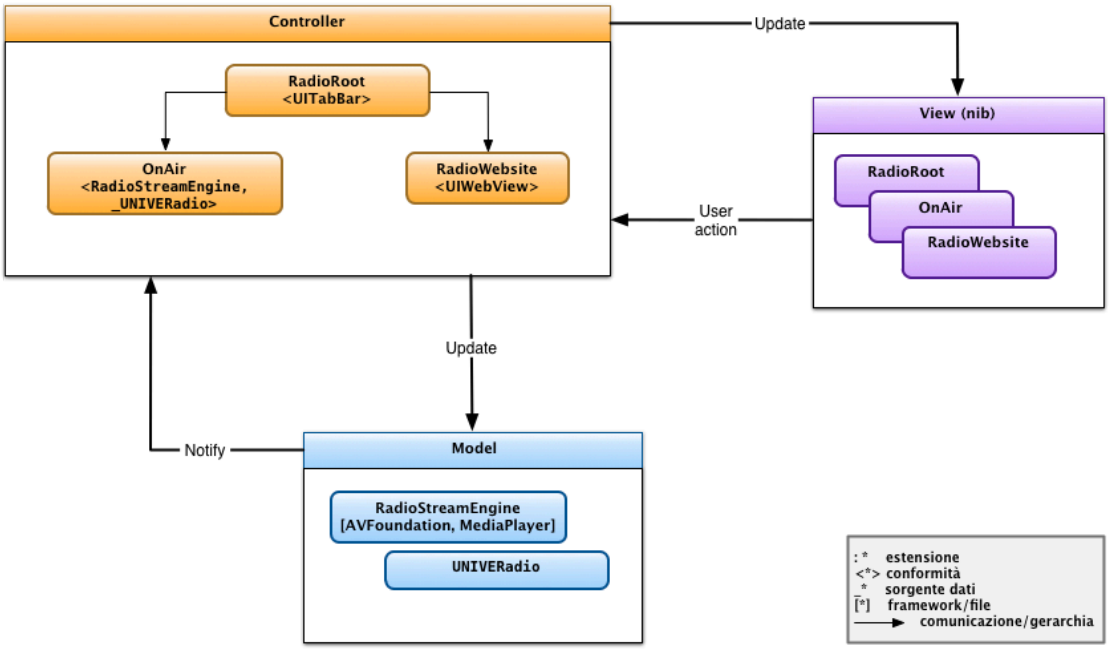

Figura 6.14: Sottosistema Radio Ca' Foscari.

Controller	Descrizione
RadioRoot	Gestisce una TabBar che controlla il caricamento dei controller per il live streaming e la visualizzazione del sito web di RCF.
OnAir	Controlla lo stream audio dalla web radio RCF collaborando con RadioStreamEngine .
RadioWebsite	Ha la responsabilità di presentare la pagina web di RCF.

Tabella 6.15: Ruoli degli oggetti controller nel sottosistema RCF.

Model	Descrizione
RadioStreamEngine	È il motore che si occupa di scaricare lo stream di dati dal server di RCF e di fornire al controller OnAir l'accesso a tale stream mediante un media player.
UNIVERadio	Fornisce allo stream engine la configurazione necessaria per collegarsi al server di RCF.

Tabella 6.16: Ruoli degli oggetti model nel sottosistema RCF.

Sottosistema Risorse

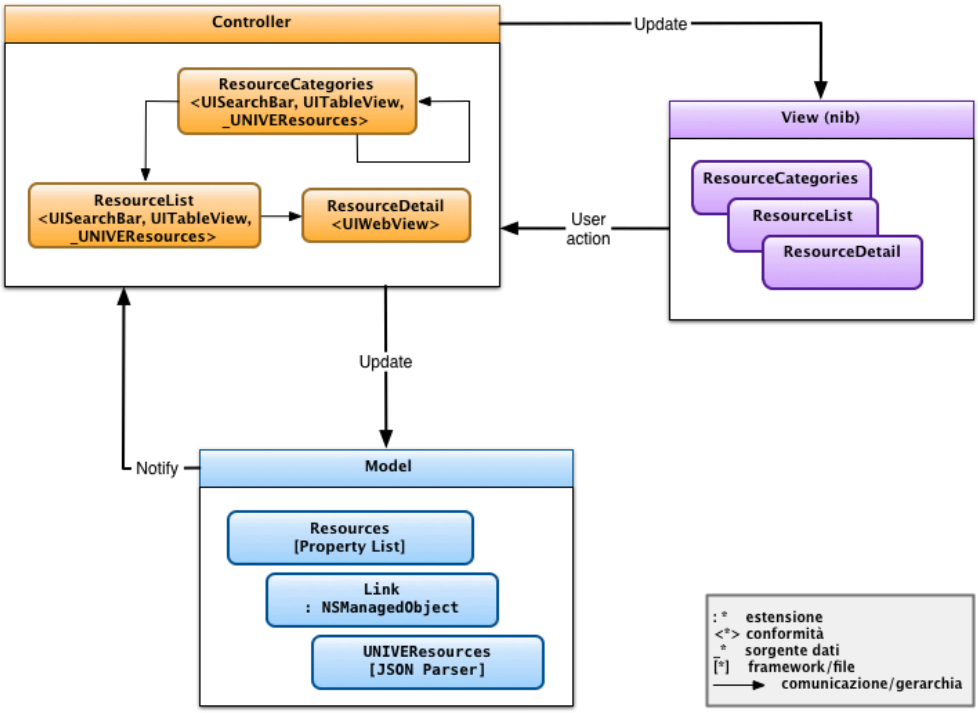

Figura 6.15: Sottosistema Risorse.

Controller	Descrizione
ResourceCategories	Elenca le categorie delle risorse in una TableView, gestendo la navigazione nelle sottocategorie e il caricamento dei contenuti della categoria selezionata o dei risultati di una ricerca.
ResourceList	Sempre in una TableView, si occupa di elencare i contenuti di una categoria e coordina la ricerca tra i contenuti listati.
ResourceDetail	Ha la responsabilità di accedere all'URL di una risorsa e di visualizzarne il contenuto su una WebView.

Tabella 6.17: Ruoli degli oggetti controller nel sottosistema Risorse.

Model	Descrizione
UNIVEResources	Si occupa di effettuare il *caching* dei dati con *Core Data* e di scaricare nuove risorse in formato JSON.
Link	Rappresenta nell'applicazione una URL. E' utilizzato nel modello Resource_Category per costituire un insieme di risorse correlate.
Resources	File Property List, utilizzato al primo avvio del sottosistema per popolare il modello e stabilire la sorgente di dati per UNIVEResources.

Tabella 6.18: Ruoli degli oggetti model nel sottosistema Risorse.

Sottosistema Avvisi

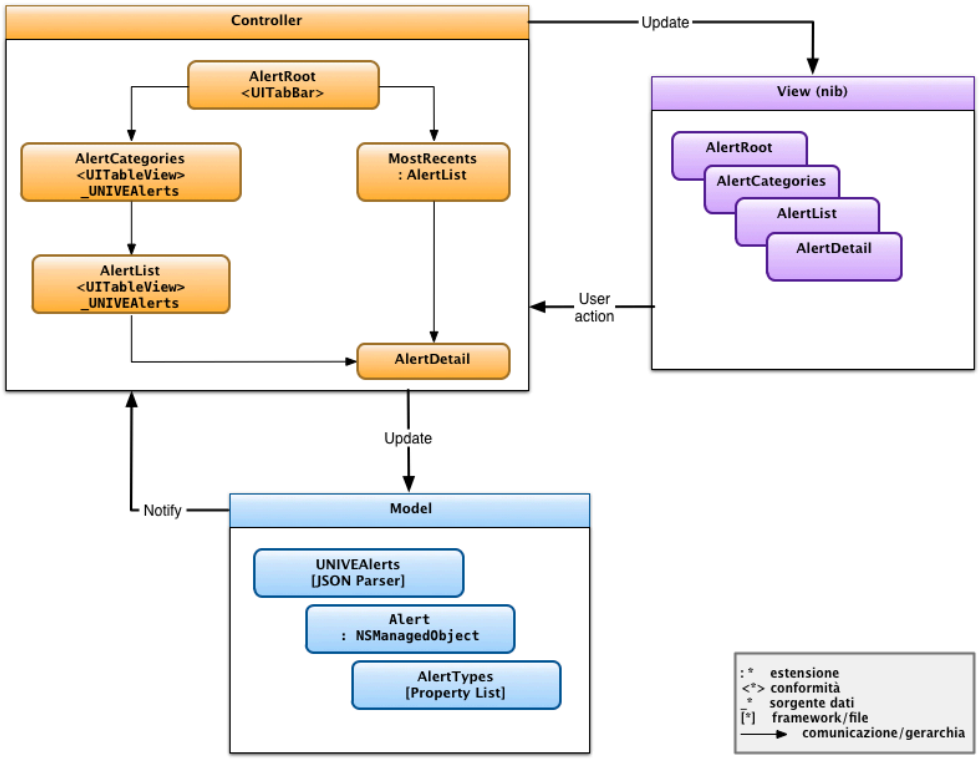

Figura 6.16: Sottosistema Avvisi.

Controller	Descrizione
AlertRoot	Organizza il caricamento dei controller che gestiscono il browsing degli avvisi e la visualizzazione delle notifiche più recenti.
AlertCategories	Si occupa di visualizzare una lista di categorie di avvisi, collaborando con UNIVEAlerts per ricevere eventuali aggiornamenti.
AlertList	Ha la responsabilità di listare gli articoli della categoria selezionata in ordine cronologico.
AlertDetail	Coordina la visualizzazione dei dettagli di un avviso e l'apertura di eventuali URL presenti nella descrizione.
MostRecents	Elenca tutti gli avvisi ricevuti più di recente.

Tabella 6.19: Ruoli degli oggetti controller nel sottosistema Avvisi.

Model	Descrizione
UNIVEAlerts	Si occupa di scaricare i dati dalle sorgenti definite in AlertTypes, effettuando il *caching* con *Core Data* nelle rispettive categorie.
Alert	Rappresenta il modello che definisce le proprietà di un avviso nell'applicazione.
AlertTypes	File Property List utilizzato da UNIVEAlerts per definire le categorie di avvisi disponibili e le rispettive sorgenti.

Tabella 6.20: Ruoli degli oggetti model nel sottosistema Avvisi.

CONCLUSIONI

In questo capitolo sono stati introdotti i principali pattern utilizzati nella progettazione. Inoltre, una serie di diagrammi ha permesso di formare una visione di insieme dell'applicazione, con un certo livello di dettaglio nell'applicazione del pattern MVC.

Nei diagrammi che illustrano i sottosistemi, è spesso presente un modello con prefisso UNIVE. Questi, nella maggior parte dei casi, hanno la responsabilità di scaricare dei dati in formato JSON e parsificarli per conformarli al modello dati dell'applicazione. Tale processo si affida ad un'infrastruttura server per la fornitura di *web services* verso l'applicazione. La progettazione di tale sistema non rientra nell'ambito di questa tesi, tuttavia il suo funzionamento verrà brevemente teorizzato nei capitoli successivi.

Il capitolo seguente, invece, descriverà la struttura integrale del modello dati di iCa'Foscari.

Capitolo 7: Diagrammi dei casi d'uso

Questo capitolo completa la definizione dei requisiti funzionali del sistema software iCa'Foscari. Nelle pagine seguenti sono riportati i diagrammi *UML* dei casi d'uso di ciascuna macrofunzionalità prevista nell'applicazione.

LEGENDA DEI DIAGRAMMI USE CASE

Gli elementi principali di un diagramma dei casi d'uso, o anche detto diagramma *use case*[7] sono: l'attore, il caso d'utilizzo e le associazioni. Gli attori sono rappresentati graficamente da un'icona che rappresenta un uomo stilizzato. Un caso d'uso, invece, è rappresentato graficamente come un'ellisse contenente il nome del caso d'uso.

L'associazione fondamentale è quella che congiunge gli attori con i casi d'utilizzo a cui essi partecipano. Un attore può essere associato a un qualsiasi numero di casi d'uso, e viceversa. Pur non richiedendo ulteriori informazioni, l'associazione fra gli *use case* e gli *actor* implica uno scambio messaggi fra attori e *use case* associati.

La relazione di estensione fra *use case*, disegnata da una linea tratteggiata con denotazione dello stereotipo *«extends»*, indica che la funzione rappresentata dal caso d'uso case "estendente" (alla base della freccia) può essere impiegata nel contesto della funzione "estesa" (il caso d'uso alla punta), ovvero ne rappresenta una sorta di arricchimento.

La relazione di inclusione fra *use case*, disegnata da una linea tratteggiata con denotazione dello stereotipo *«includes»*, indica che la funzione rappresentata da uno dei due casi d'utilizzo (quello alla base della freccia) include completamente la funzione rappresentata dall'altro (quello alla punta). Si può esprimere questa relazione anche con

[7]http://en.wikipedia.org/wiki/Use_case_diagram

lo stereotipo *«uses»*, la cui interpretazione, però, deve essere libera da considerazioni implementative.

La notazione utilizzata è riassunta brevemente nella seguente legenda:

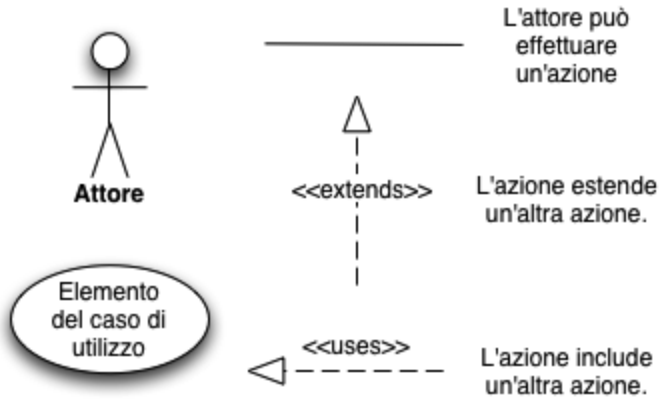

Figura 7.1: Legenda diagramma casi d'uso.

DIAGRAMMI UML

Di seguito sono riportati i diagrammi dei casi d'uso svolti dall'attore "utente" definito nel capitolo dei requisiti. Non sono invece descritti ad uno ad uno i casi d'uso, in quanto le etichette rispettivamente assegnate nei diagrammi sono già sufficientemente esplicative. Alternativamente, il capitolo sulla progettazione della *GUI* fornisce ulteriori informazioni utili per interpretare i casi di utilizzo qui definiti.

Casi d'uso News

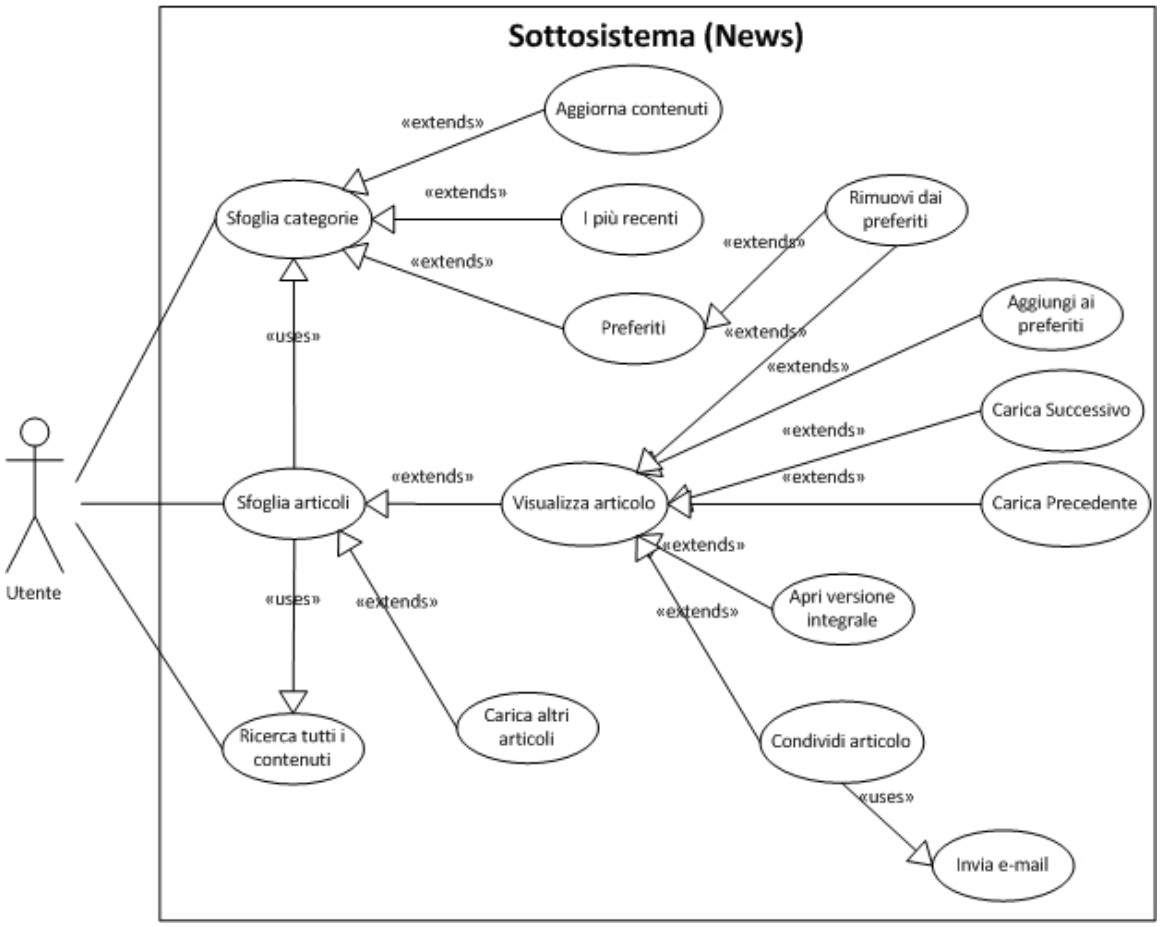

Figura 7.2: Diagramma casi d'uso News.

Casi d'uso Eventi

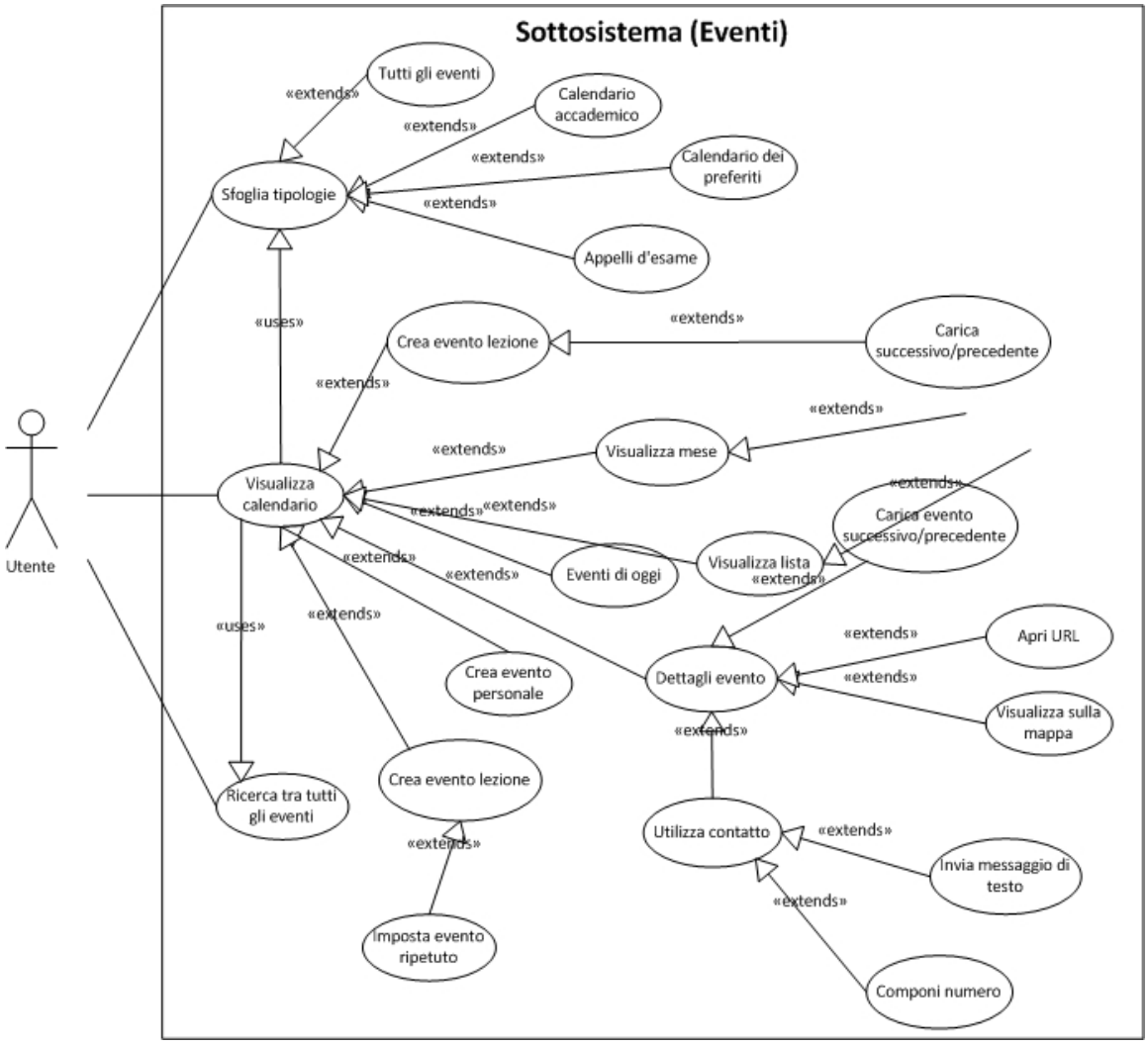

Figura 7.3: Diagramma casi d'uso Eventi.

Casi d'uso Directory

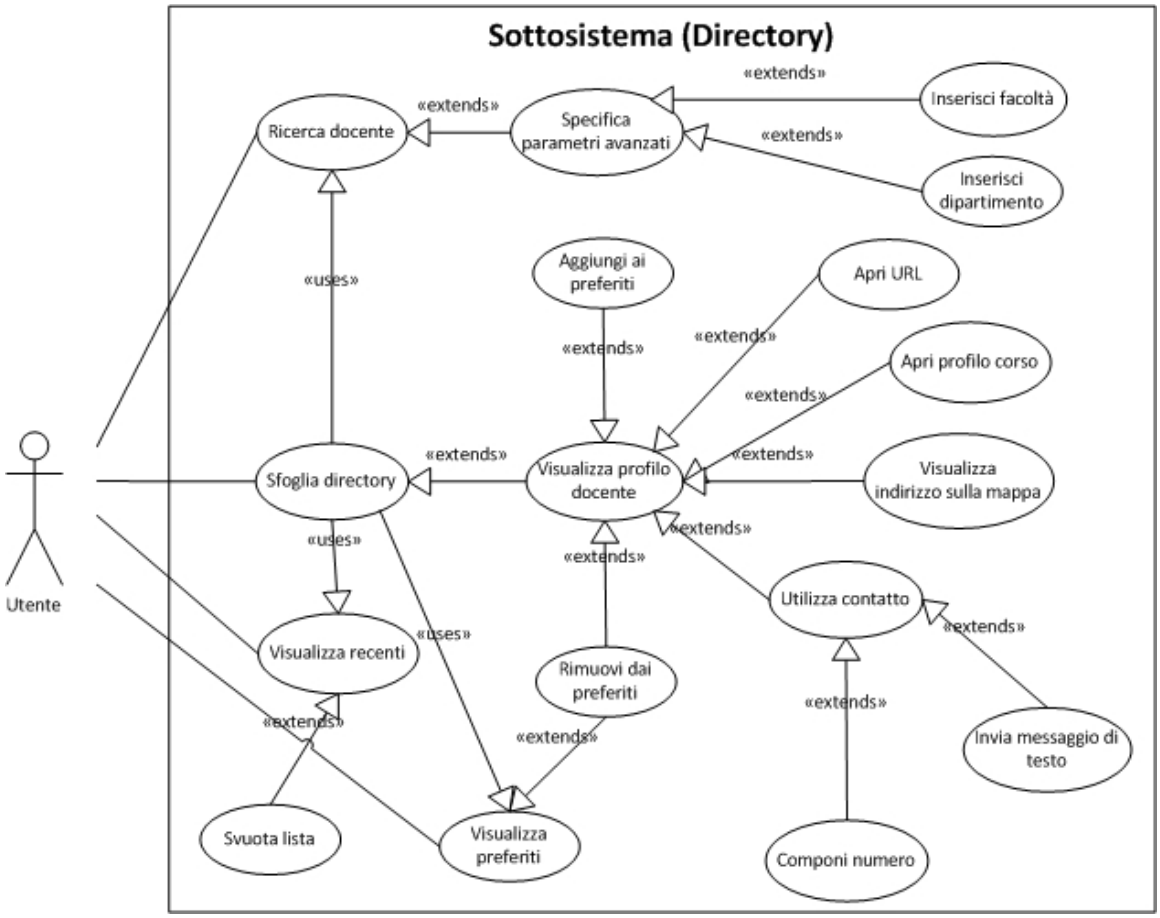

Figura 7.4: Diagramma casi d'uso Directory.

Casi d'uso Insegnamenti

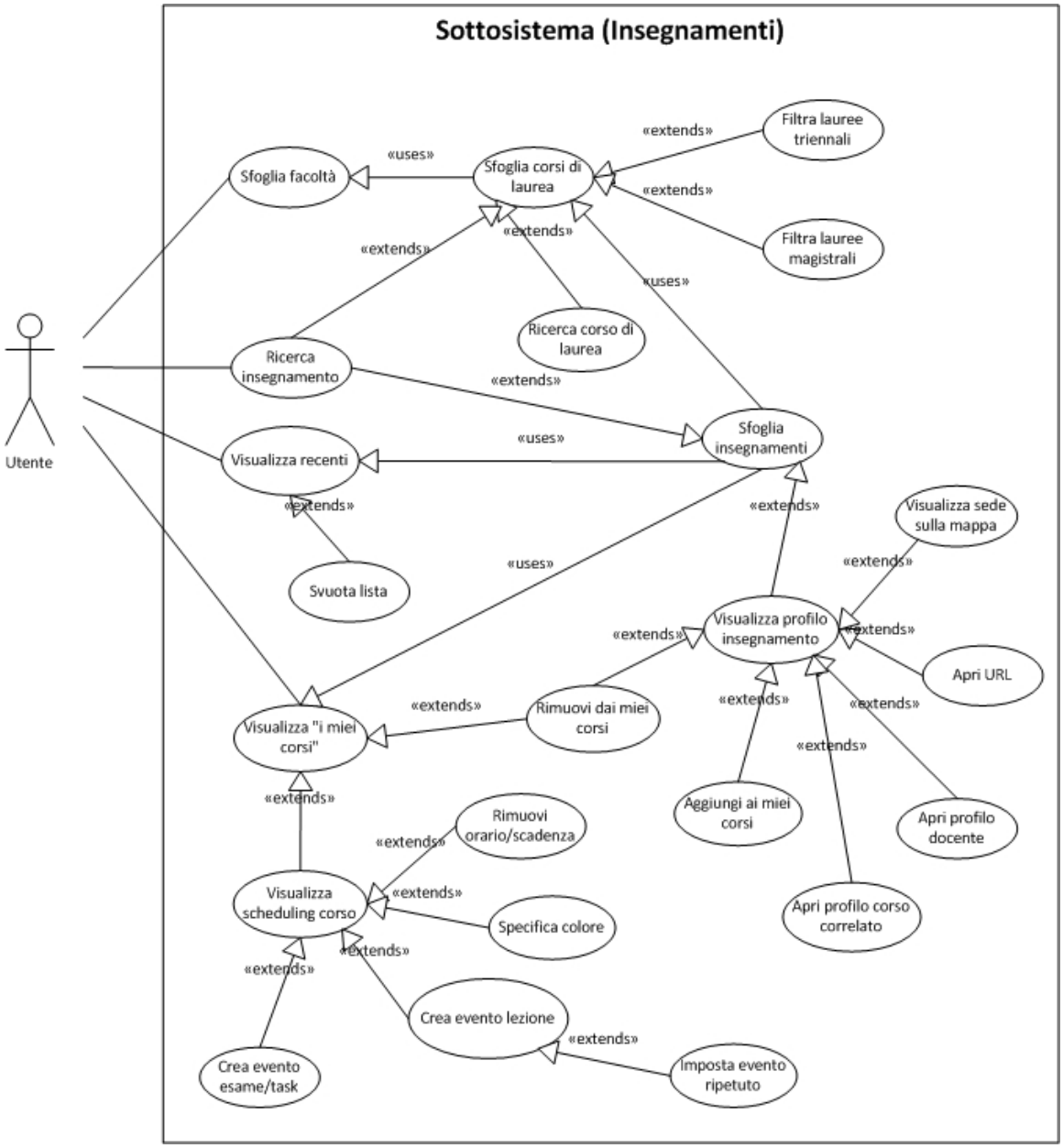

Figura 7.5: Diagramma casi d'uso Insegnamenti.

Casi d'uso Mappa

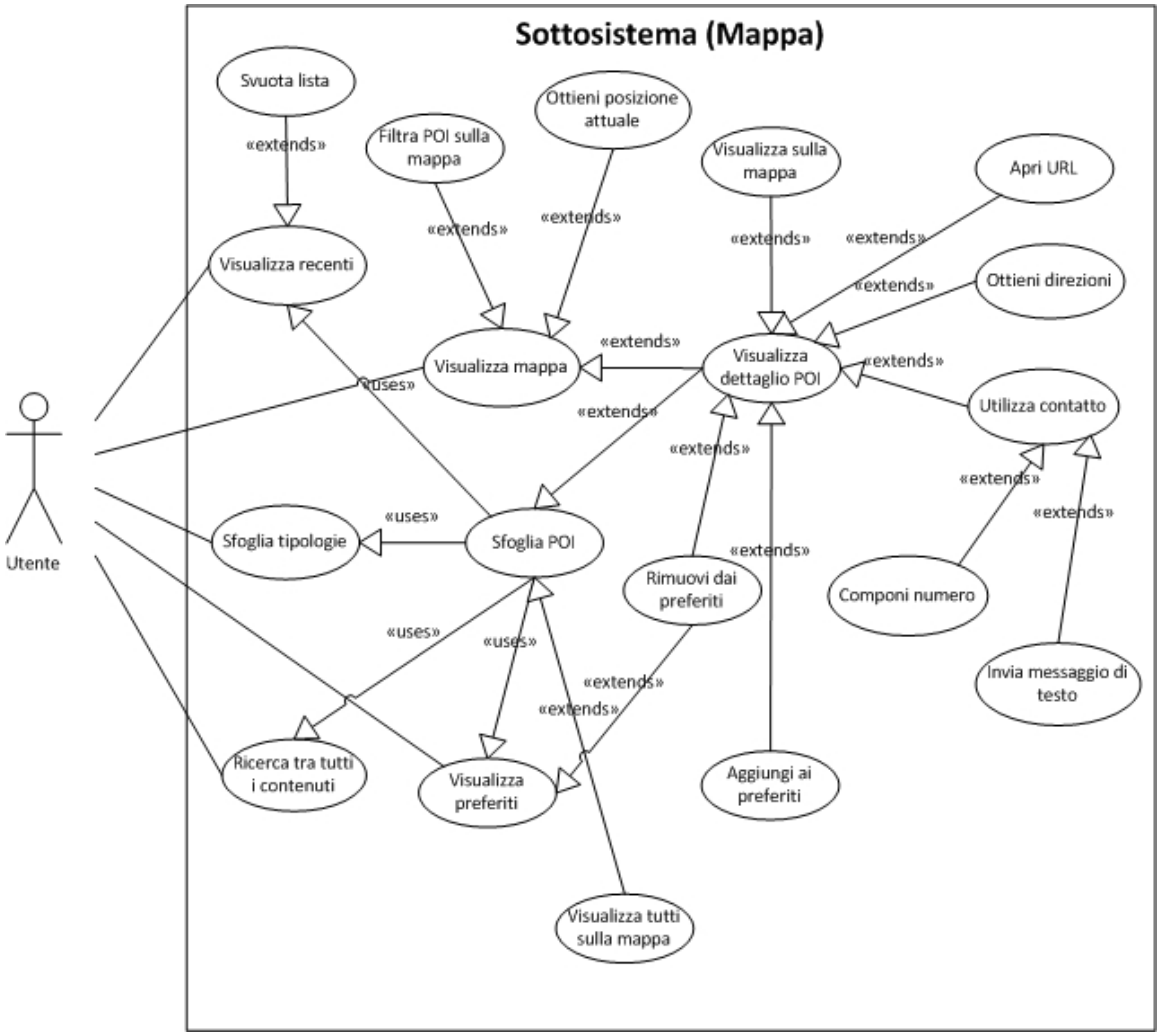

Figura 7.6: Diagramma casi d'uso Mappa.

Casi d'uso Media

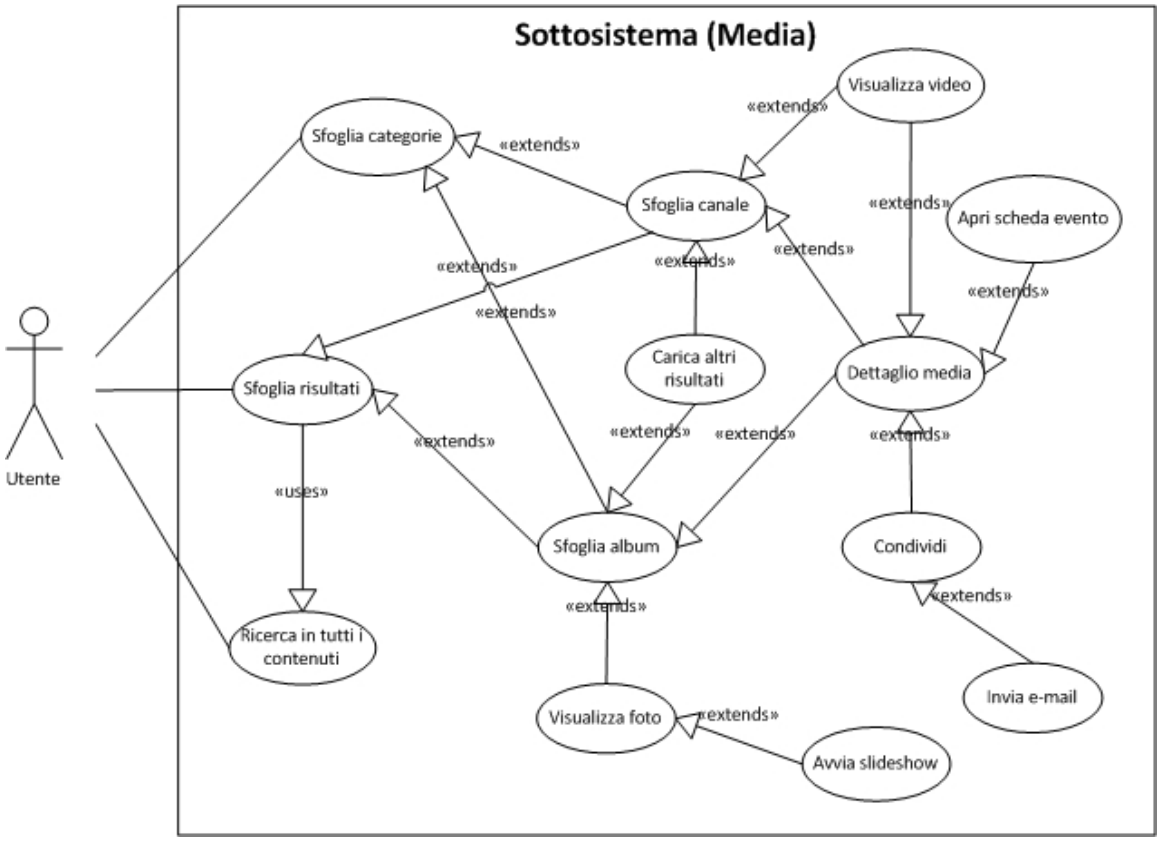

Figura 7.7: Diagramma casi d'uso Media.

Casi d'uso Avvisi

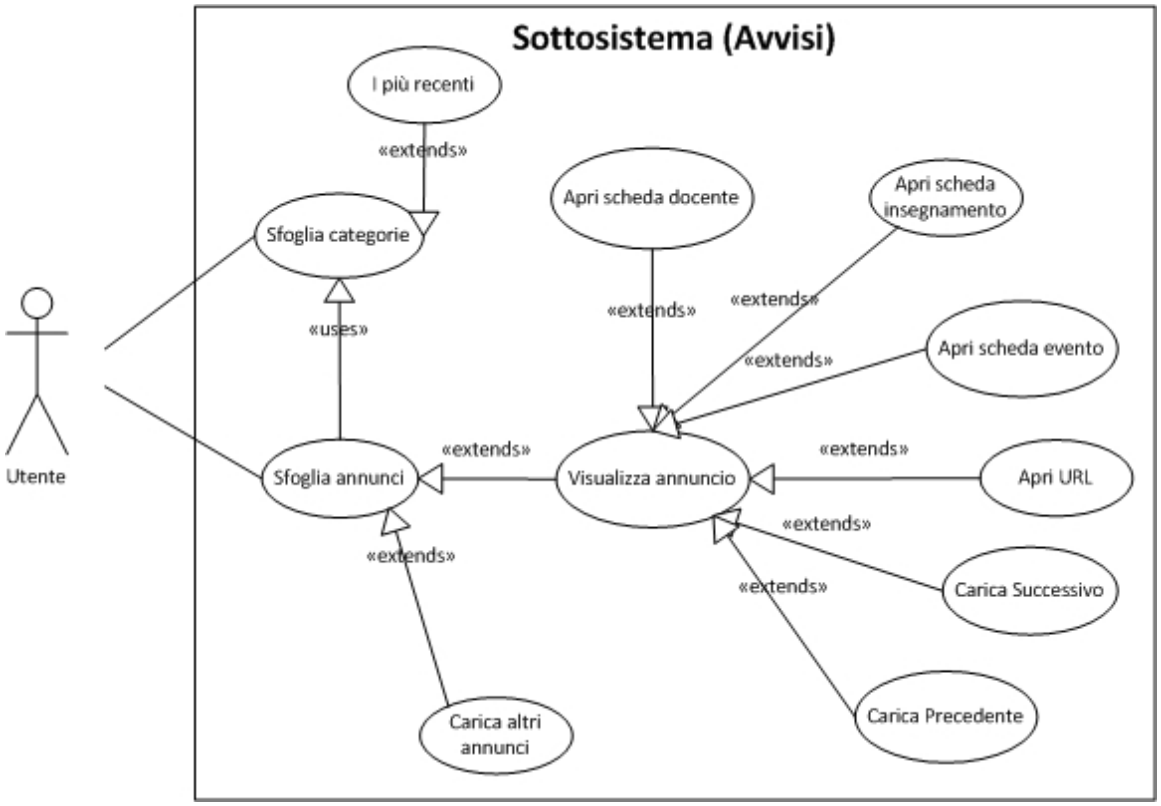

Figura 7.8: Diagramma casi d'uso Avvisi.

CONCLUSIONI

I diagrammi riportati nelle pagine precedenti descrivono, nel linguaggio *UML*, i casi di utilizzo delle funzionalità che corrispondono ai sottosistemi più complessi di iCa'Foscari. Sono stati invece tralasciati i diagrammi *use case* più banali, in particolare quelli delle funzionalità Emergenza, Social, iTunesU, Radio Ca' Foscari e Risorse. Per queste ultime, infatti, la descrizione trattata nel capitolo dei requisiti è più che sufficiente ad individuare i relativi casi di utilizzo.

Capitolo 8: Progettazione della Struttura dati

Questo capitolo descrive in maniera dettagliata la struttura dati di iCa'Foscari. Il diagramma proposto è il risultato dell'applicazione del pattern Object Modeling introdotto al capitolo precedente. Le entità presenti nel diagramma vengono poi raccolte per funzionalità e discusse ad una ad una mediante un'apposita scheda. Infine, vengono fatte alcune osservazioni su *caching*, eliminazione e migrazione dei dati.

DIAGRAMMA ENTITÀ-RELAZIONE

Di seguito è riportato il diagramma entità-relazione del modello dati, realizzato con l'*xcdatamodel designer* di *XCode*. Poiché segue il pattern Object Modeling, esso è parificato al modello *UML* concettuale delle classi, per quanto già visto al capitolo precedente.

Il diagramma, inoltre, rispetta le convenzioni stabilite da Apple per la progettazione *Core Data*. In particolare, tra le altre,

- I nomi delle entità devono iniziare con una lettera maiuscola, mentre le proprietà per lettera minuscola;
- Per ogni relazione deve essere fissata la sua inversa.

Il disegno è stato suddiviso in quattro sezioni, raggruppando le entità nei principali sottosistemi.

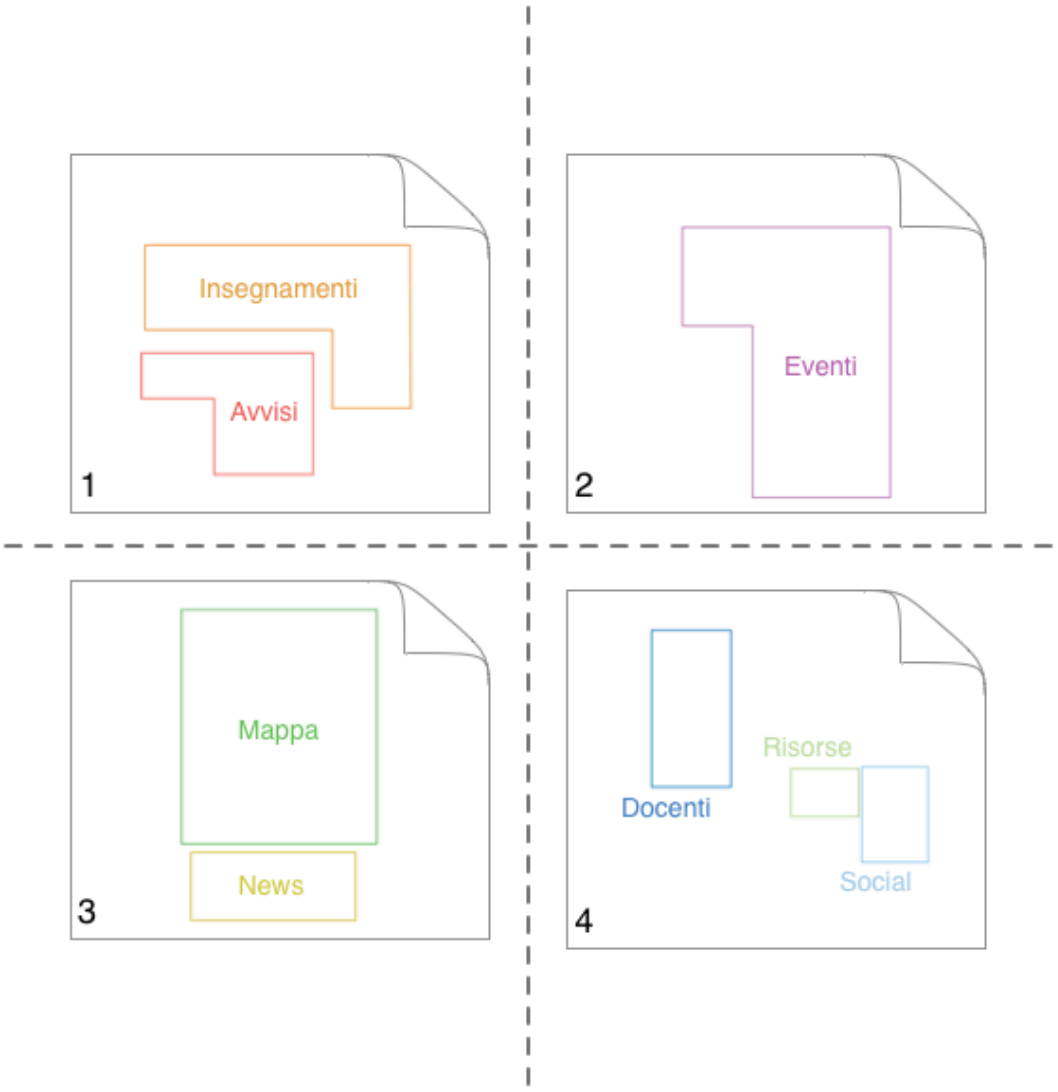

Figura 8.1: Panoramica layout diagramma.

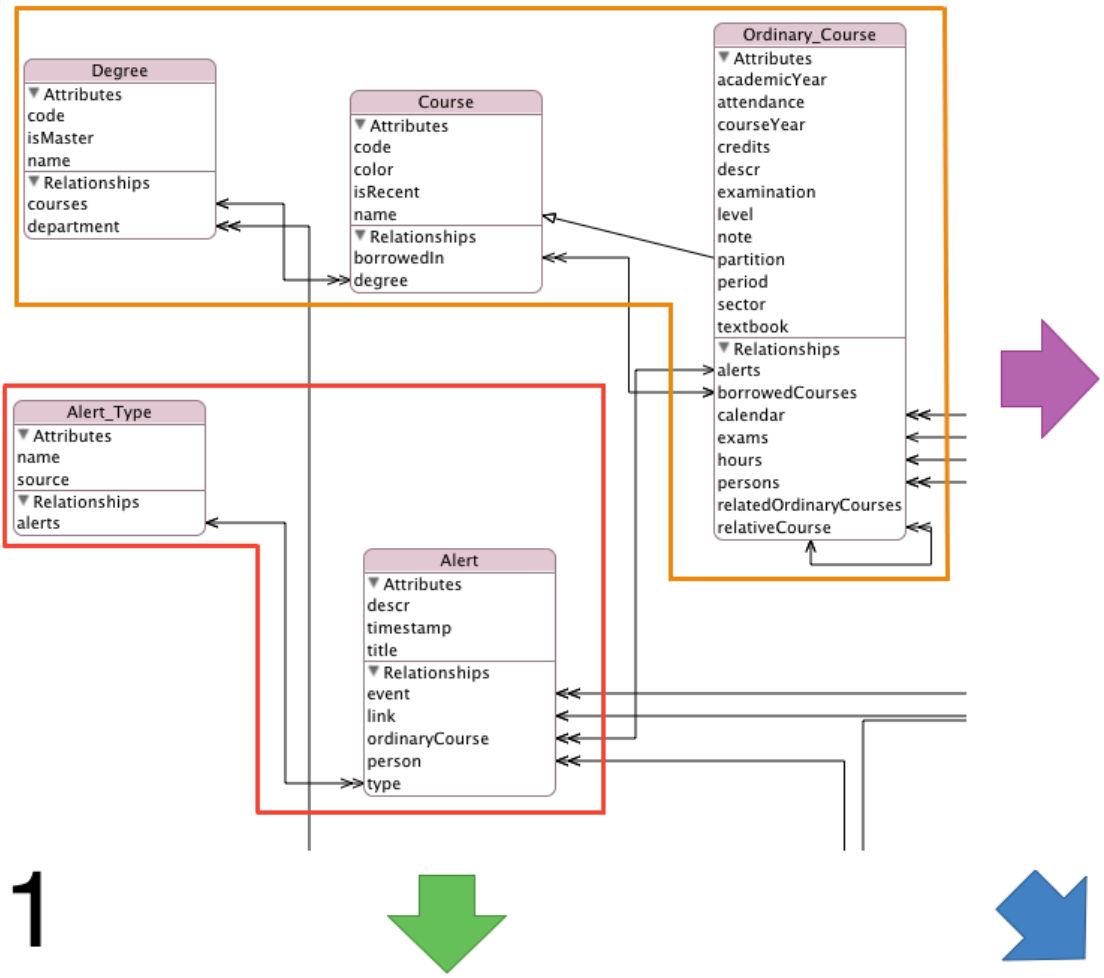

Figura 8.2: Diagramma struttura dati, sezione 1.

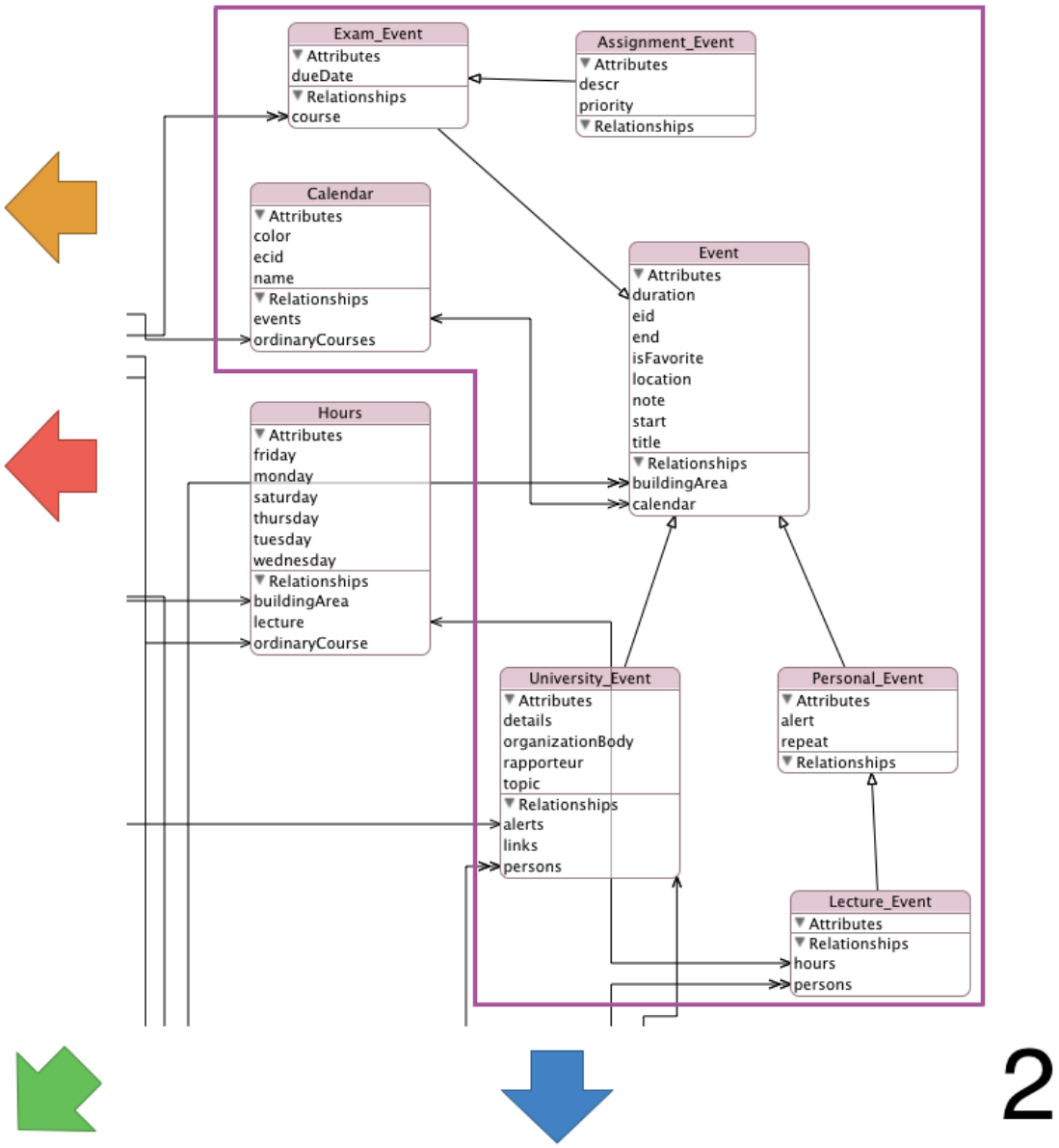

Figura 8.3: Diagramma struttura dati, sezione 2.

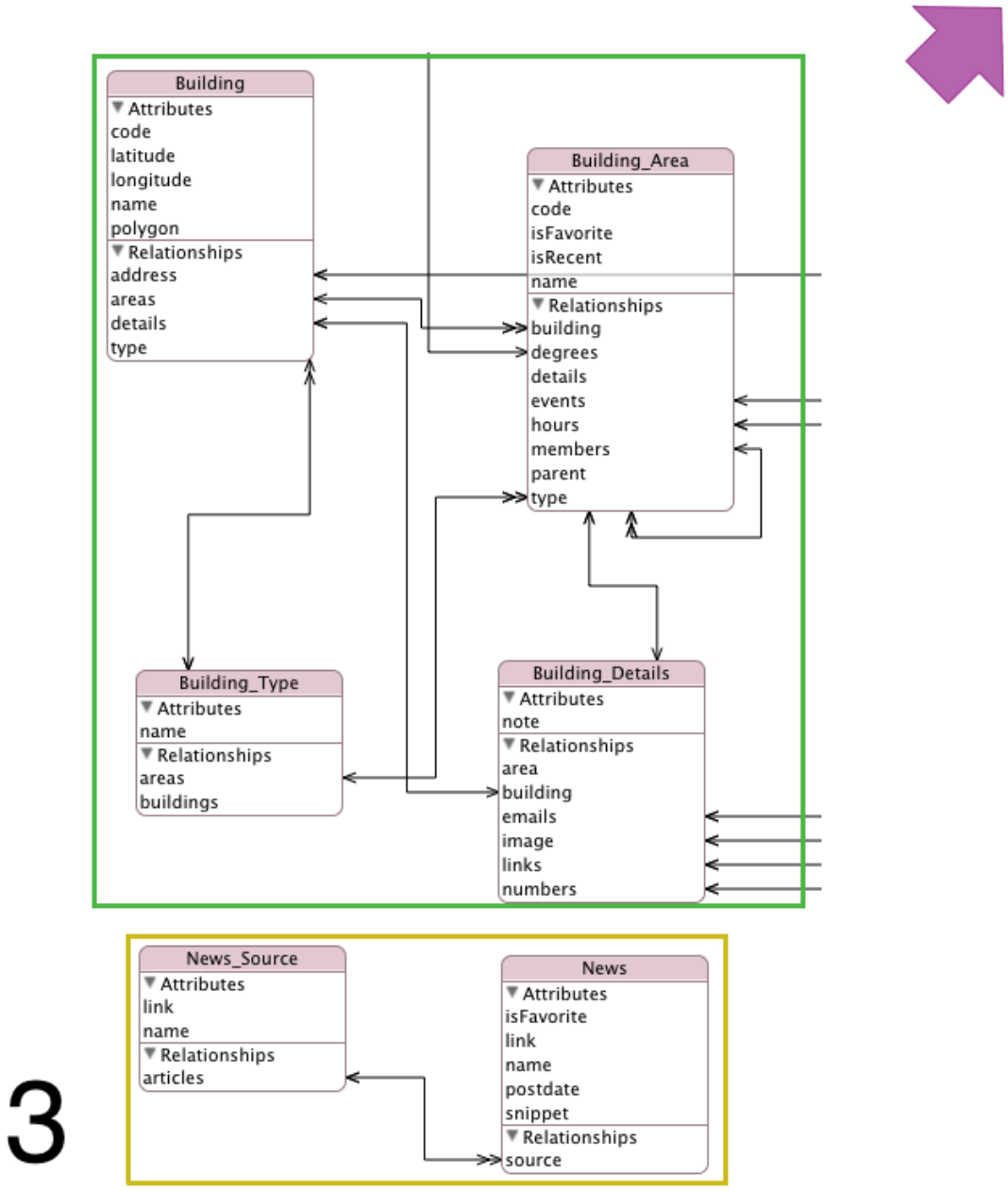

Figura 8.4: Diagramma struttura dati, sezione 3.

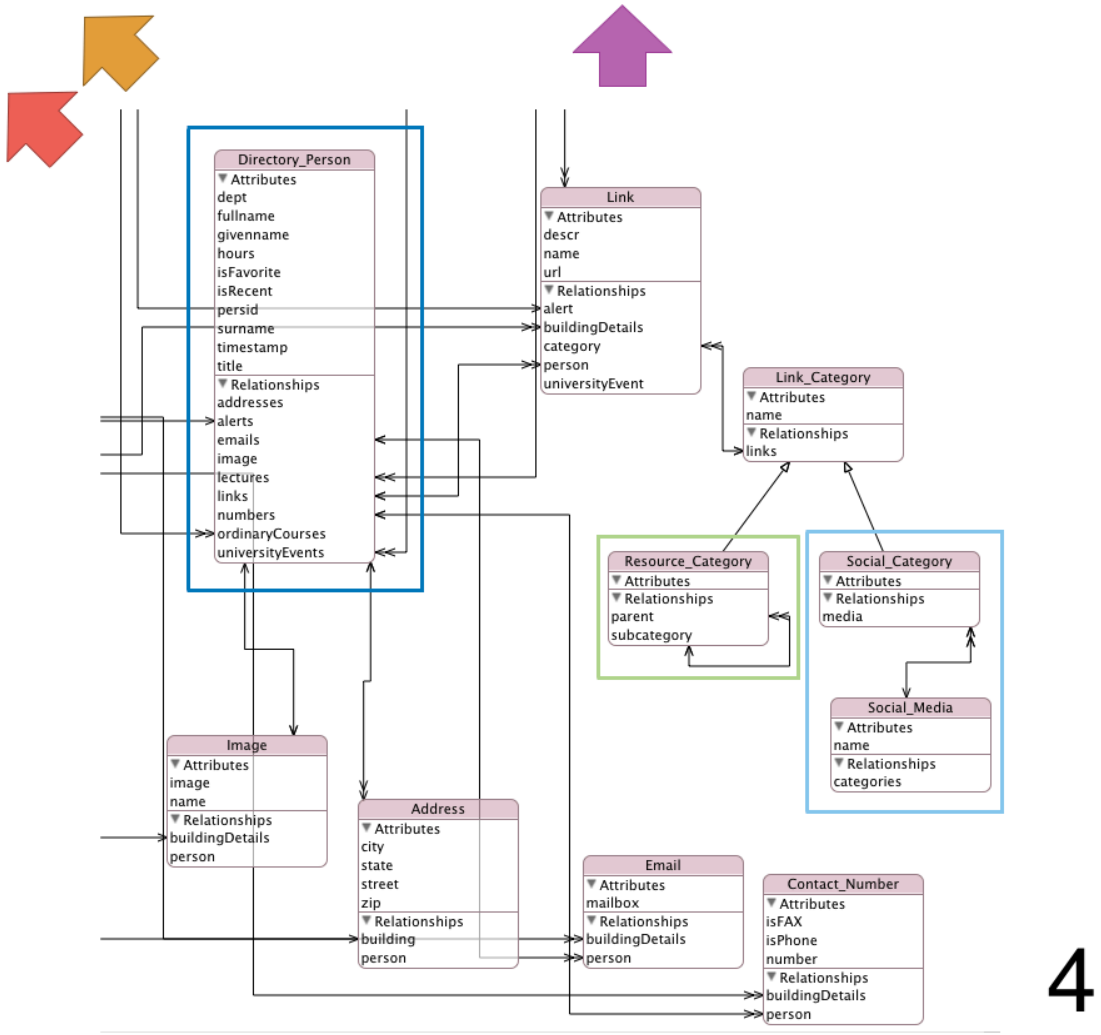

Figura 8.5: Diagramma struttura dati, sezione 4.

DESCRIZIONE DELLE ENTITÀ

Nelle prossime pagine sono riportate le schede di descrizione di tutte le entità presenti nella struttura dati, raggruppate per ogni sottosistema dell'applicazione.

Ogni scheda è strutturata nel modo seguente:

<table>
<tr><td colspan="5">Nome entità:</td></tr>
<tr><td colspan="5">Attributi</td></tr>
<tr><td colspan="2">Nome</td><td colspan="2">Tipo</td><td>Opzioni</td></tr>
<tr><td colspan="5">Relazioni</td></tr>
<tr><td>Nome</td><td colspan="2">Destinazione (inversa)</td><td>Opzioni</td><td>Regola delete</td></tr>
<tr><td colspan="5"></td></tr>
<tr><td colspan="2">Ruolo e responsabilità:</td><td colspan="3"></td></tr>
<tr><td colspan="2">Politiche caching:</td><td colspan="3"></td></tr>
<tr><td colspan="2">Origine dati:</td><td colspan="3"></td></tr>
</table>

Tabella 8.1: Scheda entità generica.

In particolare,

- Opzioni indica se per il rispettivo attributo o relazione sono stati settati dei flag come Indexed, Optional, Transient oppure To-Many;
- Regola delete specifica l'azione da intraprendere nei confronti di un'eliminazione. Le regole ammesse sono: No Action, Nullify, Cascade e Deny;
- Ruolo e responsabilità definisce le competenze dell'entità nell'applicazione;
- Politiche caching dichiara se è prevista una forma di caching dei dati. Persistente indica che i dati permangono fintanto che non vengono espressamente eliminati, altrimenti può essere prevista una politica di caching condizionale o a tempo definito;

- Origine dati stabilisce la fonte dei dati coi quali deve essere popolata l'entità. Può essere di due tipi: File Property List, se i dati sono inclusi nel *bundle* dell'applicazione) o sorgente JSON, se i dati provengono da un server esterno.

All'indirizzo http://developer.apple.com/documentation/Cocoa/Conceptual/CoreData/ è possibile ottenere maggiori informazioni sulle Opzioni e le Regole appena viste.

Struttura dati News

La struttura dati del sottosistema News è composta dalle seguenti entità (in ordine alfabetico):

- News;
- News_Source.

Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

Entità News

Nome entità: News			
Attributi			
Nome	Tipo	Opzioni	
isFavorite	Boolean (default value: NO)	Mandatory	
link	String	Mandatory	
name	String	Optional	
postdate	Date	Mandatory	
snippet	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
source	News_Source (articles)	Mandatory	Nullify
Ruolo e responsabilità:	Modella le sole informazioni necessarie per elencare un articolo in una tabella in ordine cronologico, mostrando nome e breve descrizione. Il contenuto vero e proprio potrà invece essere reperito con il campo link.		

Politiche caching:	Caching oggetto a tempo determinato sulla base del campo postdate.
Origine dati:	Feed RSS.

Tabella 8.2: Proprietà entità News.

NOTE: Il campo isFavorite funziona da discriminante per eseguire un *fetch* con *predicate* degli articoli aggiunti dall'utente nei preferiti. Per aumentare le prestazioni possono essere aggiunti degli indici sul campo postdate. La relazione source vincola l'entità News ad un feed RSS.

Entità News_Source

Nome entità: News_Source			
Attributi			
Nome	Tipo	Opzioni	
link	String	Mandatory	
name	String	Mandatory	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
articles	News (source)	Optional, To-Many	Cascade
Ruolo e responsabilità:	Specifica un feed dal quale eseguire il download di più articoli.		
Politiche caching:	Persistente.		
Origine dati:	File Property List incluso nel *bundle*.		

Tabella 8.3: Proprietà entità News_Source.

NOTE: Il campo link rappresenta la sorgente del feed RSS. Campo name possibilmente da indicizzare.

Struttura dati Eventi

La struttura dati del sottosistema Eventi è composta dalle seguenti entità (in ordine alfabetico):

- Assignment_Event;

- Calendar;
- Event;
- Exam_Event;
- Lecture_Event;
- Personal_Event;
- University_Event.

Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

Entità Assignment_Event

Nome entità: Assignment_Event Parent: Exam_Event			
Attributi			
Nome	Tipo	Opzioni	
descr	String	Optional	
priority	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
Ruolo e responsabilità:	Rappresenta un evento assignment riferito ad un oggetto Course. Da elencare nel calendario delle lezioni del corso correlato.		
Politiche caching:	Persistente.		
Origine dati:	Creato dall'utente.		

Tabella 8.4: Proprietà entità Assignment_Event.

NOTE: priority descrive la priorità dell'evento rispetto ad uno dello stesso tipo; di utilità per l'utente.

Entità Calendar

Nome entità: Calendar			
Attributi			
Nome	Tipo	Opzioni	
color	Transformable	Optional	
ecid	String	Mandatory	
name	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
events	Event (calendar)	Optional, To-Many	Cascade
ordinaryCourses	OrdinaryCourse (calendar)	Optional, To-Many	Nullify
Ruolo e responsabilità:	Corrisponde ad un calendario di eventi, associabile ai corsi inseriti dall'utente nella lista "My courses".		
Politiche caching:	Persistente.		
Origine dati:	File Property List incluso nel *bundle*.		

Tabella 8.5: Proprietà entità Calendar.

NOTE: Il campo ecid identifica univocamente il calendario ed è indispensabile per il download dei dati JSON per popolare la relazione events. color è un campo Transformable in quanto è un'istanza della classe *UIColor* da convertire per garantirne la persistenza. Campo name indicizzabile per maggiori prestazioni.

Entità Event

Nome entità: Event		
Attributi		
Nome	Tipo	Opzioni
duration	Integer 16 (default value: 0)	Optional, Transient
eid	String	Mandatory
end	Date	Optional

isFavorite	Boolean (default value: NO)	Mandatory
location	String	Optional
note	String	Optional
start	Date	Optional
title	String	Optional

Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
buildingArea	Building_Area (events)	Optional	Nullify
calendar	Calendar (events)	Optional	Nullify

Ruolo e responsabilità:	Identifica un evento appartenente ad un calendario ed eventualmente correlato ad una locazione.
Politiche caching:	Eliminabile automaticamente ad un anno dalla sua scadenza.
Origine dati:	Sorgente JSON.

Tabella 8.6: Proprietà entità Event.

NOTE: duration è di tipo Transient, in quanto è calcolato in base ai campi start ed end. eid identifica univocamente l'evento. isFavorite è settato a YES se aggiunto dall'utente nei preferiti. location è utilizzato quando il luogo dell'evento non corrisponde ad alcuna locazione esistente nella struttura dati.

Entità Exam_Event

Nome entità: Exam_Event Parent: Event		
Attributi		
Nome	Tipo	Opzioni
duedate	Date	Optional, Transient

Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
course	Ordinary_Course (exams)	Mandatory	Nullify

Ruolo e responsabilità:	Rappresenta un evento exam riferito ad un oggetto Course. Da elencare nel calendario delle lezioni del corso correlato.
Politiche caching:	Persistente.

Origine dati:	Creato dall'utente.

Tabella 8.7: Proprietà entità Exam_Event.

NOTE: dueDate è di tipo Transient, in quanto è calcolato in base ai campi ereditati start ed end. L'entità è vincolata al corso rispetto al quale viene creato l'evento esame.

Entità Lecture_Event

Nome entità: Lecture_Event Parent: Personal_Event			
Attributi			
Nome	Tipo	Opzioni	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
hours	Hours (lecture)	Optional	Cascade
persons	Directory_Person (lectures)	Optional, To-Many	Nullify
Ruolo e responsabilità:	Rappresenta un evento lecture che può ripetersi nel tempo sulla base degli orari definiti. E' correlato ad uno o più docenti.		
Politiche caching:	Persistente.		
Origine dati:	Creato dall'utente.		

Tabella 8.8: Proprietà entità Lecture_Event.

NOTE: la relazione hours serve per facilitare la modellazione di un evento di questo tipo che può ripetersi più volte la settimana in orari diversi.

Entità Personal_Event

Nome entità: Personal_Event Parent: Event		
Attributi		
Nome	Tipo	Opzioni
alert	Transformable	Optional

repeat	Boolean (default value: NO)	Optional
Relazioni		
Nome	Destinazione (inversa)	Opzioni	Regola delete
Ruolo e responsabilità:	Rappresenta un evento personale, sincronizzabile col calendario di sistema.
Politiche caching:	Persistente.
Origine dati:	Creato dall'utente.

Tabella 8.9: Proprietà entità Personal_Event.

NOTE: Il tipo dei campi alert e repeat dipende dal framework *EventKit*.

Entità University_Event

Nome entità: University_Event Parent: Event			
Attributi			
Nome	Tipo	Opzioni	
details	String	Optional	
organizationBody	String	Optional	
rapporteur	String	Optional	
topic	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
alerts	Alert (event)	Optional, To-Many	Cascade
links	Link (universityEvent)	Optional, To-Many	Cascade
persons	Directory_People (universityEvents)	Optional, To-Many	Nullify
Ruolo e responsabilità:	Corrisponde ad un evento organizzato dall'Università. Vi possono essere degli avvisi ad esso riferiti. Può avere relazioni con più docenti.		
Politiche caching:	Eliminabile automaticamente ad un anno dalla sua scadenza.		
Origine dati:	Sorgente JSON.		

Tabella 8.10: Proprietà entità University_Event.

NOTE: La relazione alerts consente di aprire il profilo dell'evento direttamente dalla descrizione dell'avviso ricevuto.

Struttura dati Directory

La struttura dati del sottosistema Directory è composta dalla seguente entità:

- Directory_People.

La tabella seguente descrive le sue proprietà.

Entità Directory_People

Nome entità: Directory_People		
Attributi		
Nome	Tipo	Opzioni
dept	String	Optional
givenname	String	Mandatory, Indexed
isFavorite	Boolean (default value: NO)	Mandatory
isRecent	Boolean (default value: NO)	Mandatory
persid	String	Mandatory
surname	String	Mandatory, Indexed
timestamp	Date	Mandatory, Indexed
title	String	Optional

Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
addresses	Address (person)	Optional, To-Many	Cascade
alerts	Alert (person)	Optional, To-Many	Nullify
emails	Email (person)	Optional, To-Many	Cascade
image	Image (person)	Optional	Cascade
lectures	Lecture_Event (persons)	Optional, To-Many	Nullify
links	Link (person)	Optional, To-Many	Cascade
numbers	Contact_Number (person)	Optional, To-Many	Cascade
ordinaryCourses	OrdinaryCourse (persons)	Optional, To-Many	Nullify

universityEvents	University_Event (persons)	Optional, To-Many	Nullify
Ruolo e responsabilità:	Modella tutte le informazioni inerenti un docente inserito nella directory. Può essere esteso ad altri tipi di persone della directory.		
Politiche caching:	Caching recenti e preferiti.		
Origine dati:	Sorgente JSON/LDAP.		

Tabella 8.11: Proprietà entità Directory_People.

NOTE: Le numerose relazioni di questa entità consentono il riutilizzo di campi in comune con altri oggetti, e in particolare offre la possibilità di aprire il profilo di un docente da sottosistemi diversi dell'applicazione. I campi isFavorite e isRecent fungono da discriminanti per elaborare le rispettive liste. Il campo hours non è modellato dalla corrispondente entità perché ciò non è considerato necessario. persid identifica univocamente una persona anche nella directory. givenname, surname e timestamp sono indicizzati per accelerare l'ordinamento

Struttura dati Insegnamenti

La struttura dati del sottosistema Insegnamenti è composta dalle seguenti entità (in ordine alfabetico):

- Course;
- Degree;
- Ordinary_Course.

Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

Entità Course

Nome entità: Course		
Attributi		
Nome	Tipo	Opzioni
code	String	Mandatory
color	Transformable	Optional
isRecent	Boolean (default value: NO)	Mandatory

name	String	Mandatory
Relazioni		

Nome	Destinazione (inversa)	Opzioni	Regola delete
borrowedIn	Ordinary_Course (borrowedCourses)	Optional	Nullify
degree	Degree (courses)	Mandatory	Nullify

Ruolo e responsabilità:	Struttura corrispondente all'insieme delle informazioni necessarie per identificare univocamente un insegnamento.
Politiche caching:	Caching recenti.
Origine dati:	Sorgente JSON.

Tabella 8.12: Proprietà entità Course.

NOTE: Il campo code identifica univocamente un insegnamento in tutto il sistema, isRecent è un discriminante per consentire il caching dei dati e color è un campo Transformable in quanto è un'istanza della classe *UIColor*. Campo name indicizzabile per maggiori prestazioni. La relazione borrowedIn identifica un insegnamento mutuato, degree invece vincola l'entità al corso di laurea di appartenenza.

Entità Degree

Nome entità: Degree		
Attributi		
Nome	Tipo	Opzioni
code	String	Mandatory
isMaster	Boolean (default value: NO)	Mandatory
name	String	Mandatory
Relazioni		

Nome	Destinazione (inversa)	Opzioni	Regola delete
courses	Course (degree)	Optional, To-Many	Cascade
department	Building_Area (degrees)	Optional	Nullify

Ruolo e responsabilità:	Fornisce le informazioni essenziali per modellare un corso di laurea, il suo dipartimento e tutti gli insegnamenti appartenenti.
Politiche caching:	Persistente.
Origine dati:	File Property List incluso nel *bundle*.

Tabella .8.13: Proprietà entità Degree.

NOTE: Il campo code identifica univocamente un corso di laurea in tutto il sistema, isMaster permette di filtrare i corsi di laurea magistrali. Campo name indicizzabile. La relazione department pone intrinsecamente un vincolo di appartenenza ad un dipartimento, ma ciò non è esplicitabile poiché Building_Area è di competenza di un altro sottosistema.

Entità Ordinary_Course

Nome entità: Ordinary_Course Parent: Course		
Attributi		
Nome	Tipo	Opzioni
academicYear	String	Optional
attendance	String	Optional
courseYear	String	Optional
credits	String	Optional
descr	String	Optional
examination	String	Optional
level	String	Optional
note	String	Optional
partition	String	Optional
period	String	Optional
sector	String	Optional
textbook	String	Optional

Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
alerts	Alert (Ordinary_Course)	Optional, To-Many	Cascade
borrowedCourses	Course (borrowedIn)	Optional, To-Many	Nullify
calendar	Calendar (ordinaryCourses)	Optional	Nullify
exams	Exam_Event (course)	Optional, To-Many	Cascade
hours	Hours (ordinaryCourse)	Optional	Cascade

persons	Directory_Person (ordinaryCourses)	Optional, To-Many	Nullify
relatedOrdinaryCourses	Ordinary_Course (relativeCourse)	Optional, To-Many	Nullify
relativeCourse	Ordinary_Course (relatedOrdinaryCourse)	Optional	Nullify
Ruolo e responsabilità:	Entità comprendente tutte le informazioni dettagliate di un insegnamento e la sua relazione con altri insegnamenti, docenti, avvisi e il rispettivo calendario.		
Politiche caching:	Caching recenti e lista “My Courses”.		
Origine dati:	Sorgente JSON.		

Tabella 8.14: Proprietà entità Ordinary_Course.

NOTE: I campi sono stati lasciati opzionali per avere maggiore libertà sui dati ottenuti dalla sorgente. La relazione relatedOrdinaryCourses e la sua inversa relativeCourse, permettono di modellare gli insegnamenti correlati. Le restanti relazioni permettono di individuare docenti, avvisi ed eventi relativi all’insegnamento.

Struttura dati Mappa

La struttura dati del sottosistema Mappa è composta dalle seguenti entità (in ordine alfabetico):

- Building;
- Building_Area;
- Building_Type;
- Building_Details.

Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

Entità Building

Nome entità: Building		
Attributi		
Nome	Tipo	Opzioni

code	String		Mandatory
latitude	String		Mandatory
longitude	String		Mandatory
name	String		Mandatory
polygon	String		Optional
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
address	Address (building)	Optional	Cascade
areas	Building_Area (building)	Optional, To-Many	Cascade
details	Building_Details (building)	Optional	Cascade
type	Building_Type (buildings)	Mandatory	Nullify
Ruolo e responsabilità:	Ha la responsabilità di localizzare una sede sulla mappa e di identificare le aree e i servizi annessi all'edificio.		
Politiche caching:	Persistente.		
Origine dati:	File Property List incluso nel *bundle*.		

Tabella 8.15: Proprietà entità Building.

NOTE: Il campo code identifica univocamente una sede, latitude e longitude sono indispensabili per localizzare la sede sulla mappa, mentre polygon include ulteriori coordinate nel caso si debba evidenziare l'area interessata dall'edificio. La relazione type pone un vincolo sul tipo di edificio, in questo caso una sede. Campo name indicizzabile.

Entità Building_Area

Nome entità: Building_Area			
Attributi			
Nome	Tipo		Opzioni
code	String		Optional
isFavorite	Boolean (default value: NO)		Mandatory
isRecent	Boolean (default value: NO)		Mandatory
name	String		Mandatory
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete

building	Building (areas)	Mandatory	Nullify
degrees	Degree (department)	Optional, To-Many	Nullify
details	Building_Details (area)	Optional	Cascade
events	Event (buildingArea)	Optional, To-Many	Nullify
hours	Hours (area)	Optional	Cascade
members	Building_Area (parent)	Optional, To-Many	Cascade
parent	Building_Area (members)	Optional	Nullify
type	Building_Type (area)	Mandatory	Nullify

Ruolo e responsabilità:	Rappresenta un'area della rispettiva sede appartenente ad una precisa tipologia di servizi. Corrisponde ad esempio ad un dipartimento, un ufficio oppure una biblioteca.
Politiche caching:	Caching preferiti e recenti.
Origine dati:	Sorgente JSON.

Tabella 8.16: Proprietà entità Building_Area.

NOTE: Il campo code, lasciato opzionale per maggiore libertà di modellazione, identifica univocamente un'area di una sede; isFavorite e isRecent servono invece da discriminanti per elaborare le rispettive liste. Le relazioni type e building pongono un vincolo di associazione con una sede e una tipologia, mentre members e parent permettono di modellare aree dipendenti l'una dall'altra. La relazione events consente, infine, di risalire agli eventi svolti nella determinata area. Il campo name è ulteriormente indicizzabile.

Entità Building_Type

Nome entità: Building_Type		
Attributi		
Nome	Tipo	Opzioni
name	String	Mandatory

Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
areas	Building_Area (areas)	Optional, To-Many	Cascade
building	Building (type)	Optional, To-	Cascade

		Many	
Ruolo e responsabilità:	Definisce una tipologia di struttura universitaria, ovvero un insieme di sedi o di aree adibite a un servizio.		
Politiche caching:	Persistente.		
Origine dati:	File Property List incluso nel *bundle*.		

Tabella 8.17: Proprietà entità Building_Type.

Entità Building_Details

Nome entità: Building_Details			
Attributi			
Nome	Tipo	Opzioni	
note	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
area	Building_Area (details)	Optional	Nullify
building	Building (details)	Optional	Nullify
emails	Email (buildingDetails)	Optional, To-Many	Cascade
image	Image (buildingDetails)	Optional	Cascade
links	Link (buildingDetails)	Optional, To-Many	Cascade
numbers	Contact_Number (buildingDetails)	Optional, To-Many	Cascade
Ruolo e responsabilità:	Le sue proprietà descrivono i dettagli di una locazione presente sulla mappa.		
Politiche caching:	Segue la politica della rispettiva entità Building o Building_Area.		
Origine dati:	Sorgente JSON.		

Tabella 8.18: Proprietà entità Building_Details.

NOTE: Le relazioni area e building costituirebbero un vincolo di appartenenza, ma non dello stesso oggetto a entrambe; per questo sono lasciate opzionali.

Struttura dati Social

La struttura dati del sottosistema Social è composta dalle seguenti entità (in ordine alfabetico):

- Social_Category;
- Social_Media.

Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

Entità Social_Category

<table>
<tr><td colspan="4">Nome entità: Social_Category
Parent: Link_Category</td></tr>
<tr><td colspan="4">Attributi</td></tr>
<tr><td colspan="2">Nome</td><td>Tipo</td><td>Opzioni</td></tr>
<tr><td colspan="4">Relazioni</td></tr>
<tr><td>Nome</td><td>Destinazione (inversa)</td><td>Opzioni</td><td>Regola delete</td></tr>
<tr><td>media</td><td>Social_Media (categories)</td><td>Mandatory, To-Many</td><td>Nullify</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="2">Ruolo e responsabilità:</td><td colspan="2">Definisce delle categorie nelle quali raggruppare i social network.</td></tr>
<tr><td colspan="2">Politiche caching:</td><td colspan="2">Persistente.</td></tr>
<tr><td colspan="2">Origine dati:</td><td colspan="2">File Property List incluso nel bundle.</td></tr>
</table>

Tabella 8.19: Proprietà entità Social_Category.

NOTE: La relazione media permette di associare una categoria ad uno o più social media.

Entità Social_Media

<table>
<tr><td>Nome entità: Social_Media</td></tr>
<tr><td>Attributi</td></tr>
</table>

Nome	Tipo	Opzioni
name	String	Mandatory
Relazioni		
Nome	Destinazione (inversa)	Opzioni	Regola delete
categories	Social_Categories (media)	Optional, To-Many	Nullify
Ruolo e responsabilità:	Rappresenta la categoria di un social network.
Politiche caching:	Persistente.
Origine dati:	File Property List incluso nel *bundle*.

Tabella 8.20: Proprietà entità Social_Category.

Struttura dati Risorse

La struttura dati del sottosistema Risorse è composta dalla seguente entità:

- Resource_Category

Le sue proprietà sono descritte nella tabella seguente.

Entità Resource_Category

Nome entità: Resource_Category Parent: Link_Category			
Attributi			
Nome	Tipo	Opzioni	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
parent	Resource_Category (subcategory)	Optional	Nullify
subcategory	Resource_Category (parent)	Optional, To-Many	Cascade
Ruolo e responsabilità:	Rappresenta la categoria di una risorsa.		
Politiche caching:	Persistente.		
Origine dati:	File Property List incluso nel *bundle*.		

Tabella 8.21: Proprietà entità Social_Category.

NOTE: La relazione subcategory e la sua inversa, parent, permettono la gestione di sottocategorie.

Struttura dati Avvisi

La struttura dati del sottosistema Avvisi è composta dalle seguenti entità (in ordine alfabetico):

- Alert;
- Alert_Type.

Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

Entità Alert

Nome entità: Alert		
Attributi		
Nome	Tipo	Opzioni
descr	String	Optional
timestamp	Date	Mandatory
title	String	Mandatory

Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
event	University_Event (alerts)	Optional	Nullify
link	Link (alert)	Optional	Cascade
ordinaryCourse	Ordinary_Course (alerts)	Optional	Nullify
person	Directory_Person (alerts)	Optional	Nullify
type	Alert_Type (alerts)	Mandatory	Nullify

Ruolo e responsabilità:	Rappresenta il modello di un avviso nell'intero sistema. Il contenuto può riferirsi ad entità di sottosistemi diversi.
Politiche caching:	Caching a scadenza in base al valore del campo timestamp.
Origine dati:	Sorgente JSON.

Tabella 8.22: Proprietà entità Alert.

NOTE: La relazione type descrive un vincolo di appartenenza di un avviso ad una rispettiva categoria.

Entità Alert_Category

Nome entità: Alert_Category			
Attributi			
Nome	Tipo	Opzioni	
name	String	Mandatory	
source	String	Mandatory	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
alerts	Alert (type)	Optional, To-Many	Cascade
Ruolo e responsabilità:	Rappresenta la categoria di un avviso.		
Politiche caching:	Persistente.		
Origine dati:	File Property List incluso nel *bundle*.		

Tabella 8.23: Proprietà entità Alert_Category.

NOTE: Il campo source specifica l'indirizzo della sorgente dati della categoria.

Strutture dati comuni

Le strutture dati elencate di seguito (in ordine alfabetico) rappresentano le entità in comune tra tutti i sottosistemi, introdotte separatamente per evitare inutili replicazioni.

- Address;
- Contact_Number;
- Email;
- Hours;
- Image;
- Link;

- Link_Category.

  Le tabelle seguenti descrivono le proprietà di ciascuna entità elencata.

NOTE: Le politiche di caching e le sorgenti dati sono le stesse delle entità a cui esse appartengono.

Entità Address

Nome entità: Address			
Attributi			
Nome	Tipo	Opzioni	
city	String	Optional	
state	Date	Optional	
street	String	Optional	
zip	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
building	Building (address)	Optional	Nullify
person	Directory_Person(addresses)	Optional	Nullify
Ruolo e responsabilità:	Definisce la struttura dati di un indirizzo.		

Tabella 8.24: Proprietà entità Address.

Entità Contact_Number

Nome entità: Contact_Number		
Attributi		
Nome	Tipo	Opzioni
isFAX	Boolean (default value: NO)	Mandatory
isPhone	Boolean (default value: NO)	Mandatory
number	String	Mandatory
Relazioni		

Nome	Destinazione (inversa)	Opzioni	Regola delete
buildingDetails	Building_Details (numbers)	Optional	Nullify
person	Directory_Person(numbers)	Optional	Nullify
Ruolo e responsabilità:	Rappresenta un numero di telefono o di FAX.		

Tabella 8.25: Proprietà entità Contact_Number.

NOTE: I campi isFAX e isPhone determinano qualora il numero risulti un telefono, un FAX o entrambi.

Entità Email

Nome entità: Email			
Attributi			
Nome	Tipo	Opzioni	
mailbox	String	Mandatory	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
buildingDetails	Building_Details (emails)	Optional	Nullify
person	Directory_Person (emails)	Optional	Nullify
Ruolo e responsabilità:	Rappresenta una casella e-mail.		

Tabella 8.26: Proprietà entità Email.

Entità Hours

Nome entità: Hours		
Attributi		
Nome	Tipo	Opzioni
friday	String	Optional
monday	String	Optional
saturday	String	Optional

thursday	String	Optional	
tuesday	String	Optional	
wednesday	String	Optional	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
buildingArea	Building_Area (hours)	Optional	Nullify
lecture	Lecture_Event (hours)	Optional	Nullify
ordinaryCourse	Ordinary_Course (hours)	Optional	Nullify
Ruolo e responsabilità:	Permette di modellare gli orari settimanali di una struttura, di una lezione oppure di un insegnamento.		

Tabella 8.27: Proprietà entità Hours.

Entità Image

Nome entità: Image			
Attributi			
Nome	Tipo	Opzioni	
name	String	Optional	
image	Transformable	Mandatory	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
buildingDetails	Building_Details (image)	Optional	Nullify
person	Directory_Person (image)	Optional	Nullify
Ruolo e responsabilità:	Rappresenta un'immagine convertibile in base64.		

Tabella 8.28: Proprietà entità Image.

NOTE: Il campo image è di tipo Transformable in quanto un'immagine deve essere convertita in base64 per poter essere memorizzata.

Entità Link

Nome entità: Link			
Attributi			
Nome	Tipo	Opzioni	
descr	String	Optional	
name	String	Optional	
url	String	Mandatory	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
alert	Alert (link)	Optional	Nullify
buildingDetails	Building_Details (link)	Optional	Nullify
category	Link_Category (links)	Optional	Nullify
person	Directory_Person (links)	Optional	Nullify
universityEvent	University_Event (link)	Optional	Nullify
Ruolo e responsabilità:	Rappresenta un collegamento URL.		

Tabella 8.29: Proprietà entità Link.

Entità Link_Category

Nome entità: Link_Category			
Attributi			
Nome	Tipo	Opzioni	
name	String	Mandatory	
Relazioni			
Nome	Destinazione (inversa)	Opzioni	Regola delete
links	Link (category)	Optional, To-Many	Cascade
Ruolo e responsabilità:	Permette ai link di essere suddivisi in categorie.		

Tabella 8.30: Proprietà entità Link_Category.

OSSERVAZIONI SULL'ELIMINAZIONE E IL CACHING DEI DATI

L'applicazione, sia per il modo in cui è strutturata, ma soprattutto per la necessità di presentare all'utente informazioni sempre valide e aggiornate, fa affidamento su una sorgente dati che andrebbe integrata direttamente nel sistema informativo dell'Università. Sebbene la progettazione e le caratteristiche di questa componente lato server non rientrino nell'ambito di questa tesi, verranno comunque introdotte in questo documento. In particolare in questo paragrafo vengono puntualizzati fin da subito alcuni aspetti che interessano il mantenimento della struttura dati di iCa'Foscari.

Guardando le schede descrittive delle entità si nota che per le relazioni sono state definite delle regole di eliminazione. Queste regole hanno sicuramente un'importanza logica e concettuale nel progetto della struttura dati, tuttavia le azioni di eliminazione riguardano solamente una piccola parte degli oggetti, come si evince anche dai casi d'uso, per cui spesso queste regole risulterebbero facoltative oppure addirittura sostituibili in favore della regola *No Action*.

L'eliminazione dei dati, infatti, deve riguardare solo quelli creati dall'utente o il cui salvataggio è attivato espressamente da una sua azione. Inoltre, a fronte di alcuni requisiti non funzionali, l'applicazione prevede il caching dei dati scaricati dal server. Si ricorda, però, che tutte le informazioni elaborate devono essere sempre *up to date*, dunque è indispensabile mantenere aggiornata anche il modello dati dell'applicazione.

La sincronizzazione delle "*cache*" può così essere effettuata con un banale *UPDATE* dei dati, ciò nonostante potrebbe essere richiesta l'eliminazione di alcune informazioni divenute obsolete oppure più semplicemente rimosse dalla sorgente dati.

Pertanto, entrambi i processi di *caching* e sincronizzazione dei dati dovranno essere riveduti prima della fase di implementazione, apportando le dovute modifiche alle

regole di eliminazione e assegnando eventuali nuove responsabilità ai controller; garantendo l'integrità e la sicurezza della struttura dati.

SUPPORTO ALLA MIGRAZIONE DEI DATI

Sia nella fase di implementazione (o di testing) di iCa'Foscari, che nel corso del suo ciclo di vita, la struttura dati è soggetta a delle modifiche più o meno importanti; come ad esempio l'aggiunta di nuovi attributi o l'eliminazione di un'intera entità.

Se vengono apportate delle modifiche al modello dati, i dati esistenti, che risiedono nel *persistent store* sul dispositivo di un beta tester o di un utente, diventano inutilizzabili nella nuova *build* dell'applicazione.

Per ovviare a tale spiacevole situazione, a maggior ragione se si tratta di una modifica effettuata in occasione di una nuova *release*, è necessario predisporre l'applicazione alla migrazione dei dati. Ciò è possibile attivando il *versioning* del modello dati in *XCode* e utilizzando uno dei metodi di migrazione supportati da *Core Data*[8]:

- *lightweight migration* oppure,
- *standard migration*.

Quando *Core Data* rileva che il *persistent store* in uso è incompatibile con la struttura dati corrente, lancia un'eccezione. La soluzione consiste nel predisporre una migrazione per dire a *Core Data* come spostare i dati dal vecchio *persistent store* a quello nuovo corrispondente al *data model* attuale.

Nel caso le modifiche alla struttura dati siano semplici, come aggiungere o rimuovere un attributo da un'entità, *Core Data* è perfettamente in grado di risolvere

[8]http://developer.apple.com/documentation/Cocoa/Conceptual/CoreDataVersioning/index.html

autonomamente la migrazione dei dati esistenti nel nuovo modello. Questo è il caso della migrazione *lightweight*.

Se invece vengono apportate delle modifiche molto più complesse, è necessario utilizzare una migrazione *standard*. Ciò consiste nel creare un modello di mapping ed eventualmente nello scrivere del codice per informare *Core Data* su come spostare i dati tra i due *persistent store*.

Nella fase di sviluppo di iCa'Foscari potrà essere adottata fin da subito la migrazione *lightweight*, apportando le dovute modifiche all'*iCaFoscariAppDelegate*. Nel caso questo metodo dovesse fallire, l'eventuale perdita dei dati in questa fase non costituirebbe un problema serio. Nella fase di testing, così come nelle *release* successive, è decisamente consigliato l'utilizzo della migrazione *standard*; al fine di garantire i criteri di scalabilità e robustezza dell'applicazione.

Nel caso di migrazioni irrisolvibili da una versione all'altra, l'utente dovrà essere informato della perdita dei dati prima dell'aggiornamento.

CONCLUSIONI

Lo scopo di questo capitolo è stato quello di fornire informazioni dettagliate sulla versione iniziale della struttura dati di iCa'Foscari. Ciascuna entità del modello presentato è stata descritta in modo esauriente, al fine di fornire tutti i chiarimenti necessari alla fase di integrazione e conformazione col sistema informativo dell'Università Ca' Foscari, propedeutica all'implementazione effettiva dell'applicazione.

Per di più, sono stati espressi alcuni orientamenti sulle procedure di *caching* e eliminazione da adottare, e sulla predisposizione alla migrazione dei dati tra versioni diverse del *data model*.

Capitolo 9: Progettazione della GUI

Il lavoro di progettazione di iCa'Foscari descritto in questa tesi si conclude con la costruzione dei *wireframe* dell'interfaccia utente. Oltre a questi ultimi, in questo capitolo sono dettate alcune specifiche riguardanti,tra le altre, la consistenza nella presentazione dei contenuti e il comportamento dell'applicazione nella navigazione tra schermate diverse. Inoltre, alcuni diagrammi descrivono il collegamento tra le varie interfacce dell'applicazione.

SPECIFICHE PER IL DESIGN DELL'UI

Il design dell'interfaccia utente è la fase più delicata nella progettazione di un'applicazione per *iOS*. Tale processo deve infatti far fronte a diversi requisiti, a partire dalle linee guida definite da Apple, la quale assume un sorprendente livello di controllo sullo stile a cui le applicazioni sviluppate su questi dispositivi devono aderire. Tali requisiti si riferiscono, infatti, alla qualità del *look and feel* dell'applicazione, alla risoluzione da supportare e ad un dimensionamento delle aree dell'interfaccia adeguato al *finger input*. La corretta implementazione di questi requisiti contribuisce alla realizzazione di una *UI* consistente con le funzionalità dell'applicazione e apprezzabile dall'utente.

Consistenza

Relativamente alla consistenza dell'interfaccia grafica, Apple esprime il proprio concetto di *"integrità estetica"*[9]:

[9]http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Principles/Principles.html

"L'integrità estetica non è la misura di quanto bella sia un'applicazione. E' una misura di quanto bene l'aspetto di un'applicazione si integra con le sue funzionalità. Ad esempio, un'applicazione di produttività dovrebbe porre in secondo piano gli elementi decorativi, dando invece prominenza al task dell'utente, fornendo controlli e comportamenti standard.

Dall'altro capo della corda si trovano invece le applicazioni immersive, dalle quali l'utente esige un bell'aspetto che prometta divertimento e incoraggi la scoperta di nuovi contenuti. Tuttavia, nonostante un'applicazione immersiva tenda a focalizzarsi sulla diversione dell'utente, il suo aspetto deve comunque integrarsi con il task. Pertanto, il design degli elementi dell'interfaccia utente deve essere svolto con cura, al fine di fornire un'esperienza internamente coerente e consistente."

Il design dell'interfaccia, dunque, non deve essere fine a se stesso. L'applicazione è progettata per degli scopi ben precisi e per aiutare l'utente a completare dei task. L'interfaccia grafica ha il compito di rendere tutto ciò il più facile e apprezzabile possibile.

Non solo, i dispositivi *iOS* sono operati dall'utente esclusivamente per mezzo delle proprie dita, che, secondo Apple, sono l'unico mezzo di puntamento di cui ha bisogno:

"Vi sono dei vantaggi reali nell'utilizzo delle dita per far funzionare un dispositivo: Queste sono sempre disponibili, sono in grado di fare molti movimenti diversi, e danno all'utente un senso di immediatezza e connessione al dispositivo che è impossibile da ottenere con un dispositivo di input esterno, come un mouse.

Tuttavia, le dita hanno un grande svantaggio: Esse sono molto più grandi di un puntatore del mouse, indipendentemente dalla loro dimensione, la loro forma, o la destrezza del loro possessore. Nel contesto di un display, le dita non possono essere mai tanto precise quanto il puntatore di un mouse."

Nel design dell'interfaccia, è importante tener presente questo fatto. A tale proposito è necessario seguire fedelmente le linee guida di Apple, indicate nei requisiti di questa applicazione, dove sono riportate tutte le specifiche riguardanti il *look and feel* e le dimensioni delle aree dell'*UI*.

Risoluzioni dello schermo

La risoluzione standard, o meglio di base, dei dispositivi *iOS*è di 480x320 pixel con una densità di 163 ppi. Tuttavia, con l'ultima *release* dell'iPhone, grazie alla nuova tecnologia *retina display*, essa è raddoppiata a 960x640 pixel, raggiungendo una densità di pixel eccezionale di 336 ppi. La risoluzione di iPad è invece di 1024x768 pixel a 132 ppi.

Per i dispositivi iPod Touch e iPhone devono essere inserite nel *bundle* due versioni di ogni stile applicato agli elementi grafici nativi di *iOS* e di ogni icona o immagine utilizzata per decorare l'interfaccia utente: Una versione nella risoluzione di base e un'altra, con suffisso @2x nel nome del file, nella nuova risoluzione *retina*. Il sistema grazie al suffisso del file, effettua automaticamente la selezione delle risorse giuste senza alcuna logica aggiuntiva nel codice. Le risorse grafiche di iPad devono essere trattate separatamente, in quanto i file con suffisso @2x non hanno sempre una risoluzione adeguata. Allo stesso modo, per iPad è consigliabile creare file *nib* specifici.

I *wireframe* presentati in questo capitolo si riferiscono esclusivamente alla versione per iPod/iPhone dell'applicazione. Inizialmente, iCa'Foscari può supportare da subito l'iPad in qualità HD, utilizzando gli stessi *wireframe* con delle risorse di adeguate dimensioni; mentre in futuro potrà essere preso in considerazione il design di un'*UI* specifica per iPad capace di sfruttare tutte le sue funzionalità.

Al fine di supportare la risoluzione di tutti i dispositivi *iOS* è fondamentale evitare l'*hard coding* delle dimensioni degli oggetti grafici nel codice sorgente; nel caso ciò non fosse possibile è necessario allora utilizzare una logica condizionale per il targeting del dispositivo corretto.

Autorotation

Grazie agli accelerometri integrati nei dispostivi *iOS*, le applicazioni possono rispondere alle rotazioni orientando a loro volta il contenuto dello schermo.

Nel caso specifico di iCa'Foscari, dovranno rispondere alle rotazioni solamente le schermate che contengono una *UIWebView*, al fine di consentire all'utente di visualizzare meglio le pagine web.

Accorgimenti prestazionali

Anche nella realizzazione degli *asset* della *UI* sorgono delle problematiche legate alle prestazioni di *rendering* limitate, soprattutto per i dispositivi non provvisti del nuovo processore Apple A4.

Infatti, il calcolo dell'*alpha transparency* e il ridimensionamento delle risorse grafiche contribuisce fortemente al decadimento delle prestazioni dell'applicazione. Per ovviare a tale problema è necessario fin da subito eliminare ogni tipo di trasparenza calcolata a tempo di esecuzione e pre-elaborarla già quando la risorsa viene disegnata nell'apposita applicazione di grafica *raster* o vettoriale. Analogamente, tutte le immagini e i componenti dello stile grafico devono essere delle stesse dimensioni di come vengono presentate sullo schermo, evitando il calcolo di inutili ridimensionamenti a *runtime*.

Infine, è importante ottimizzare la dimensione su disco di ciascuna risorsa ed è ancora più importante riutilizzarle il più possibile. Infatti, poiché gli accessi alla memoria *FLASH* sono molto costosi in termini di tempo, il sistema mantiene in cache tutte le risorse caricate dal *filesystem*, rendendole rapidamente disponibili agli accessi futuri.

ICONA DELL'APPLICAZIONE

L'applicazione per poter essere installata su *iOS* deve includere il seguente set di icone:

Descrizione	Dimensione per iPhone e iPodTouch (in pixel)	Dimensione per iPad (in pixel)
IconaApplicazione (obbligatoria)	57x57 114x114 (retina)	72x72
Icona App Store (obbligatoria)	512x512	512x512
Icona Spotlight e Impostazioni (consigliata)	29x29 58x58 (retina)	50x50 per Spotlight 29x29 per Settings
Launch image (obbligatoria)	320x480 640x960 (retina)	768x1004 (portrait) 1024x768 (landscape)

Tabella 9.1: Set di icone per l'applicazione.

Di seguito è riportato un esempio sia dell'icona che dell'immagine di caricamento di iCa'Foscari.

Figura 9.1: IconadiiCa'Foscari.

Figura 9.2: Launch image di iCa'Foscari.

COLORI E STILE DELL'APPLICAZIONE

L'aspetto dell'applicazione deve conformarsi ai colori e allo stile che verranno stabiliti dagli organi competenti dell'Università Ca' Foscari di Venezia. Il prototipo sviluppato utilizza il colore #660000, prelevato dal logo di Ca' Foscari[10].

[10]http://www.scambieuropei.com/wp-content/uploads/2010/03/ca-foscari-logo.jpg

WIREFRAME E DIAGRAMMI DI COLLEGAMENTO

Nelle pagine seguenti sono presentati i *wireframe* delle interfacce grafiche di iCa'Foscari. Essi stabiliscono la struttura iniziale dell'interfaccia utente. Qualsiasi colore, immagine o icona, dovranno essere pertanto ignorati.

Per ogni funzionalità principale dell'applicazione, è disegnato un diagramma che illustra il collegamento tra le varie schermate, referenziate attraverso dei numeri. Questi diagrammi permettono di capire il rapporto tra un'interfaccia e l'altra e come l'utente può muoversi tra loro. Le connessioni orizzontali indicano che la navigazione avviene attraverso un *UINavigationController*, oppure mediante la selezione degli elementi di una *UITabBar*. Le linee verticali, invece, stabiliscono che la schermata deve essere presentata *modalmente* attraverso un *modalViewController*. Le immagini riportate di seguito illustrano le differenze tra questi due metodi di presentazione.

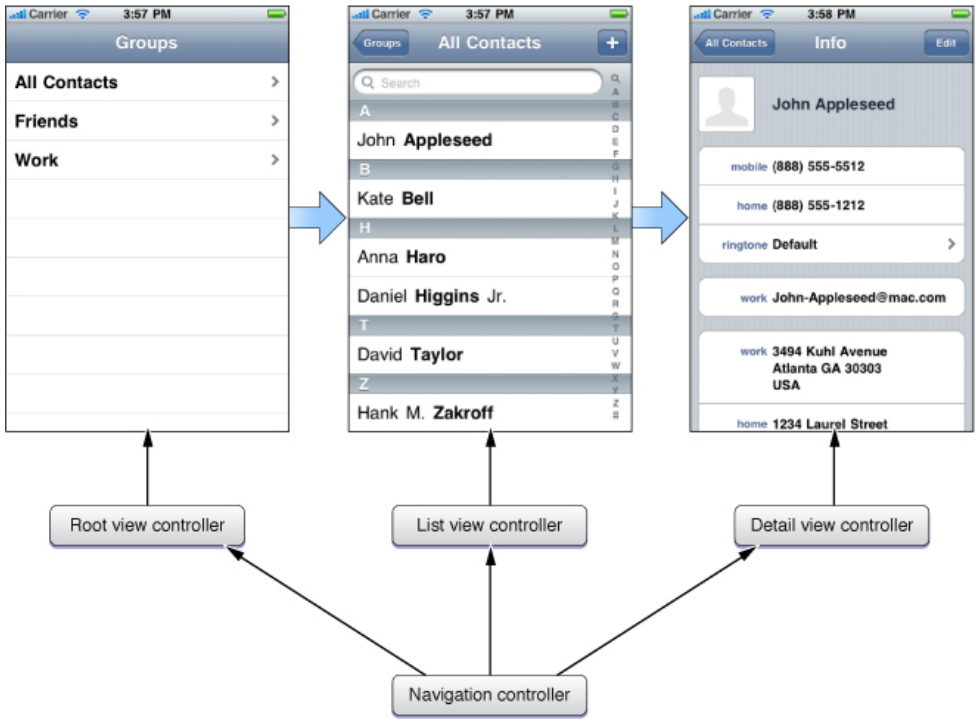

Figura9.3: Presentazione con Navigation controller.

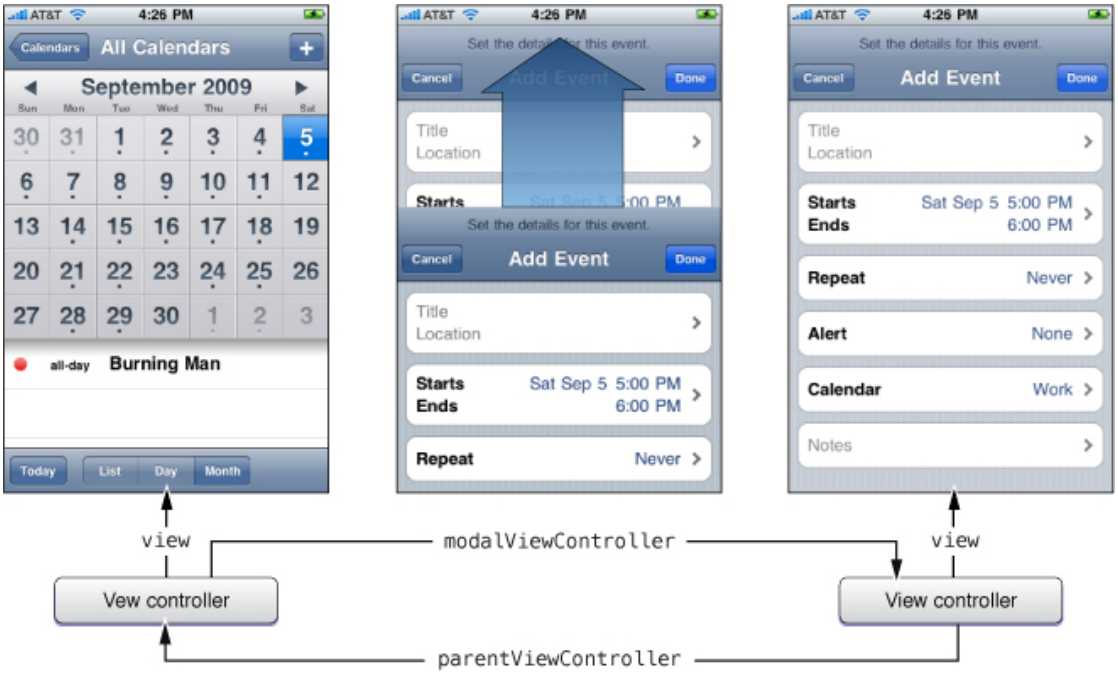

Figura 9.4: Presentazione con modalViewController.

Le specifiche di Apple[11] stabiliscono che la navigazione con *Navigation controller* deve essere utilizzata per presentare informazioni che sono organizzate gerarchicamente. Al contrario, tutte le schermate che pensate per raccogliere dei dati dall'utente, oppure per catturare la sua attenzione ad uno scopo specifico, devono essere presentate con un *modalViewController*. Una volta che tale scopo è stato esaurito, il controller *modale* viene dismesso permettendo all'utente di proseguire la navigazione attraverso l'applicazione.

Home screen

Lo scopo della schermata Home, dove viene emulata la *Springboard* di *iOS*, è quello di dare accesso all'utente a tutte le funzionalità integrate nell'applicazione, mettendo questi in condizione di scegliere quale task deve essere svolto.

[11]http://developer.apple.com/library/ios/#featuredarticles/viewcontrollerpgforiphoneos

La schermata principale di ogni funzionalità, compresa la visualizzazione degli avvisi dal sito UNIVE accessibili dalla *TickerBar*(estremità inferiore della schermata), deve essere presentata *modalmente*, al fine di evidenziare il cambio di contesto da una funzionalità all'altra.

Allo stesso modo la *status bar* di sistema deve apparire diversamente a seconda che ci si trovi nella *Home screen* oppure all'interno di una funzionalità. Nel primo caso ad essa deve essere associato lo stile *UIStatusBarStyleBlackOpaque*, nel secondo invece lo stile *UIStatusBarStyleDefault*.

Figura 9.5: Home screen di iCa'Foscari con TickerBar avvisi.

Figura 9.6: Visualizzazione di un avviso della TickerBar.

News

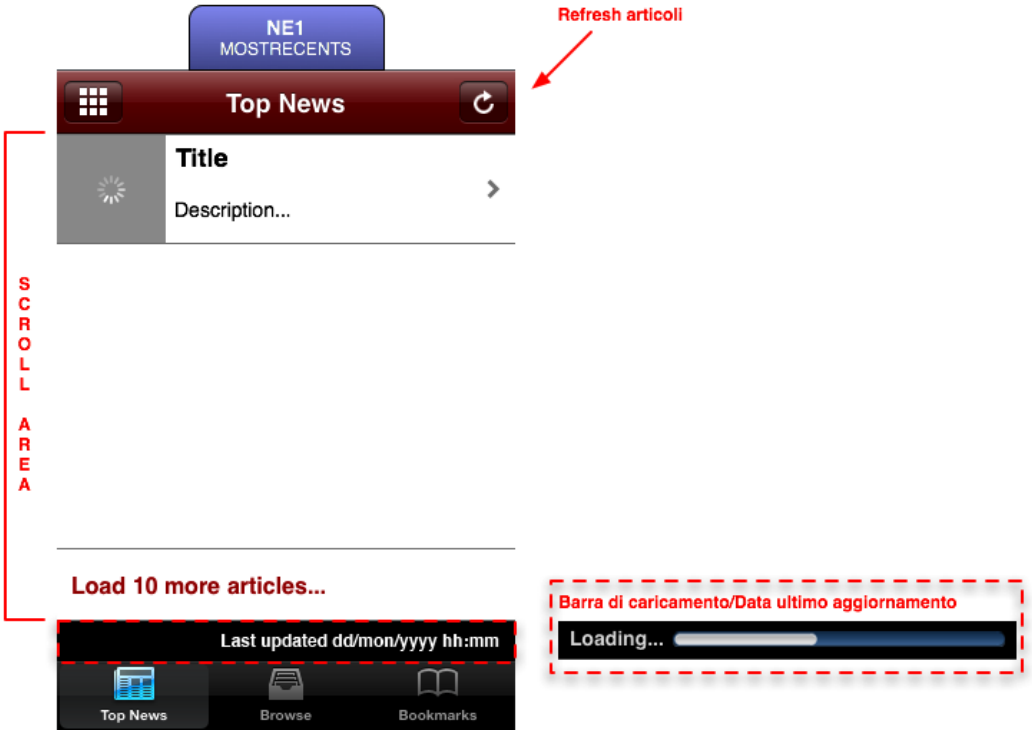

Figura 9.7: Funzionalità News, schermata NE1.

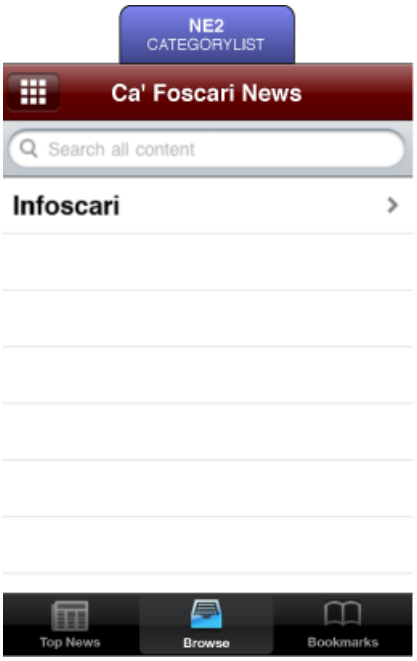

Figura 9.8: Funzionalità News, schermata NE2.

Figura 9.9: Funzionalità News, schermata NE3.

Figura 9.10: Funzionalità News, schermata NE4.

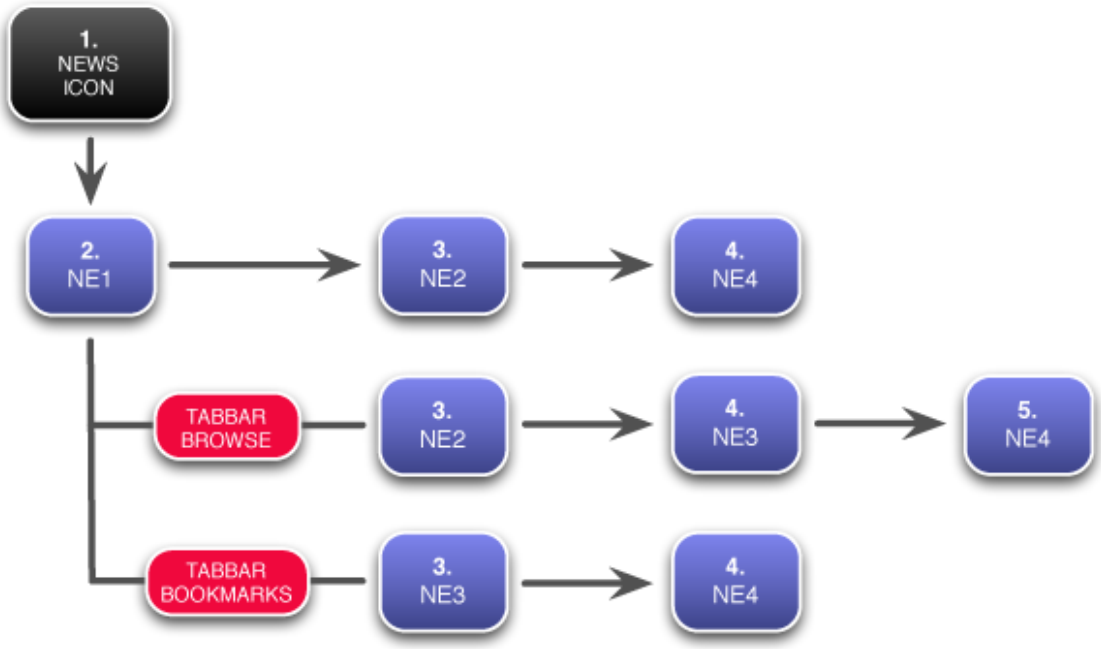

Figura 9.11: Funzionalità News, diagramma di collegamento.

Eventi

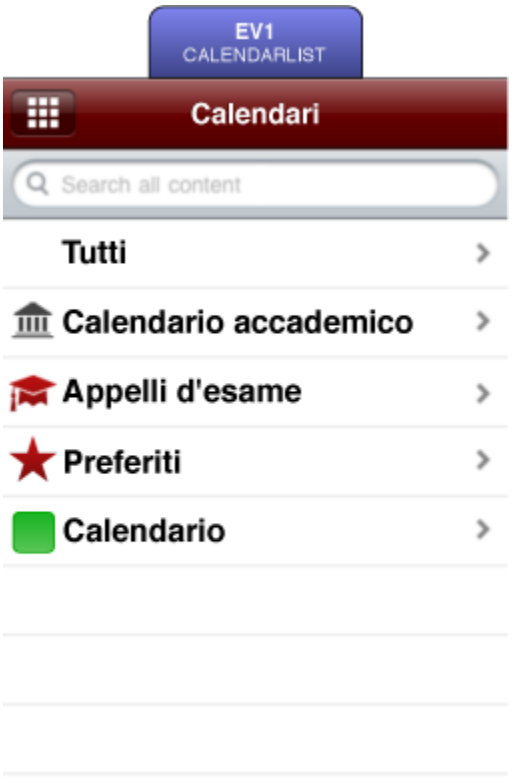

Figura 9.12: Funzionalità Eventi, schermata EV1.

Figura 9.13: Funzionalità Eventi, schermata EV2.

Figura 9.14: Funzionalità Eventi, schermata EV2-1.

Figura 9.15: Funzionalità Eventi, schermata EV2-2.

Figura 9.16: Funzionalità Eventi, schermata EV4.

Figura 9.17: Funzionalità Eventi, schermata EV4-1.

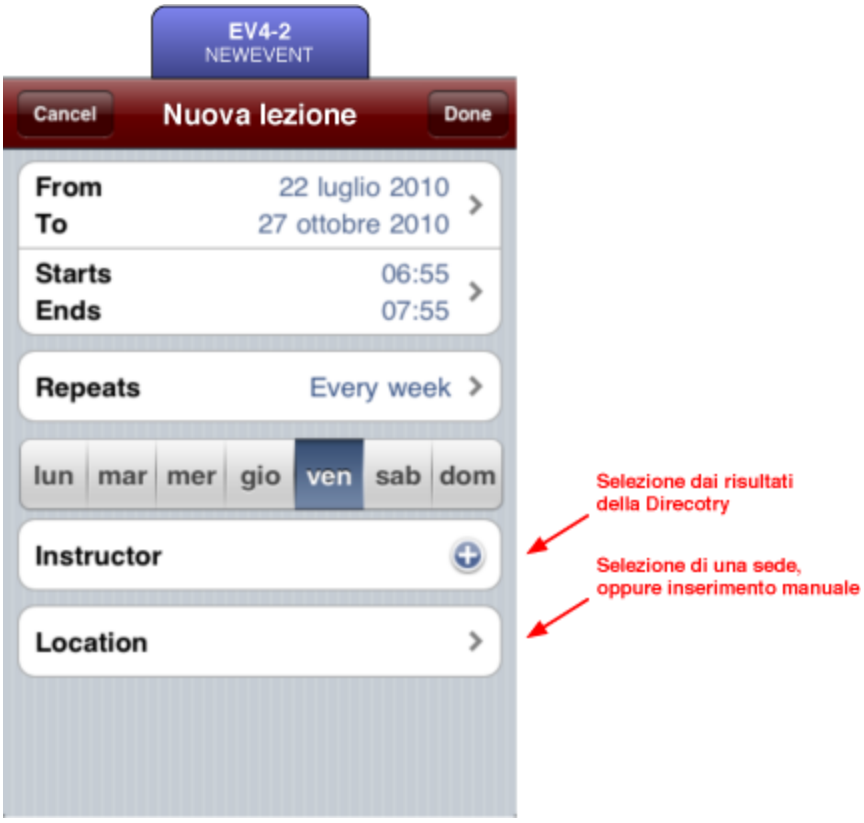

Figura 9.18: Funzionalità Eventi, schermata EV4-2.

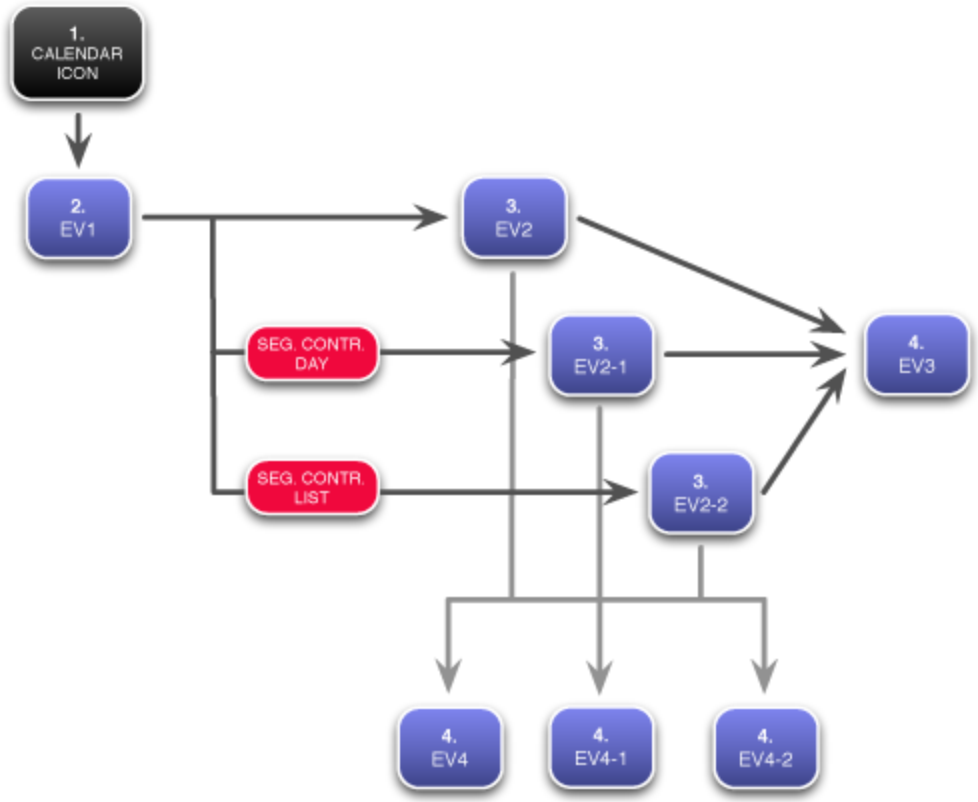

Figura 9.19: Funzionalità Eventi, diagramma di collegamento.

Directory

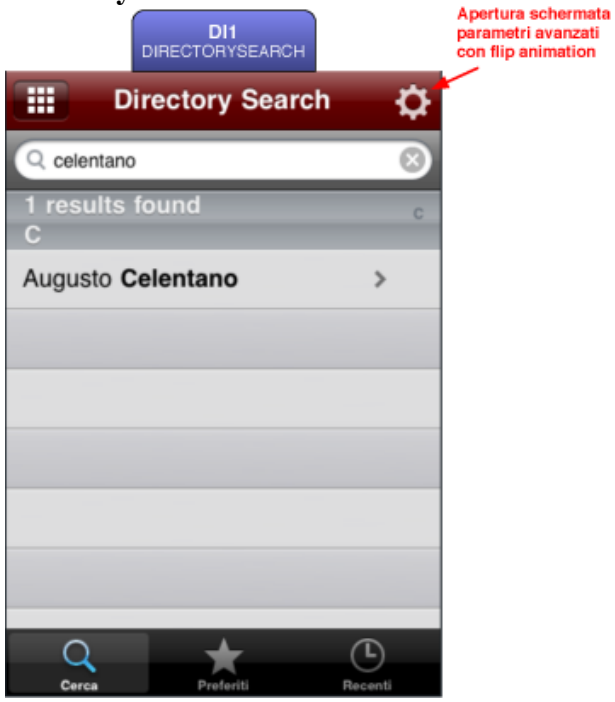

Figura 9.20: Funzionalità Directory, schermata DI1.

Figura 9.21: Funzionalità Directory, schermata DI2.

Figura 9.22: Funzionalità Directory, schermata DI3.

Figura 9.23: Funzionalità Directory, schermata DI4.

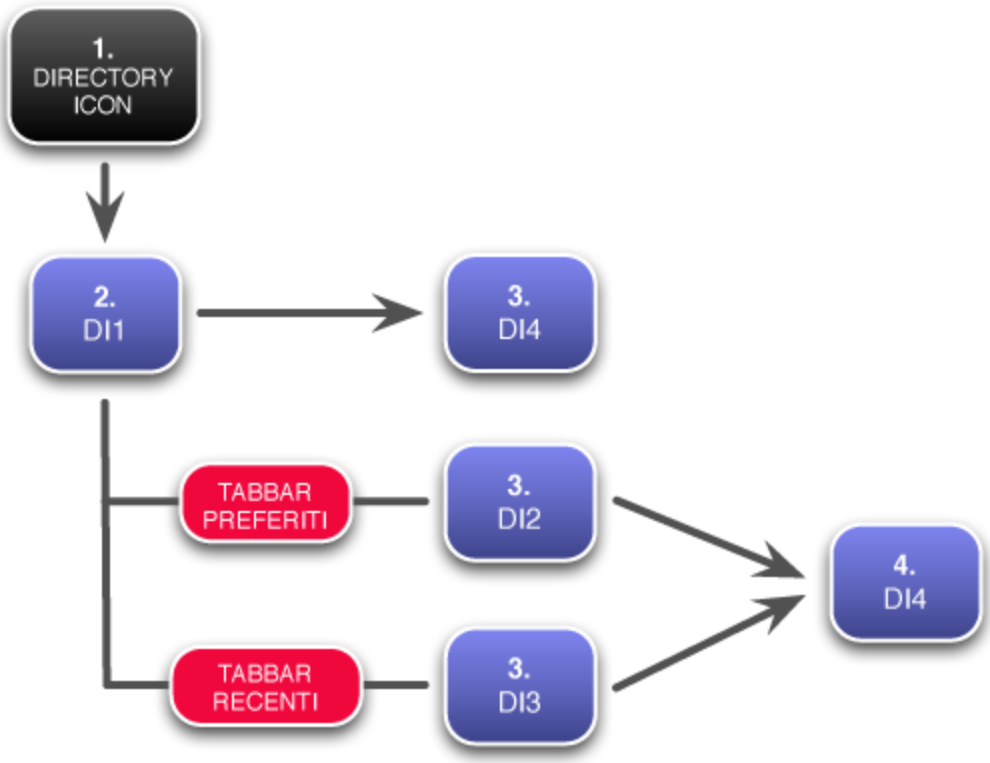

Figura 9.24: Funzionalità Directory, diagramma di collegamento.

Insegnamenti

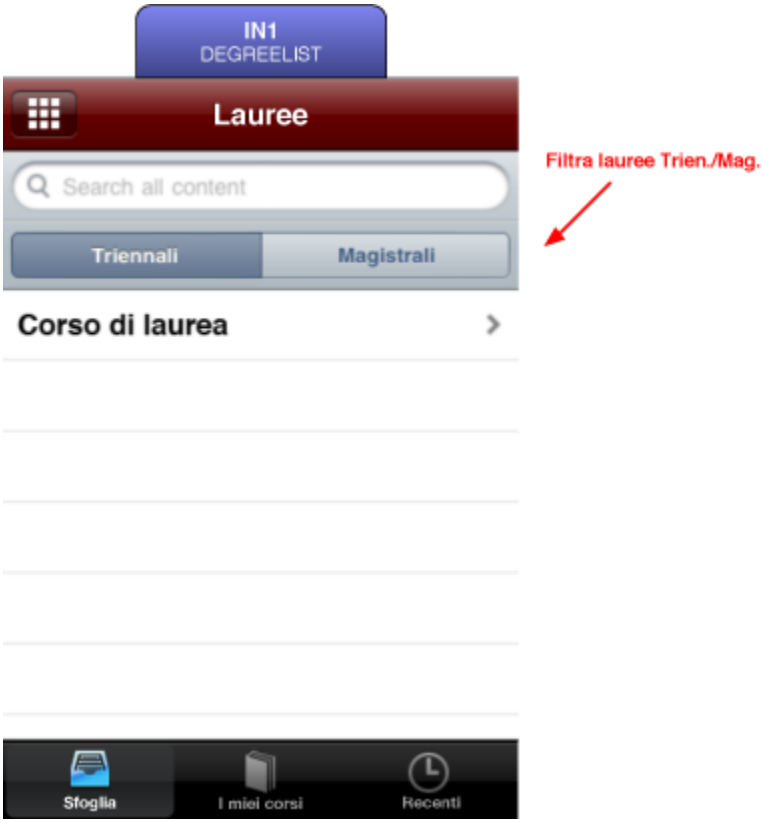

Figura 9.25: Funzionalità Insegnamenti, schermata IN1.

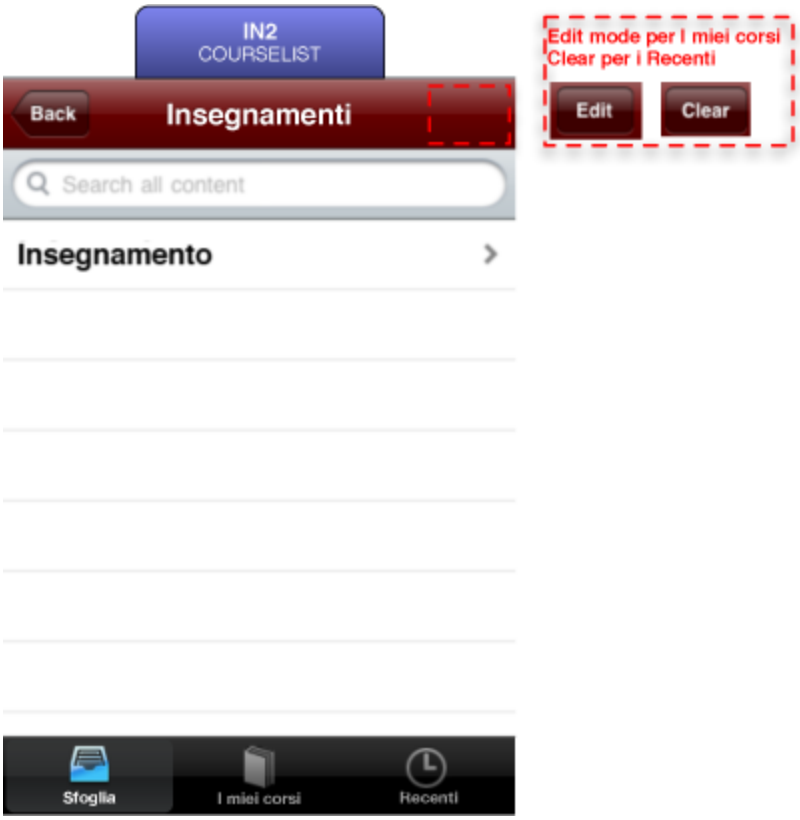

Figura 9.26: Funzionalità Insegnamenti, schermata IN2.

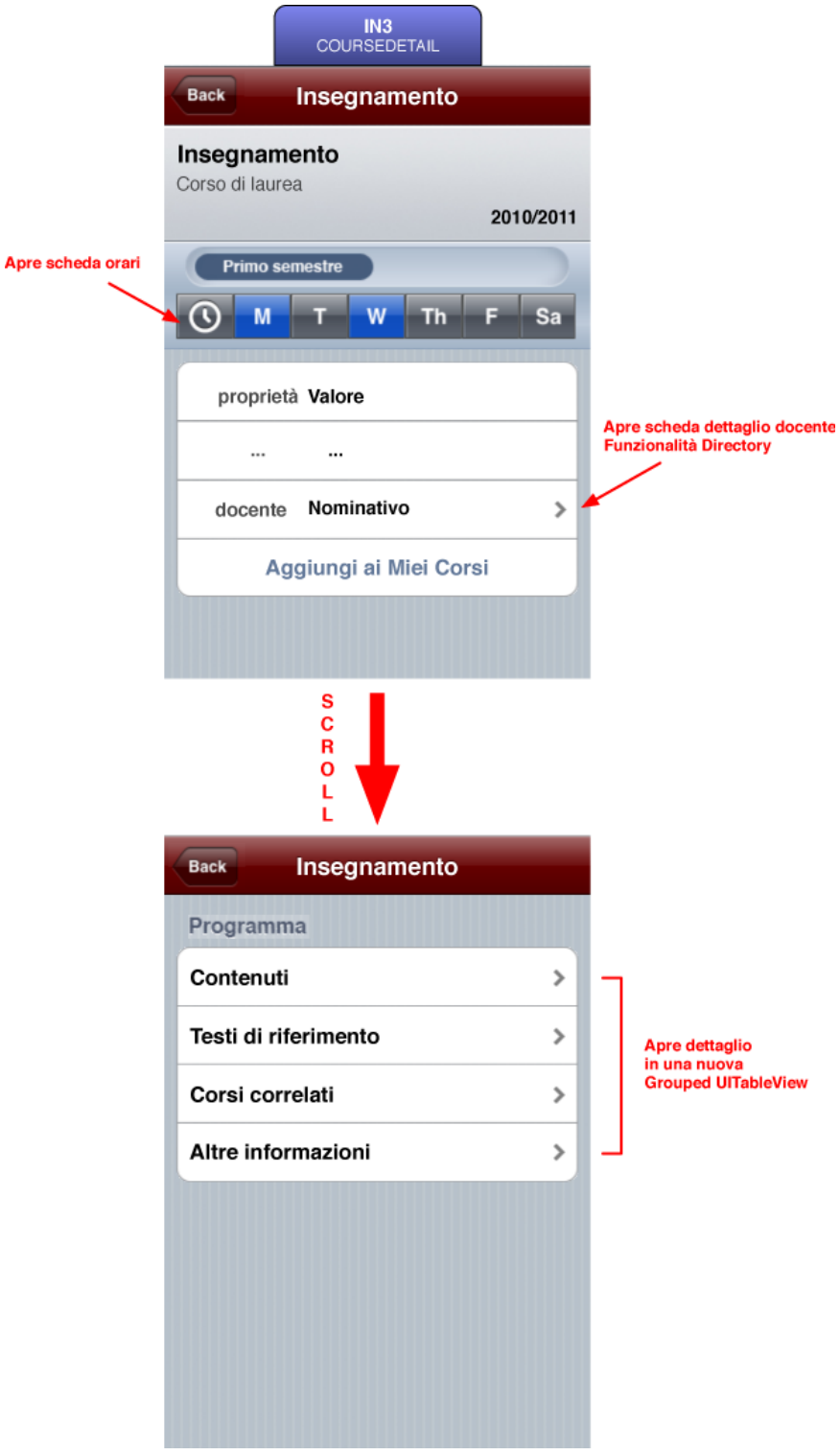

Figura 9.27: Funzionalità Insegnamenti, schermata IN3.

Figura 9.28: Funzionalità Insegnamenti, schermata IN3-1.

Figura 9.29: Funzionalità Insegnamenti, schermata IN4.

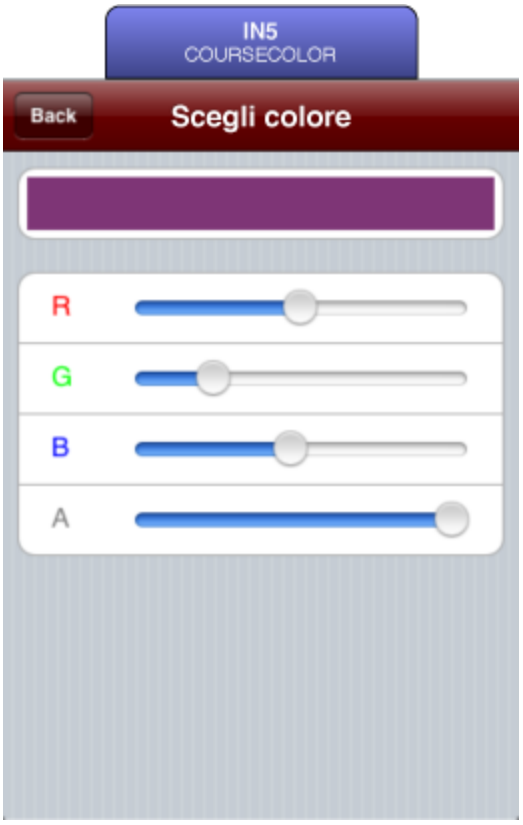

Figura 9.30: Funzionalità Insegnamenti, schermata IN5.

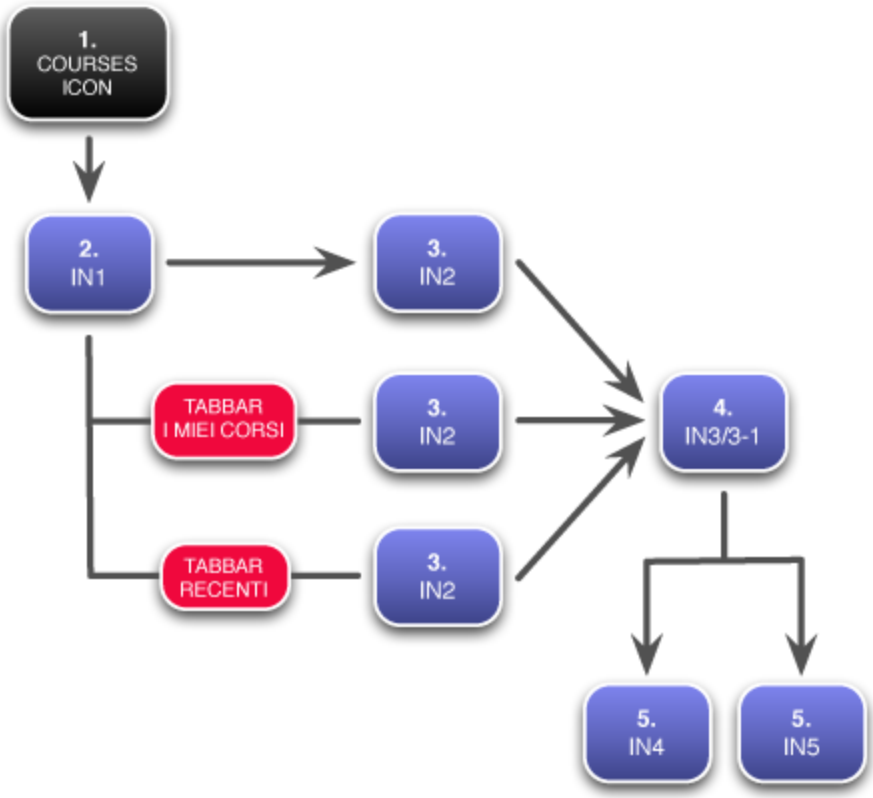

Figura 9.31: Funzionalità Insegnamenti, diagramma di collegamento.

Mappa

Figura 9.32: Funzionalità Mappa, schermata MA1.

Figura 9.33: Funzionalità Mappa, schermata MA2.

Figura 9.34: Funzionalità Mappa, schermata MA3/3-1.

Figura 9.35: Funzionalità Mappa, schermata MA4.

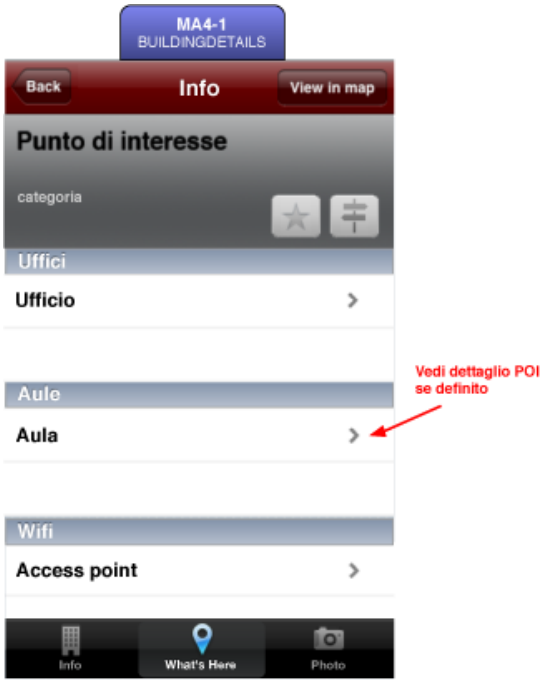

Figura 9.36: Funzionalità Mappa, schermata MA4-1.

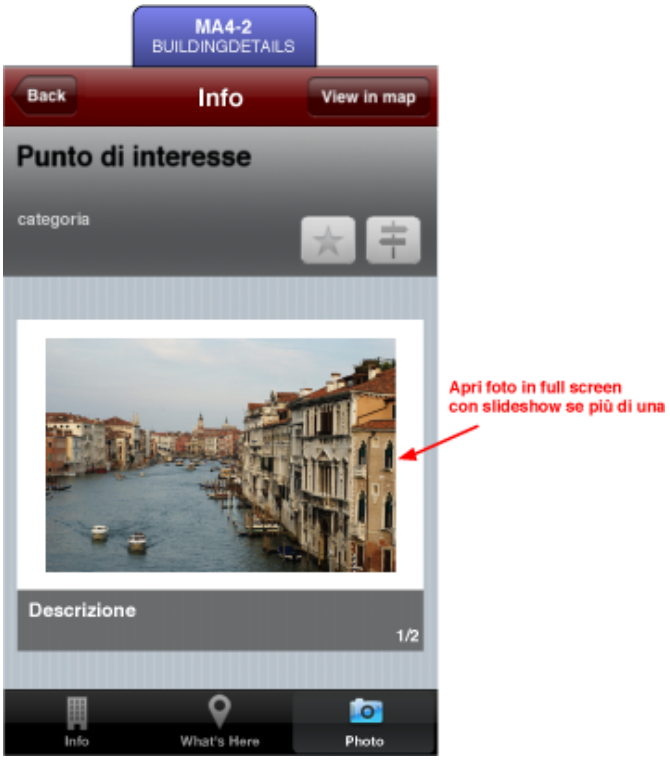

Figura 9.37: Funzionalità Mappa, schermata MA4-2.

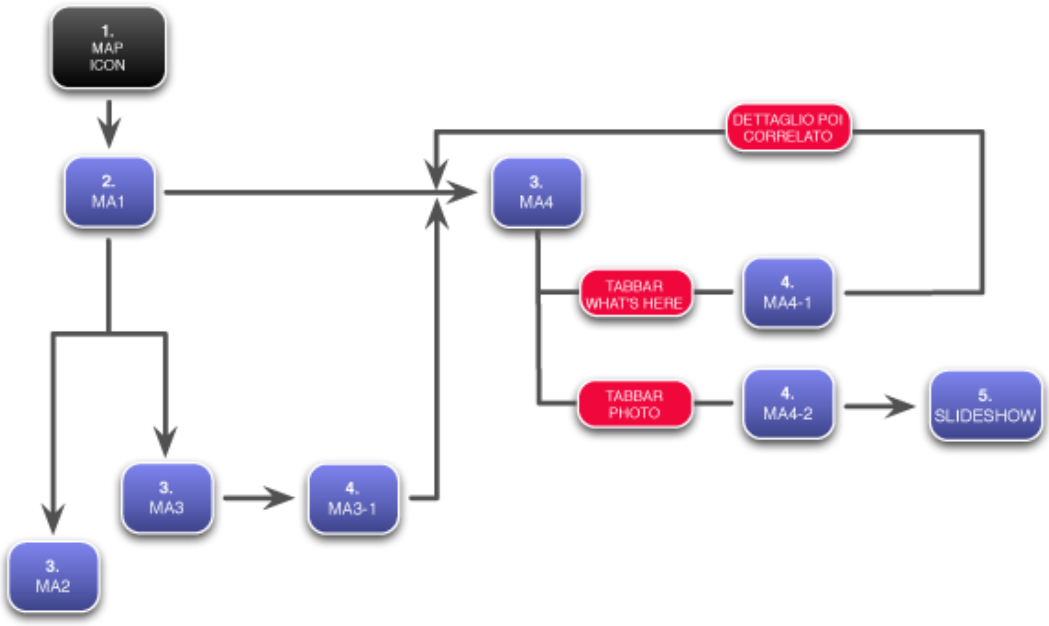

Figura 9.38: Funzionalità Mappa, diagramma di collegamento.

Media

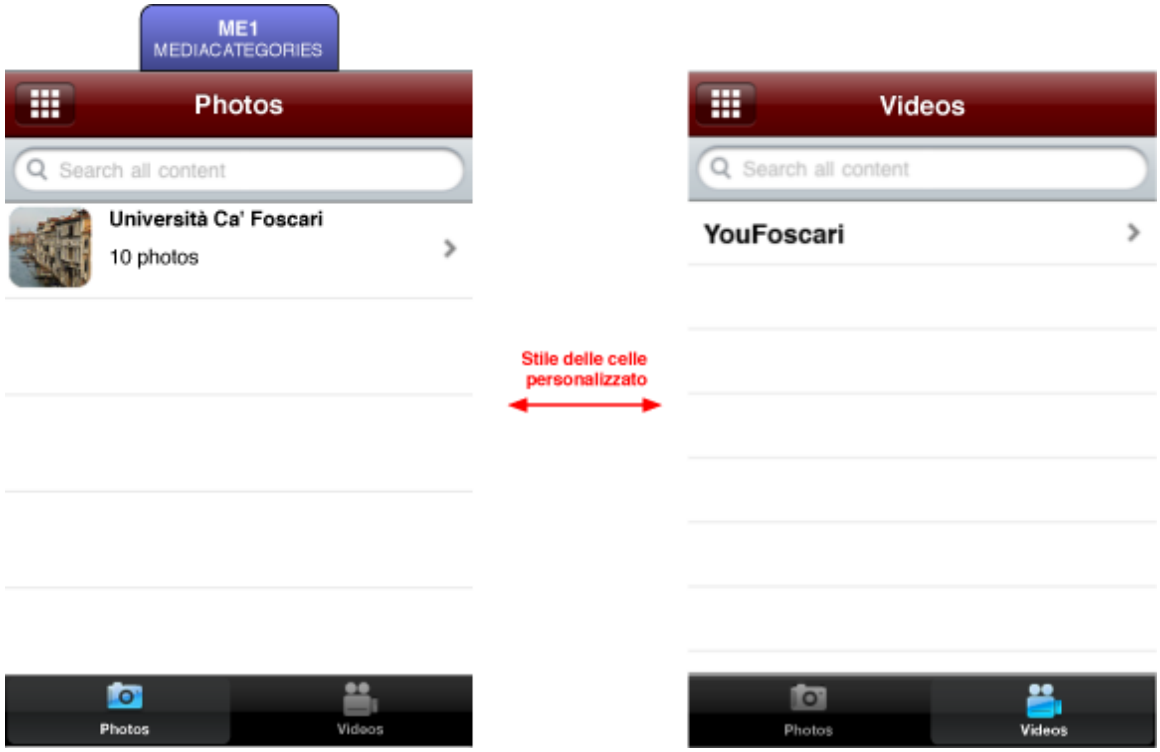

Figura 9.39: Funzionalità Media, schermata ME1.

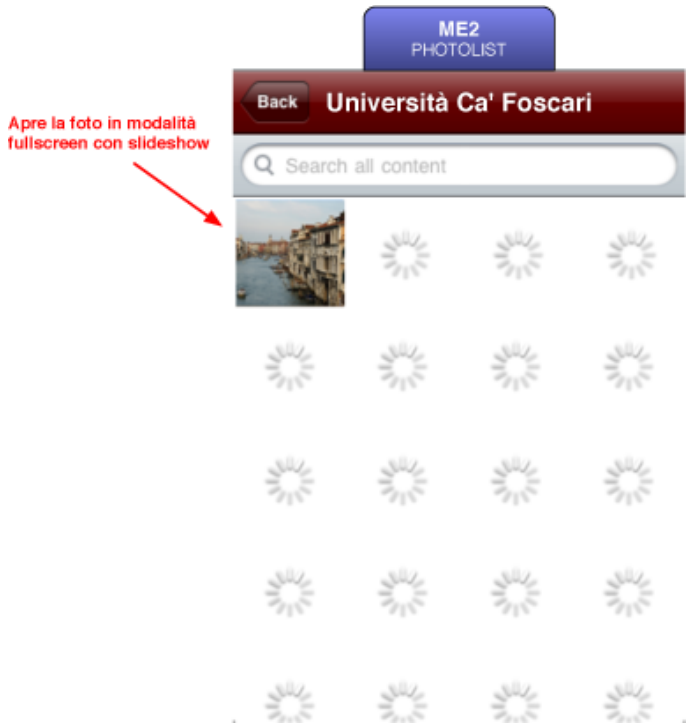

Figura 9.40: Funzionalità Media, schermata ME2.

Figura 9.41: Funzionalità Media, schermata ME3.

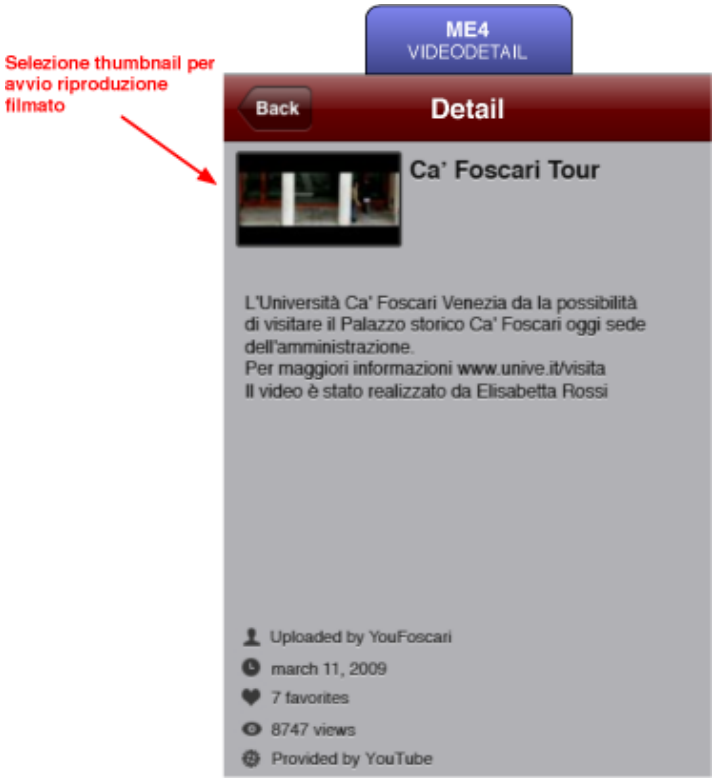

Figura 9.42: Funzionalità Media, schermata ME4.

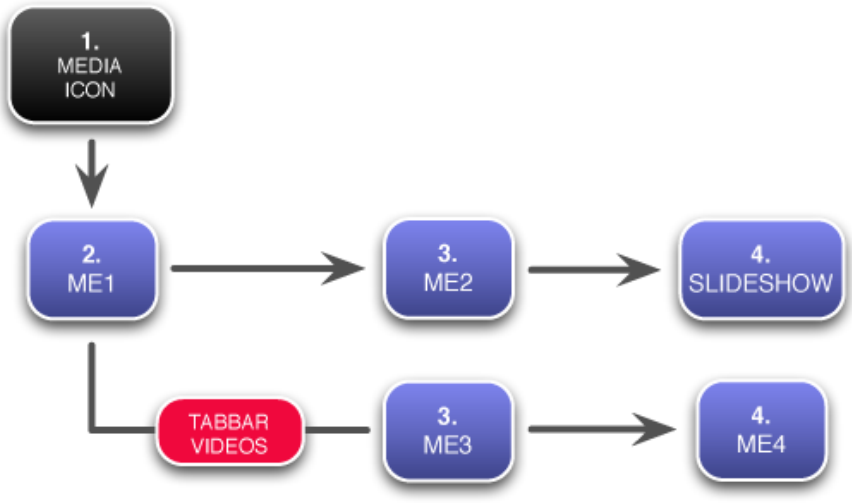

Figura 9.43: Funzionalità Media, diagramma di collegamento.

Emergenza

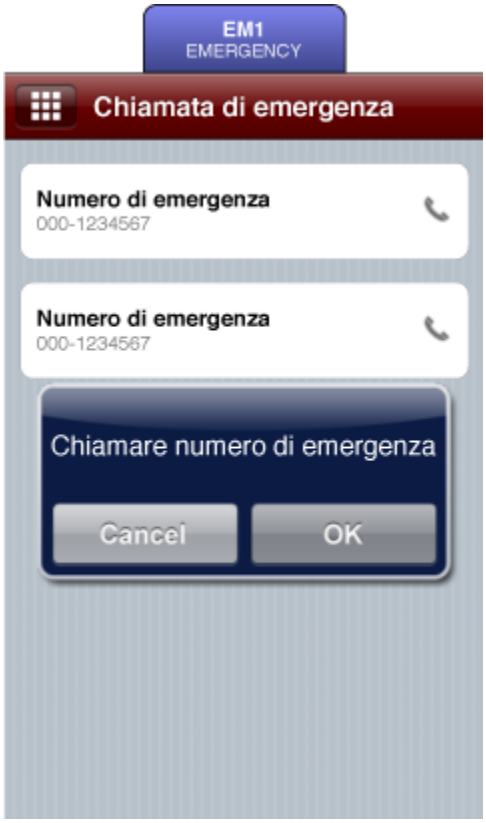

Figura 9.44: Funzionalità Emergenza, schermata EM1.

Social

Figura 9.45: Funzionalità Social, schermata SO1.

Figura 9.46: Funzionalità Social, schermata SO2.

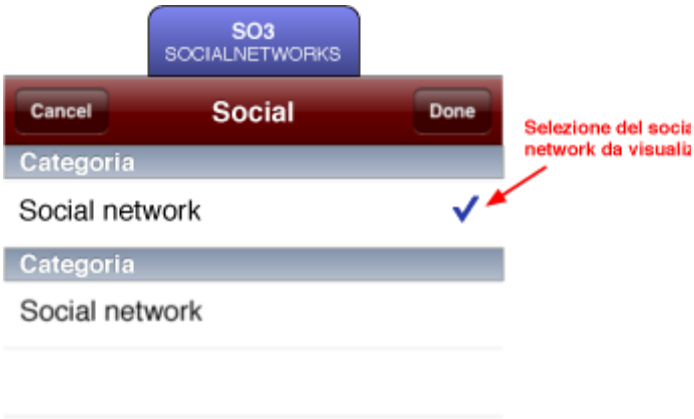

Figura 9.46: Funzionalità Social, schermata SO3.

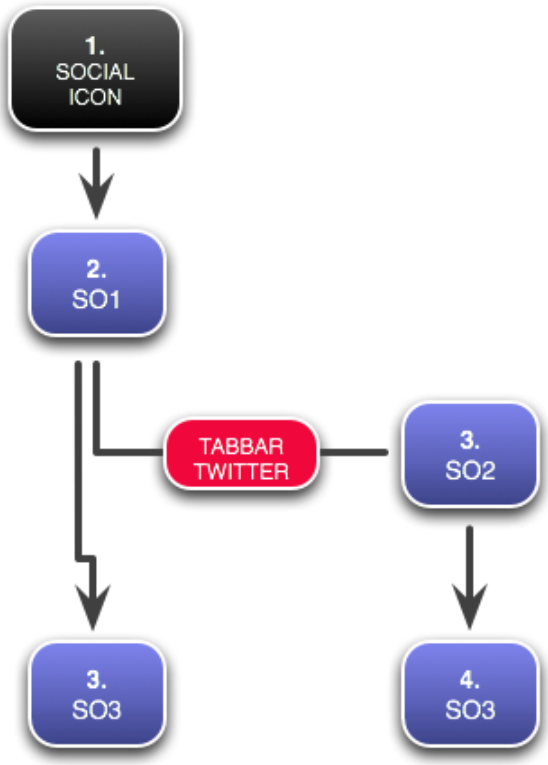

Figura 9.47: Funzionalità Social, diagramma di collegamento.

iTunesU

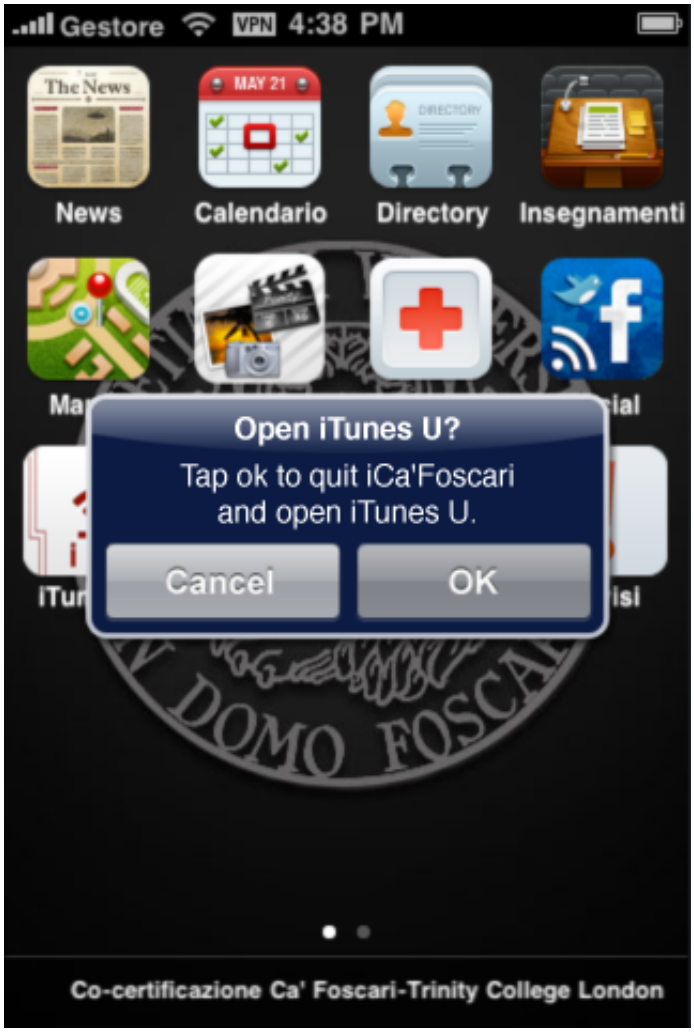

Figura 9.48: Funzionalità iTunesU.

Radio Ca' Foscari

Figura 9.49: Funzionalità RCF, schermata RA1.

Figura 9.50: Funzionalità RCF, schermata RA2.

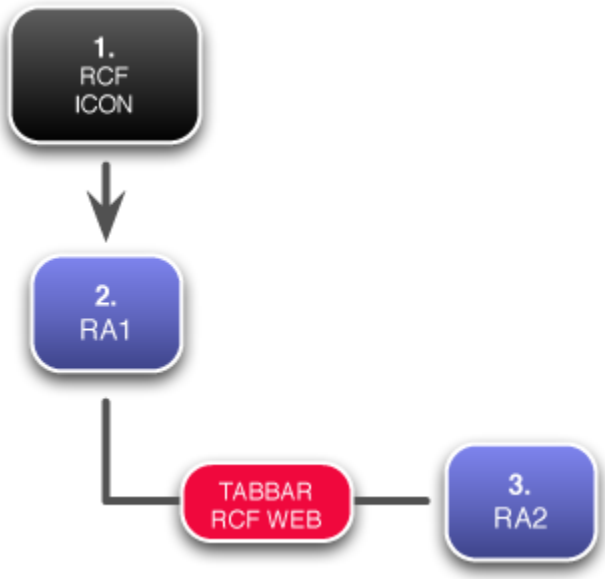

Figura 9.51: Funzionalità RCF, diagramma di collegamento.

Risorse

Figura 9.52: Funzionalità Media, schermata RI1.

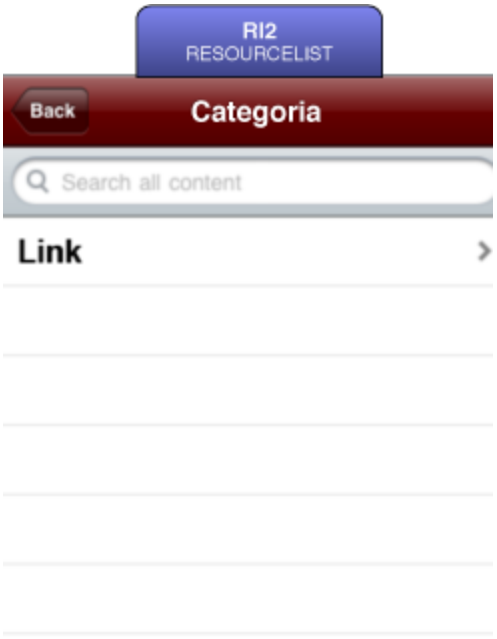

Figura 9.53: Funzionalità Media, schermata RI2.

Figura 9.54: Funzionalità Media, schermata RI3.

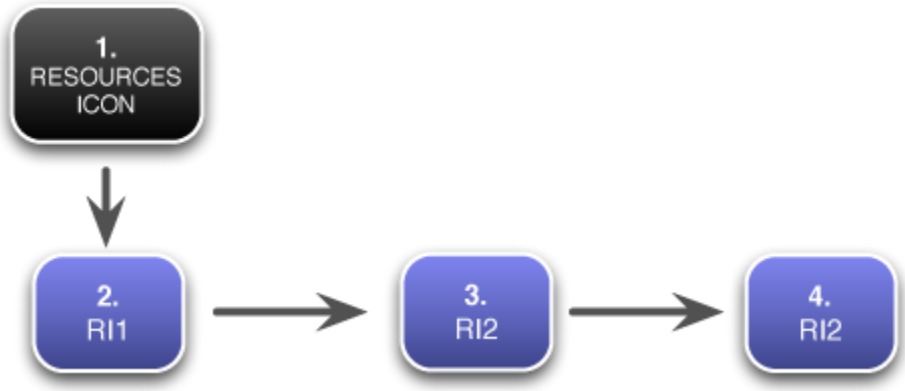

Figura 9.55: Funzionalità Media, diagramma di collegamento.

Avvisi

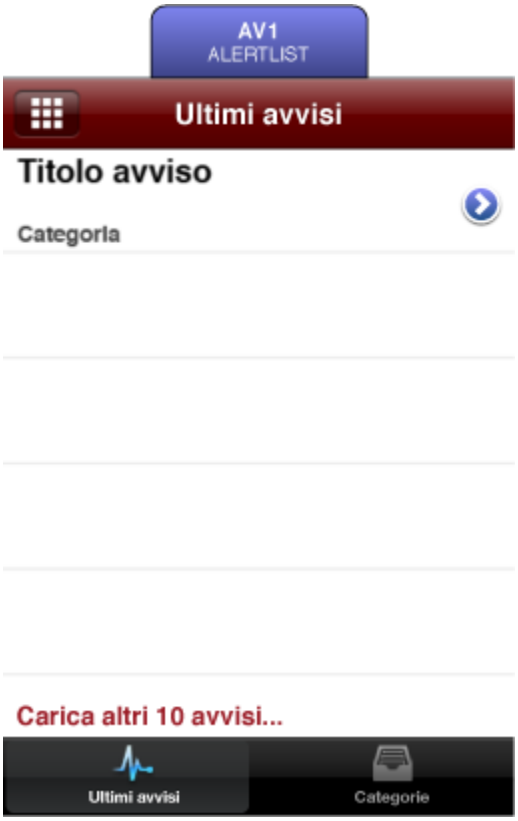

Figura 9.56: Funzionalità Avvisi, schermata AV1.

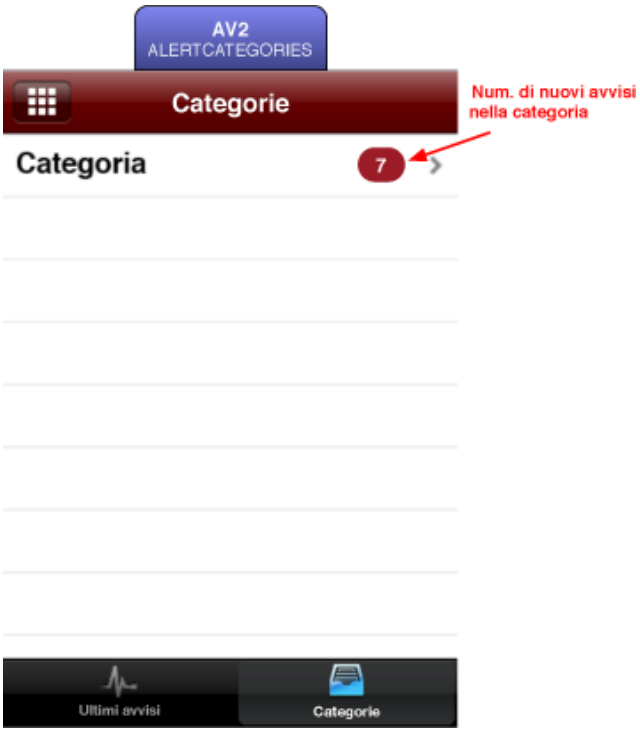

Figura 9.57: Funzionalità Avvisi, schermata AV2.

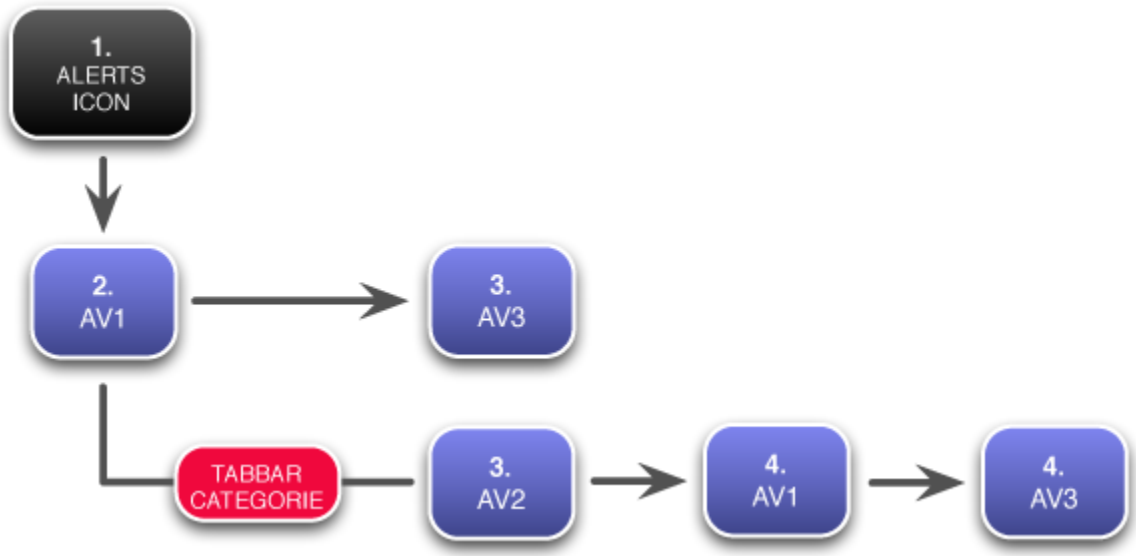

Figura 9.58: Funzionalità Avvisi, diagramma di collegamento.

CONCLUSIONI

In questo capitolo sono state definite le specifiche di progettazione dell'interfaccia utente di iCa'Foscari. Per di più sono stati presentati i *wireframe* di tutte le schermate previste nella progettazione discussa in questa parte dell'elaborato di tesi. Quest'ultimi non sono stati descritti in maniera distensiva, in quanto si desidera lasciare spazio ad eventuali revisioni future.

Infatti, l'interfaccia grafica definita nelle pagine precedenti non rappresenta necessariamente quella dell'applicazione finale. Bensì costituisce un'ossatura, coerente con i requisiti funzionali e adatta ai contenuti dell'applicazione, sulla quale applicare uno stile originale, approvato dagli organi amministrativi dell'Università.

Infine, sono state introdotte delle soluzioni grafiche disegnate specificamente per assolvere ai casi d'uso di alcune funzionalità e offrire allo stesso tempo un livello di usabilità elevato. Tali accorgimenti non sono vincolanti e possono essere sostituiti a vantaggio di soluzioni ritenute migliori.

Capitolo 10: Ambiente e infrastruttura di scambio dati

Uno dei requisiti di iCa'Foscari è quello di fornire contenuti sempre validi e aggiornati. Affinché ciò sia vero è necessario predisporre di un'infrastruttura informativa sulla quale l'applicazione potrà affidarsi per reperire le informazioni desiderate.

In questo capitolo viene descritta sommariamente la struttura di tale sistema, limitandosi a spiegare come avviene lo scambio dei dati e come questa si andrebbe a integrare nell'attuale Sistema Informativo dell'Università. Infatti, non rientra nell'ambito di questa tesi la progettazione di tale infrastruttura, la quale viene proposta invece come possibile *future work* di questo progetto.

DATA INTERCHANGE: XML VS JSON

Le comunicazioni tra l'applicazione e la sorgente dati esterna avvengono principalmente attraverso una connessione ai punti di accesso *WiFi* della rete *VPN* di Ca' Foscari. Tuttavia vi possono essere situazioni in cui l'utente stia utilizzando una connessione a pagamento su rete *3G* o *GPRS*. Per di più, gli attuali dispositivi *iOS* non dispongono di risorse e prestazioni comparabili a quelle di un normale computer. Per questi motivi è necessario scegliere un formato di trasferimento dati capace di ottimizzare l'utilizzo della banda e delle risorse a disposizione.

In questo scenario di utilizzo fanno la loro comparsa due formati di serializzazione e scambio dati molti diffusi e tra loro molto simili nelle caratteristiche, ma del tutto diversi nelle prestazioni. Questi sono *XML* e *JSON*[12].

[12]http://www.json.org/xml.html

XML come linguaggio di rappresentazione ha il vantaggio di essere basato su testo e di essere indipendente dalla posizione, permettendo di ottenere un livello di *application-indipendence* superiore ad ogni altro linguaggio di *data-interchange*. Non per niente *XML* è uno standard *W3C*. Malgrado ciò, *XML* non è in realtà molto adatto allo scambio dati, in quanto i documenti trasferiti sono pesanti, e non è direttamente compatibile con il modello dati della maggior parte delle applicazioni.

JSON al contrario ha gli stessi vantaggi di *XML*, in termini di interoperabilità e leggibilità, ma è molto più adatto alle operazioni di serializzazione e scambio dati; mentre il secondo è più orientato allo scambio di documenti. Le caratteristiche che hanno fatto di *JSON* il linguaggio più comunemente utilizzato per il *data-interchange* sono le seguenti:

- *JSON* è *data-oriented*, il che vuol dire che è più facile da mappare verso i sistemi orientati agli oggetti rispetto a *XML*, che è invece *document-oriented*;
- *XML* separa la presentazione dei dati dalla loro struttura, richiedendo quindi di tradurre la struttura dei dati in una struttura documentale. Questa operazione di *mapping* può però risultare complicata. Le strutture di *JSON* invece sono basate su *array* e *record*. Cioè proprio ciò di cui sono fatti i dati, semplificandone ulteriormente l'elaborazione;
- *JSON* ha una notazione più semplice di *XML*, pertanto richiede software meno specializzato. Inoltre, si trovano *parser* e *generator* di codice *JSON* già implementati nella maggior parte dei linguaggi di programmazione;
- *JSON* è un formato di tipo *lightweight* ottimizzato per i dati. Pertanto non è adatto, al contrario di *XML*, al trasferimento di suoni, immagini o grandi *payload* in generale. Inoltre, l'invio di programmi eseguibili in un sistema di *data-interchange* può comportare dei seri problemi di sicurezza.

In sintesi *JSON* è molto più conveniente di *XML* per lo scambio dati per via della sua grammatica più semplice e ristretta. Ciò vuol dire, infatti, che può essere mappato direttamente nelle strutture dati utilizzate nei linguaggi di programmazione più moderni, semplificando la logica di *parsing* e impegnando meno banda e risorse. Pertanto la progettazione di iCa'Foscari prevede già da subito l'utilizzo di *JSON* come unico formato dei dati provenienti dalla sorgente.

SINTESI DELL'INFRASTRUTTURA DI SERVIZIO

L'illustrazione riportata di seguito offre una panoramica dell'infrastruttura di servizio per iCa'Foscari. Quanto introdotto non rappresenta nulla di completo o definitivo, ma si propone di fornire un'idea di come andrebbe collocato il nuovo servizio e di quante e quali risorse sarebbero necessarie per mettere in opera l'applicazione così come progettata in questo lavoro di tesi. Possibili *future work* ed estensioni riguardanti questo sistema sono discussi nell'ultima parte di questo elaborato.

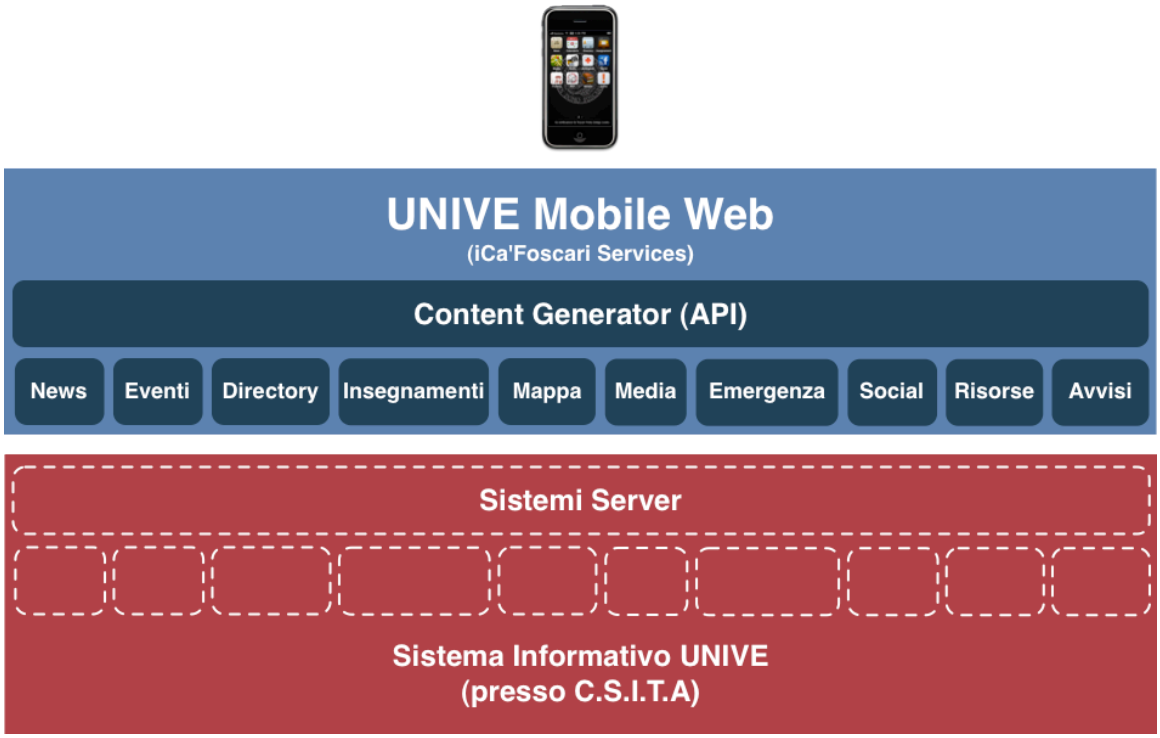

Figura 10.1: Infrastruttura Servizi iCa'Foscari.

Qui di seguito sono delineati i componenti dell'infrastruttura, dal basso verso l'alto:

- **Sistema Informativo UNIVE**: rappresenta il Sistema Informativo attualmente in uso dall'Università Ca' Foscari e gestito dal Centro Servizi Informatici e di Telecomunicazioni di Ateneo (C.S.I.T.A.);
- **Sistemi Server**: corrisponde ad una ripartizione dei servizi offerti dall'attuale Sistema Informativo. Nell'insieme costituisce un nuovo sistema di gestione ed elaborazione dei contenuti, capace di consentire l'adeguamento di tali servizi alla loro fruizione da parte di iCa'Foscari, attraverso l'integrazione di un nuovo strato di comunicazione e una nuova *API* di sistema;

- **Content Generator**: costituisce il fulcro di tutta l'infrastruttura. La sua responsabilità è quella di elaborare tutte le richieste in entrata dai dispositivi su cui è installato iCa'Foscari, inviando in risposta la serializzazione in formato *JSON* dei dati ottenuti dal corrispondente sistema di supporto integrato nel Sistema Informativo. Un'*API* specificamente progettata fornirà l'interfacciamento sia con iCa'Foscari, che con il Sistema Informativo.

L'illustrazione pone in evidenza l'introduzione di un nuovo sistema per la generazione dei contenuti (Content Generator). Questo componente, infatti, è necessario non solo per gestire l'interfacciamento con l'applicazione, ma soprattutto per semplificare l'integrazione di iCa'Foscari con il Sistema Informativo, e garantire il funzionamento di tutto il sistema anche nel caso di eventuali modifiche apportate ad ogni altro suo componente, compresa l'applicazione stessa.

Nella struttura proposta si nota, inoltre, la scomposizione dei servizi offerti dal Sistema Informativo e il loro interfacciamento con il Content Generator. Tale scomposizione, oltre ad indicare la coerenza e l'integrazione tra i due livelli dell'infrastruttura, vedrebbe l'esistenza di un sistema per ogni servizio offerto. Tuttavia è del tutto ammissibile che uno o più servizi risiedano nel medesimo server. In ogni caso è preferibile organizzare i servizi in sistemi modulari, al fine di rendere l'infrastruttura complessivamente più affidabile nel caso di malfunzionamenti, e in generale più efficiente, scalabile e facile da mantenere.

Inoltre, il numero di sistemi server e di servizi indicato nella figura non rispecchia realmente la situazione attuale del Sistema Informativo. Sarà così parte di un lavoro futuro, progettare dettagliatamente un'infrastruttura che tenga conto di questi presupposti,

capace soprattutto di mediare tra le risorse hardware attualmente disponibili e quelle richiesta dalla quantità di dispositivi su cui verrà distribuito iCa'Foscari.

PROCEDIMENTO DI SCAMBIO DEI DATI

Lo scambio dei dati tra l'applicazione e l'infrastruttura UNIVE Mobile Web avviene in tre fasi distinte:

1. L'applicazione invia una richiesta all'*host address* del Content Generator formulando una chiamata all'*API* di sistema;
2. Il Content Generator interpreta la chiamata ed elabora i risultati della query prelevando i dati richiesti dal server responsabile per il servizio invocato;
3. Ottenuti i dati da inviare all'applicazione, il Content Generator si occupa di serializzarli e restituirli in formato *JSON* all'indirizzo *IP* del dispositivo da cui iCa'Foscari aveva effettuato la richiesta.

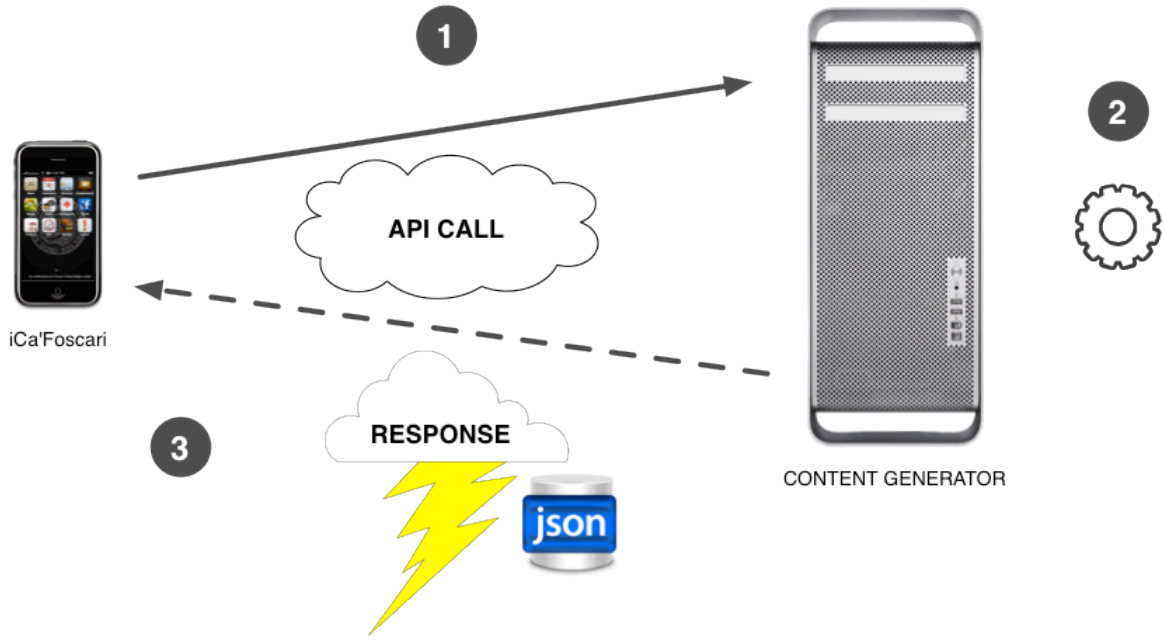

Figura 10.2: Procedimento di scambio dei dati.

PROFILO DI CONFIGURAZIONE DEI DISPOSITIVI

L'*iPhone Configuration Utility* permette di generare dei profili di configurazione da installare sui dispositivi *iOS*[13]. Questi per poter essere distribuiti, devono essere ospitati su un server, come ad esempio il server *hotspot.unive.it*; in modo tale che gli utenti di iCa'Foscari possano scaricarli, da una pagina web o da un messaggio e-mail, ed installarli sul proprio dispositivo. Inoltre, i profili possono essere firmati da una *Certification Authority* per dimostrarne l'originalità.

La procedura permette di semplificare, tra le altre, la configurazione delle impostazioni *VPN* necessarie per la connessione agli *hotspot* della rete *WiFi* di Ateneo e la configurazione dell'applicazione Mail per la ricezione della posta dalla propria casella sul server *mail.stud.unive.it*. Per di più, possono essere distribuiti dei profili anche per la configurazione di servizi *LDAP* o *CalDAV* eventualmente utilizzati da iCa'Foscari, oppure per la distribuzione di chiavi e credenziali per l'accesso sicuro ai server dell'Università.

La schermata seguente mostra un esempio di profilo per la configurazione delle impostazioni *VPN* Ca' Foscari.

[13]http://developer.apple.com/library/ios/featuredarticles/iphoneconfigurationprofileref/introduction/

Figura 10.3: Esempio di iPhone Configuration Profile.

CONCLUSIONI

In queste pagine è stato stabilito il formato di scambio dati con l'applicazione ritenuto più efficiente. Inoltre, è stata introdotta una nuova architettura che si colloca tra l'applicazione e il Sistema Informativo attualmente in uso dall'Ateneo. Tale architettura non si limita alla fornitura dei servizi solamente alla piattaforma *iOS* tramite iCa'Foscari, ma, come vedremo nell'ultima parte di questa tesi, potrà costituire un'infrastruttura di supporto a tutti i dispositivi mobili esistenti.

Capitolo 11: Implementazione del sottosistema Directory

In questa sezione della seconda parte dell'elaborato viene presentata in dettaglio l'implementazione del requisito funzionale Directory, a titolo di esempio per lo sviluppo futuro degli altri sottosistemi. In particolare, sono riportati i listati degli *header* delle principali classi coinvolte e i diagrammi di sequenza, che descrivono l'interazione degli oggetti del sottosistema nello svolgimento di alcuni scenari di maggiore importanza. Infine, sono dettate alcune linee guida per l'ottimizzazione del codice in generale.

HEADER DELLE CLASSI

I listati riportati nei paragrafi di questa sottosezione rappresentano gli *header* dei modelli e dei controllori definiti nello stesso diagramma architetturale introdotto al capitolo sulla progettazione di sistema.

Gli *header* sono scritti in linguaggio *Objective-C*[14]. In sostituzione dei sorgenti dell'implementazione, per ogni classe segue una tabella che descrive sinteticamente l'utilità dei campi e dei metodi dichiarati.

UNIVEDirectorySearch

Header

```
#import <Foundation/Foundation.h>
#import <AddressBook/AddressBook.h>
#import <AddressBookUI/AddressBookUI.h>

@protocol UNIVEDirectorySearchDelegate;

@interface UNIVEDirectorySearch : NSObject {
    NSDictionary *indexedResults;
    NSArray *keys;
    id<UNIVEDirectorySearchDelegate> delegate;
```

[14]http://developer.apple.com/library/mac/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/

```
    NSUInteger resultsNumber;
    NSThread *aThread;
}

@property(nonatomic, retain) NSDictionary *indexedResults;
@property(nonatomic, retain) NSArray *keys;
@property(nonatomic, assign) NSUInteger resultsNumber;
@property(nonatomic, assign) id<UNIVEDirectorySearchDelegate>
delegate;

- (id)initWithReachability;
- (void)searchWithString:(NSString *)searchQuery;
+ (ABRecordRef)createABPerson:(NSDictionary *)dict;
- (NSString *)getPrivateKey;
- (BOOL)directoryIsReachable;
- (void)reachabilityChanged;

@end

@protocol UNIVEDirectorySearchDelegate

- (void)newDirectoryDataAvailable:(UNIVEDirectorySearch
*)aSearchEngine;
- (void)directorySearchErrorOccured:(NSString *)message;
- (void)testReachability;
@end
```

Listato 11.1: UNIVEDirectorySearch.h

Scheda descrittiva

Nome: UNIVEDirectorySearch Tipo: Modello	
Campi	
Nome	Descrizione
indexedResults	Dizionario indicizzato in ordine alfabetico per iniziale del cognome dei risultati di una ricerca nella directory.
keys	Array delle lettere utilizzate per l'indicizzazione del campo precedente.

delegate	Riferimento all'oggetto delegato definito dal protocollo UNIVEDirectorySearchDelegate.
resultsNumber	Numero di risultati della ricerca.
aThread	Thread per il download e il parsing del codice JSON ottenuto dalla directory.

Metodi	
Nome	Descrizione
initWithReachability	Inizializza l'oggetto e definisce l'host sui cui testare la connettività.
searchWithString	Riceve la query inserita dall'utente e avvia un nuovo thread per inviarla alla directory.
createABPerson	Metodo statico per la creazione di un profilo compatibile col framework AddressBook sui dati ottenuti dalla directory.
getPrivateKey	Carica un file Property List contenente la chiave necessaria per effettuare delle query nella directory LDAP.
directoryIsReachable	Informa i controller se l'host dove si trova la directory è raggiungibile.
reachabilityChanged	Rileva e informa il delegato sui cambiamenti nella raggiungibilità dell'host.

Tabella 11.1: Interfaccia UNIVEDirectorySearch.

Protocolli e oggetti delegati

L'interfaccia definisce un protocollo a cui l'oggetto controller, che coordina l'interazione con l'utente, si deve conformare implementando i metodi stabiliti.

Nome: UNIVEDirectorySearchDelegate Tipo: Protocollo	
Metodi	
Nome	Descrizione
newDirectoryDataAvailable	Tramite il pattern Target-Action, viene chiamato quando i risultati di una ricerca sono disponibili e si occupa di caricarli nella tabella.
directorySearchErrorOccured	Con lo stesso meccanismo del precedente, presenta all'utente un Alert per notificare un errore.
testReachability	Richiede lo stato di raggiungibilità della directory.

Tabella 11.2: Protocollo UNIVEDirectorySearchDelegate.

DirectoryRoot

Header

```
#import <UIKit/UIKit.h>
#import "DirectoryRecents.h"
#import "DirectoryFavorites.h"
#import "PersonInfo.h"
#import <AddressBook/AddressBook.h>
#import <AddressBookUI/AddressBookUI.h>

#define kRecentsNumber 10

@interface DirectoryRoot : UIViewController <UITabBarDelegate,
UIAlertViewDelegate, DirectoryRecentsDelegate,
DirectoryFavoritesDelegate, NSFetchedResultsControllerDelegate,
ABUnknownPersonViewControllerDelegate> {

    DirectoryRecents *directoryRecents;
    DirectoryFavorites *directoryFavorites;
    PersonInfo *personInfoViewController;
    IBOutlet UITabBarController *theTabBarController;
    NSMutableArray *recents;
    NSMutableArray *favorites;
```

```
    @private
     NSFetchedResultsController *_fetchedResultsController;
     NSManagedObjectContext *managedObjectContext;
}

@property (nonatomic, retain) NSFetchedResultsController
*fetchedResultsController;
@property (nonatomic, retain) NSManagedObjectContext
*managedObjectContext;

@property (nonatomic, retain) UITabBarController
*theTabBarController;
@property (nonatomic, retain) NSMutableArray *recents;
@property (nonatomic, retain) NSMutableArray *favorites;
@property (nonatomic, retain) PersonInfo
*personInfoViewController;

- (void)openPersonInfo:(ABRecordRef)person:
(id<DirectoryFavoritesDelegate>)delegate:(NSUInteger)index:(BOOL)
isFavorite;

@end
```

Listato 11.2: DirectoryRoot.h

Scheda descrittiva

Nome: DirectoryRoot Tipo: Controllore	
Campi	
Nome	Descrizione
directoryRecents	Riferimento all'oggetto controllore che gestisce la visualizzazione della lista dei recenti.
directoryFavorites	Riferimento all'oggetto controllore che gestisce la visualizzazione della lista dei preferiti.
theTabBarController	Gestisce l'interazione con la TabBar e il caricamento del ViewController corrispondente alla selezione dell'utente.

recents	Array temporaneo contente i profili dei recenti salvati nella persistenza.
favorites	Come il precedente, ma per i profili dei preferiti.
_fetchedResultsController	Coordina e semplifica il caricamento e il salvataggio degli oggetti provenienti dalla persistenza. Non è obbligatorio e al momento non viene utilizzato.
managedObjectContext	Dirige tutte le operazioni effettuate sul Context dell'applicazione, come fetch, save e delete, collaborando col Coordinator.
Metodi	
Nome	Descrizione
openPersonInfo:delegate:index	Consente l'apertura di un profilo caricando il controllore responsabile, a cui viene passato un profilo compatibile con AddressBook.
Protocolli Conformati	
Nome	Metodi implementati/ridefiniti
UITabBarDelegate	tabBarController:didSelectViewController;
UIAlertViewDelegate	alertView:clickedButtonAtIndex;
DirectoryRecentsDelegate	Tutti.
DirectoryFavoritesDelegate	Tutti.
NSFetchedResultsControllerDelegate	fetchedResultsController;
ABUnknownPersonViewControllerDelegate	unknownPersonViewController:didResolveToPerson; unknownPersonViewController:shouldPerformDefaultActionForPerson:property:identifier;

Tabella 11.3: Interfaccia DirectoryRoot.

DirectorySearch

Header

```
#import <UIKit/UIKit.h>
#import "UNIVEDirectorySearch.h"
#import "DirectoryRecents.h"
#import "DirectoryFavorites.h"
#import "SearchParams.h"

@interface DirectorySearch : UIViewController
<UISearchBarDelegate, UITableViewDelegate, UITableViewDataSource,
UNIVEDirectorySearchDelegate> {

    UNIVEDirectorySearch *dirSearchEngine;
    DirectoryRecents *directoryRecents;
    DirectoryFavorites *directoryFavorites;
    SearchParams *searchParamsController;
    IBOutlet UITableView *theTableView;
    IBOutlet UISearchBar *theSearchBar;
    UIView *activityIndicator;
    UIButton *flipIndicatorButton;
    UIView *theOverlay;
    BOOL frontViewIsVisible;
    BOOL showsResults;
    TKEmptyView *emptyView;
}

@property (assign) BOOL frontViewIsVisible;
@property (assign) BOOL showsResults;
@property (nonatomic, retain) SearchParams
*searchParamsController;
@property (nonatomic, retain) UIButton *flipIndicatorButton;
@property (nonatomic, retain) UIView *theOverlay;
@property (nonatomic, retain) UIView *activityIndicator;
@property (nonatomic, retain) TKEmptyView *emptyView;

- (void)createEmptyView;
- (IBAction)switchViews:(id)sender;
- (void)transitionDidStop:(NSString *)animationID
finished:(NSNumber *)finished context:(void *)context;
- (void)showActivityIndicator;
- (void)scrollToBestPosition;

@end
```

Listato 11.3: DirectorySearch.h

Scheda descrittiva

Nome: DirectorySearch Tipo: Controllore	
Campi	
Nome	Descrizione
dirSearchEngine	Oggetto che si occupa di effettuare ricerche nella directory.
directoryRecents	Riferimento all'oggetto controllore che gestisce la visualizzazione della lista dei recenti.
directoryFavorites	Riferimento all'oggetto controllore che gestisce la visualizzazione della lista dei preferiti.
theTableView	Rappresenta la tabella in cui vengono visualizzati i risultati di una ricerca.
theSearchBar	La barra di ricerca.
activityIndicator	Spinner che indica l'attesa di un'operazione in corso.
flipIndicatorButton	Pulsante per la visualizzazione della schermata dei parametri di ricerca avanzati. Al momento non implementata.
theOverlay	Overlay scuro che ricopre la parte di schermo sovrastante la tastiera quando questa compare.
frontViewIsVisible	NO se al momento viene visualizzata la schermata dei parametri avanzati.
showsResults	Permette di gestire logicamente la visualizzazione di una tabella vuota, quando uguale a NO.
emptyView	Vista che sostituisce la schermata di ricerca quando la directory non è raggiungibile.
Metodi	
Nome	Descrizione

createEmptyView	Instanzia e/o aggiunge alla vista principale l'oggetto emptyView.
switchViews	Alterna la visualizzazione dei risultati e la configurazione dei parametri avanzati con una transizione.
transitionDidStop:finished:context	Metodo chiamato quando la transizione iniziata dal metodo precedente è terminata.
showActivityIndicator	Mostra lo spinner la ricerca è in corso.
scrollToBestPosition	Esegue lo scroll fino in cima alla tabella quando vengono caricati i risultati di una nuova ricerca.
Protocolli Conformati	
Nome	Metodi implementati/ridefiniti
UISearchBarDelegate	searchBarShouldEndEditing; searchBarCancelButtonClicked; searchBarTextDidBeginEditing; searchBarSearchButtonClicked
UITableViewDelegate	tableView:willDisplayCell:forRowAtIndexPath; tableView:viewForHeaderInSection; setTableViewHeader (di supporto al precedente); sectionIndexTitlesForTableView; tableView:didSelectRowAtIndexPath
UITableViewDataSource	tableView:cellForRowAtIndexPath; numberOfSectionsInTableView; tableView:numberOfRowsInSection
UNIVEDirectorySearchDelegate	Tutti.

Tabella 11.4: Interfaccia DirectorySearch.

DirectoryRecents

Header

```
#import <UIKit/UIKit.h>
#import <TapkuLibrary/TapkuLibrary.h>
#import "DirectoryFavorites.h"

@protocol DirectoryRecentsDelegate;

@interface DirectoryRecents : UIViewController
<UITableViewDelegate, UITableViewDataSource> {

    DirectoryFavorites *directoryFavorites;
    IBOutlet UITableView *theTableView;
    id<DirectoryRecentsDelegate> delegate;
    TKEmptyView *emptyView;
}

@property(nonatomic, assign) id<DirectoryRecentsDelegate>
delegate;

- (void)reloadTableData;
- (void)checkIfEmpty;

@end

@protocol DirectoryRecentsDelegate

- (void)loadRecents;
- (void)addRecent:(NSDictionary *)profile;
- (void)alertClear;
- (void)clearRecents;
- (NSArray *)getRecents;

@end
```

Listato 11.4: DirectoryRecents.h

Scheda descrittiva

Nome: DirectoryRecents Tipo: Controllore	
Campi	
Nome	Descrizione
directoryFavorites	Riferimento all'oggetto controllore che gestisce la visualizzazione della lista dei preferiti.
theTableView	Rappresenta la tabella in cui viene visualizzata la lista dei recenti.
delegate	Oggetto delegato che implementa i metodi del protocollo DirectoryRecentsDelegate.
emptyView	Vista che sostituisce la tabella dei recenti quando questa è vuota.
Metodi	
Nome	Descrizione
reloadTableData	Ricarica la tabella quando sono state apportate modifiche alla persistenza.
checkIfEmpty	Controlla se ci sono elementi nella tabella, altrimenti instanzia e visualizza emptyView.
Protocolli Conformati	
Nome	Metodi implementati/ridefiniti
UITableViewDelegate	tableView:willDisplayCell:forRowAtIndexPath; tableView:didSelectRowAtIndexPath
UITableViewDataSource	tableView:cellForRowAtIndexPath; numberOfSectionsInTableView; tableView:numberOfRowsInSection

Tabella 11.5: Interfaccia DirectoryRecents.

Protocolli e oggetti delegati

Il controller DirectoryRecents delega all'oggetto conforme al protocollo seguente la gestione della persistenza degli oggetti corrispondenti ai profili visti di recente.

Nome: DirectoryRecentsDelegate Tipo: Protocollo	
Metodi	
Nome	Descrizione
loadRecents	Carica dalla persistenza una lista dei profili consultati di recente in ordine cronologico decrescente.
addRecent	Aggiunge il risultato di una ricerca appena consultato nella lista dei recenti e lo salva nella persistenza.
alertClear	Mostra un Alert all'utente per chiedere conferma dell'eliminazione di tutti i recenti.
clearRecents	Svuota la lista dei recenti e salva le modifiche nella persistenza.
getRecents	Ridefinisce il getter dell'array dei recenti per sfruttare il *lazy loading*.

Tabella 11.6: Protocollo DirectoryRecentsDelegate.

DirectoryFavorites

Header

```
#import <UIKit/UIKit.h>
#import <TapkuLibrary/TapkuLibrary.h>


@protocol DirectoryFavoritesDelegate;

@interface DirectoryFavorites : UIViewController
<UITableViewDelegate, UITableViewDataSource> {

    IBOutlet UITableView *theTableView;
    id<DirectoryFavoritesDelegate> delegate;
    TKEmptyView *emptyView;
}

@property(nonatomic, assign) id<DirectoryFavoritesDelegate>
delegate;

- (IBAction)toggleEdit:(id)sender;
- (void)reloadTableData;
- (void)checkIfEmpty;

@end

@protocol DirectoryFavoritesDelegate

- (void)loadFavorites;
- (BOOL)addFavorite:(NSUInteger)index;
- (void)deleteFavorite:(NSUInteger)index;
- (NSArray *)getFavorites;

@end
```

Listato 11.5: DirectoryFavorites.h

Scheda descrittiva

Nome: DirectoryFavorites Tipo: Controllore	
Campi	
Nome	Descrizione
theTableView	Rappresenta la tabella in cui viene visualizzata la lista dei preferiti.
delegate	Oggetto delegato che implementa i metodi del protocollo DirectoryFavoritesDelegate.
emptyView	Vista che sostituisce la tabella dei preferiti quando questa è vuota.
Metodi	
Nome	Descrizione
toggleEdit	Attiva la modalità editing della tabella quando viene premuto il pulsante modifica oppure quando l'utente elimina con un tocco una riga.
reloadTableData	Ricarica la tabella quando sono state apportate modifiche alla persistenza.
checkIfEmpty	Controlla se ci sono elementi nella tabella, altrimenti instanzia e visualizza emptyView.
Protocolli Conformati	
Nome	Metodi implementati/ridefiniti
UITableViewDelegate	tableView:willDisplayCell:forRowAtIndexPath; tableView:didSelectRowAtIndexPath; tableView:didEndEditingRowAtIndexPath

UITableViewDataSource	tableView:cellForRowAtIndexPath; numberOfSectionsInTableView; tableView:numberOfRowsInSection; tableView:commitEditingStyle:forRowAtIndexPath; tableView:canEditRowAtIndexPath

Tabella 11.7: Interfaccia DirectoryFavorites.

Protocolli e oggetti delegati

Il controller DirectoryFavorites delega all'oggetto conforme al protocollo seguente la gestione della persistenza degli oggetti corrispondenti ai profili aggiunti nei preferiti.

Nome: DirectoryFavorites Delegate Tipo: Protocollo	
Metodi	
Nome	Descrizione
loadFavorites	Carica dalla persistenza una lista dei profili aggiunti dall'utente nei preferiti.
addFavorite	Aggiunge il profilo visualizzato nei preferiti, salvandolo nella persistenza.
deleteFavorite	Elimina il preferito corrispondente alla riga della tabella eliminata, salvando le modifiche nella persistenza.
getFavorites	Ridefinisce il getter dell'array dei preferiti per sfruttare il *lazy loading*.

Tabella 11.8: Protocollo DirectoryFavorites Delegate.

PersonInfo

Header

```
#import <UIKit/UIKit.h>
```

```
#import <AddressBook/AddressBook.h>
#import <AddressBookUI/AddressBookUI.h>
#import "DirectoryFavorites.h"

@interface PersonInfo : ABUnknownPersonViewController
<ABUnknownPersonViewControllerDelegate> {
    id<DirectoryFavoritesDelegate> delegate;
    BOOL isFavorite;
    NSUInteger index;
}
@property (nonatomic, assign) id<DirectoryFavoritesDelegate>
delegate;
@property (nonatomic, assign) BOOL isFavorite;
@property (nonatomic, assign) NSUInteger index;
- (void)addFavorite:(NSObject *)profile;

@end
```

Listato 11.6: PersonInfo.h

Scheda descrittiva

Nome: PersonInfo Tipo: Controllore	
Campi	
Nome	Descrizione
delegate	Oggetto delegato che implementa i metodi del protocollo DirectoryFavoritesDelegate.
isFavorite	Stabilisce se il profilo è già tra i preferiti.
index	E' l'indice del profilo nell'array dei recenti o dei preferiti, a seconda del valore del campo precedente.
Metodi	
Nome	Descrizione
addFavorite	Se isFavorite è uguale a NO, chiede al delegate di aggiungere il profilo ai preferiti..

Protocolli Conformati	
Nome	Metodi implementati/ridefiniti
ABUnknownPersonViewController Delegate	unknownPersonViewController:didResolveToPerson; unknownPersonViewController:shouldPerformDefaultActionForPerson:property:identifier;

Tabella 11.9: Interfaccia PersonInfo.

DIAGRAMMI DI SEQUENZA

Un diagramma di sequenza, o *sequence diagram*, è un diagramma previsto dall'*UML*, il cui scopo primario è quello di mostrare le interazioni tra oggetti nell'ordine sequenziale in cui avvengono tali interazioni[15]. I diagrammi di sequenza, al contrario del diagramma delle classi, non servono solamente agli sviluppatori, ma anche all'organizzazione di progetto per comunicare al proprio staff il funzionamento attuale del sistema e definire eventuali nuovi requisiti per una futura implementazione del sistema. Durante la fase di definizione dei requisiti, gli *use case* vengono spesso raffinati in uno o più diagrammi di sequenza, per mezzo dei quali è possibile documentare come dovrà comportarsi il sistema che si andrà a sviluppare.

In questo lavoro di tesi non sono stati sviluppati i diagrammi di sequenza di tutti i requisiti funzionali di iCa'Foscari, bensì solamente quelli della funzionalità Directory. Per di più, i diagrammi riportati nelle pagine seguenti sono stati ulteriormente approfonditi e arricchiti dopo aver ultimato lo sviluppo del sottosistema, in maniera tale da illustrare con maggiore dettaglio come avviene effettivamente l'interazione tra gli oggetti coinvolti, specificando classi e tipi propri dell'*SDK* e del linguaggio *Objective-C*.

Di seguito è riportata schematicamente la legenda dei diagrammi di sequenza.

[15]http://www.ibm.com/developerworks/rational/library/3101.html

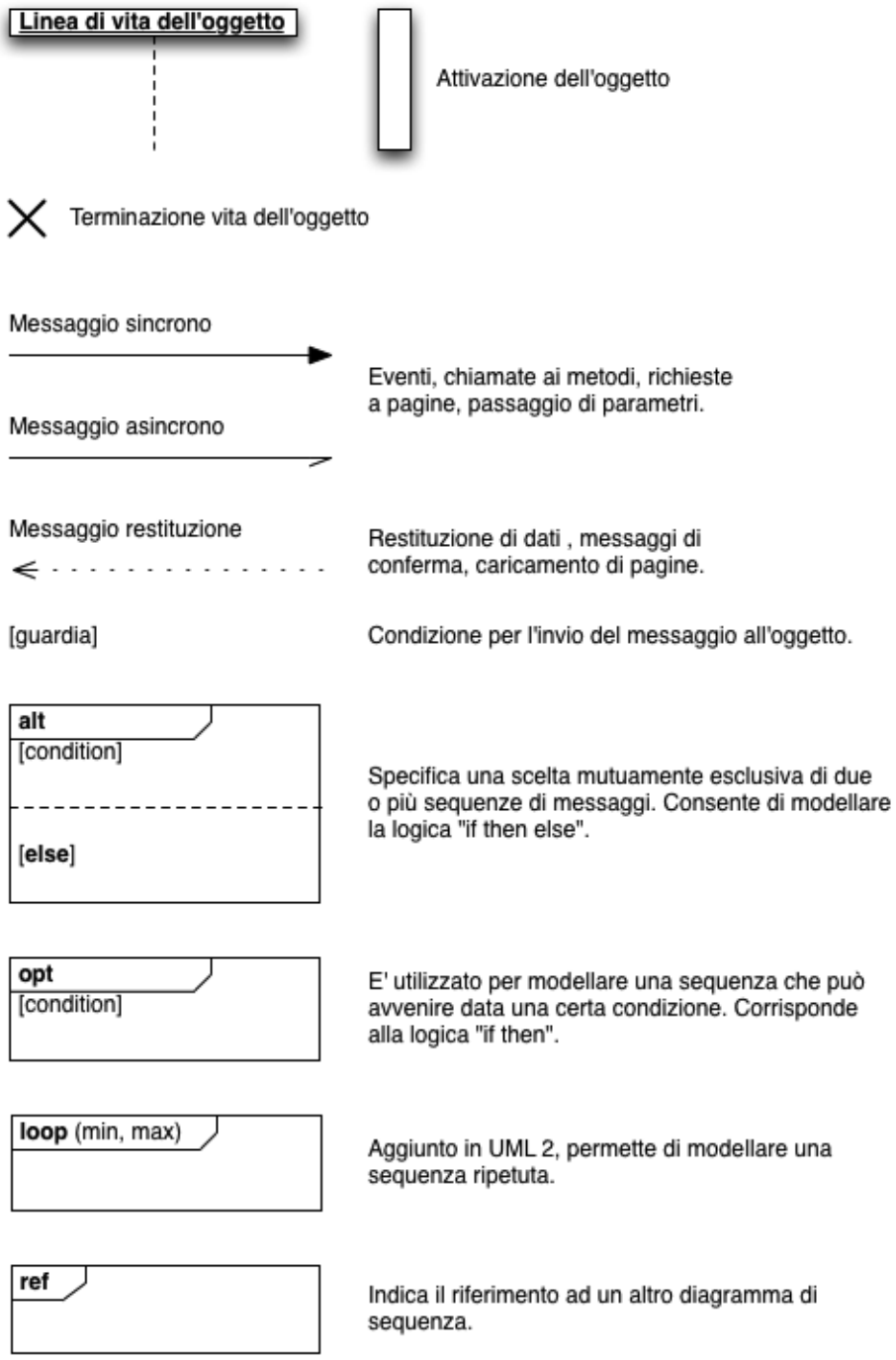

Figura 11.1: Legenda diagramma di sequenza.

Ricerca di un docente nella directory

In questo scenario di utilizzo, l'utente effettua la ricerca di un docente nella directory inserendo una *query* nella *SearchBar*, ad esempio il cognome, e premendo il pulsante "Cerca" della tastiera virtuale.

La ricerca può avere tre esiti distinti, a seconda dei quali il sottosistema reagisce in maniera differente:

- Nel primo caso, si può essere verificato un errore sconosciuto nella directory, oppure un problema nella connessione, e l'utente viene informato dell'errore con un *Alert*;
- Nel secondo caso invece, la directory non ha restituito alcun risultato. L'utente viene allora avvisato sempre per mezzo di un *Alert*;
- Nell'ultimo caso, la ricerca è andata a buon fine e il controller aggiorna la tabella dei risultati ottenuti dalla directory.

I diagrammi riportati di seguito descrivono le interazioni che avvengono in ciascuno di questi tre casi.

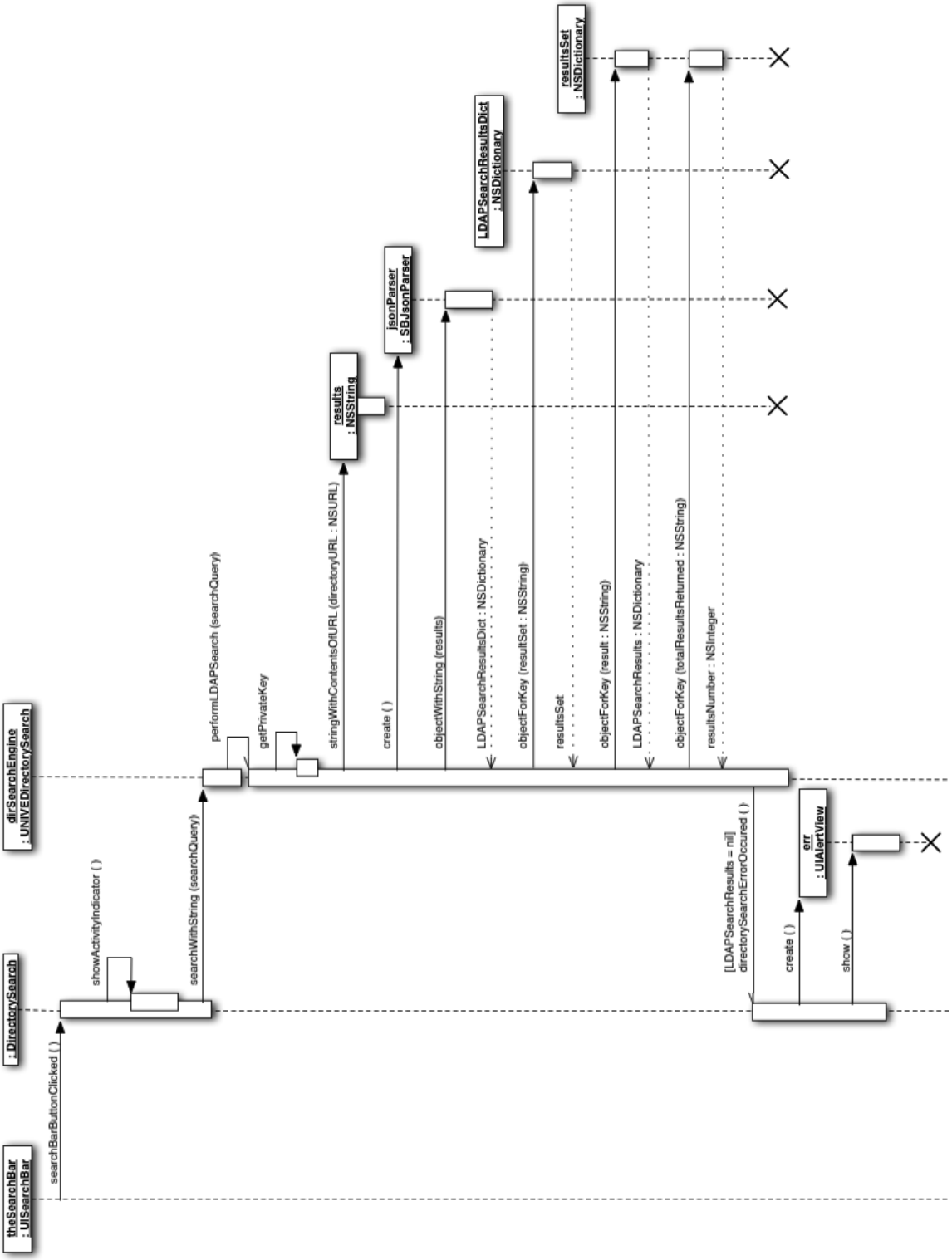

Figura 11.2: Sequenza ricerca docente, caso 1.

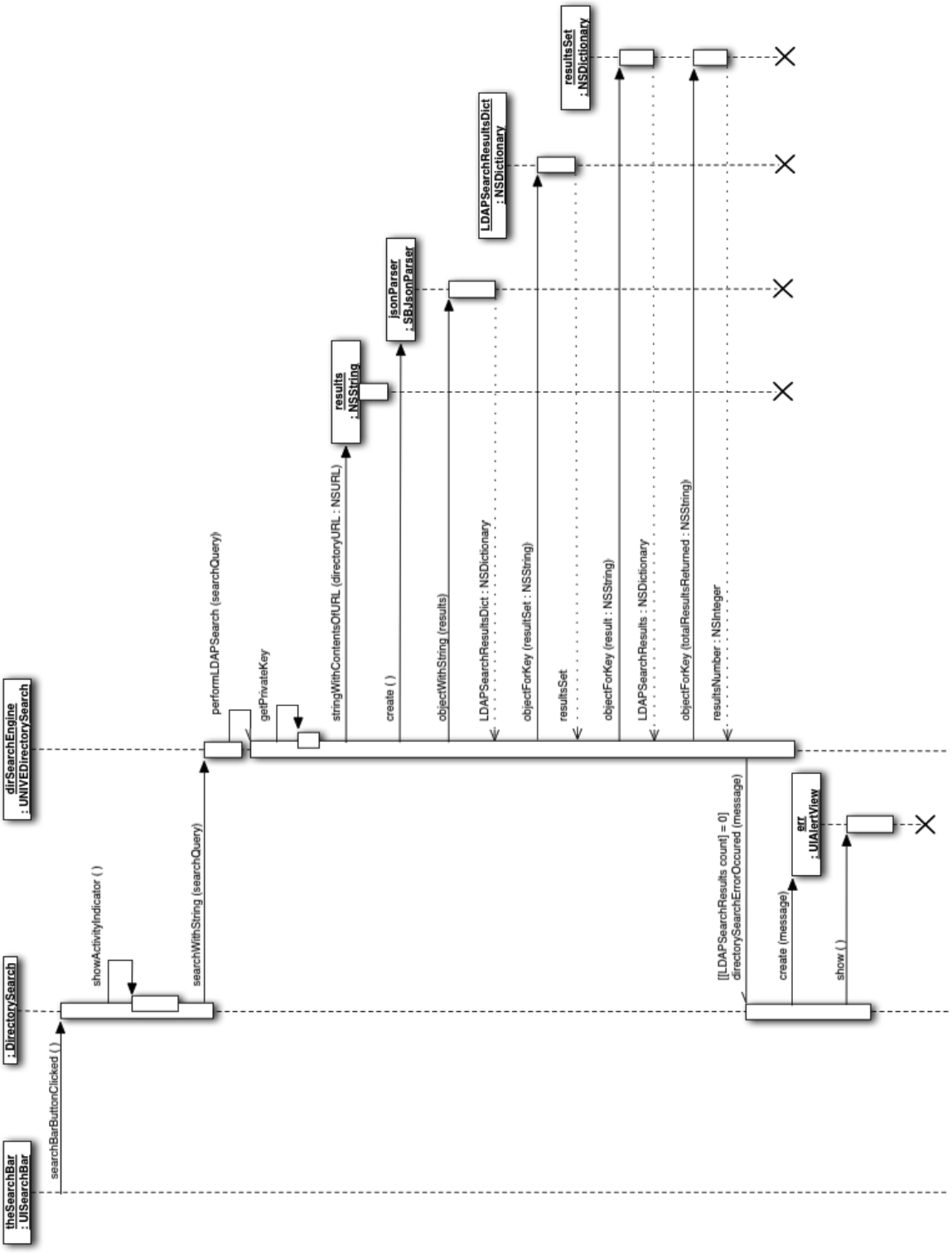

Figura 11.3: Sequenza ricerca docente, caso 2.

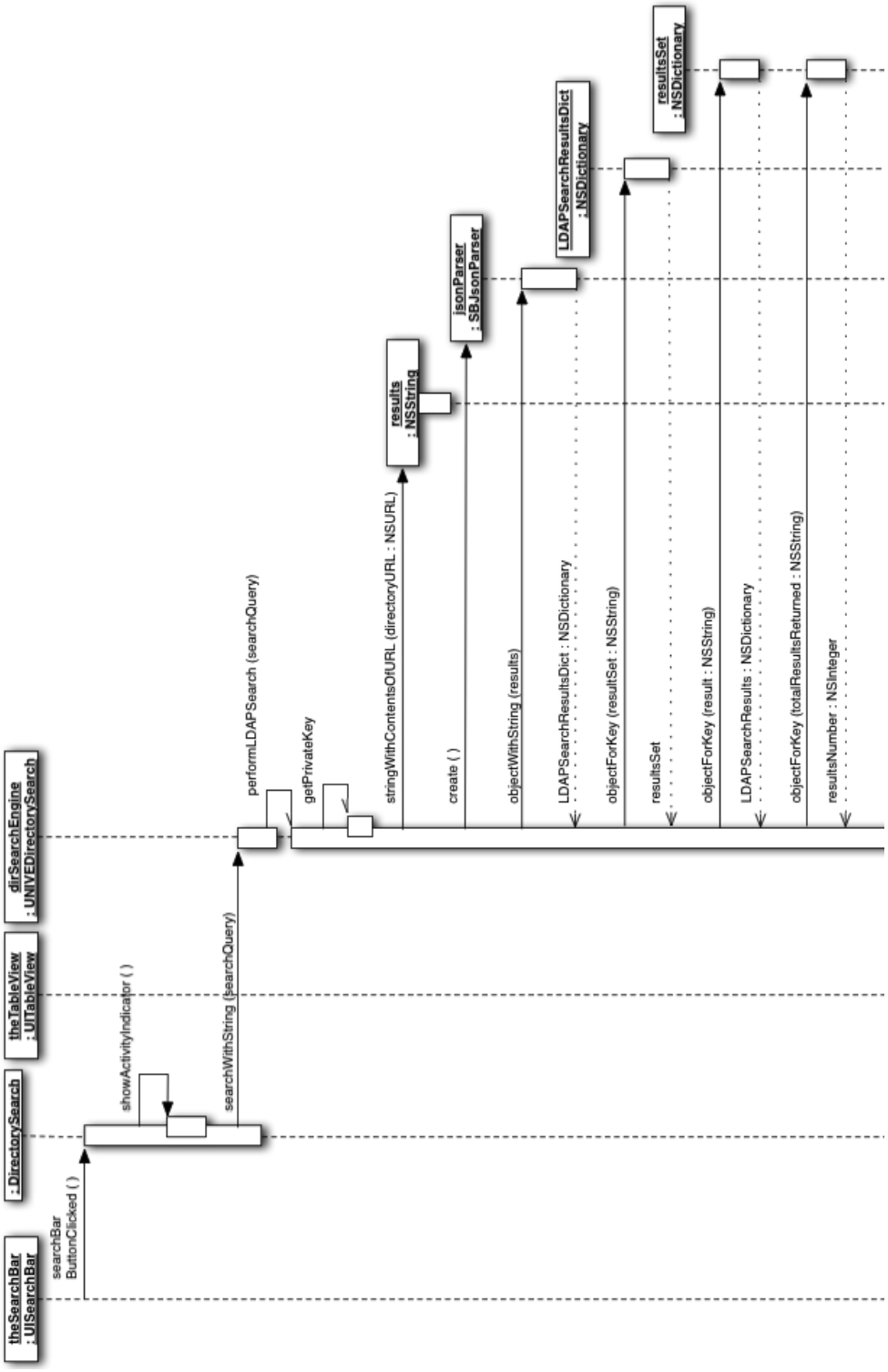

Figura 11.4: Sequenza ricerca docente, caso 3 parte 1.

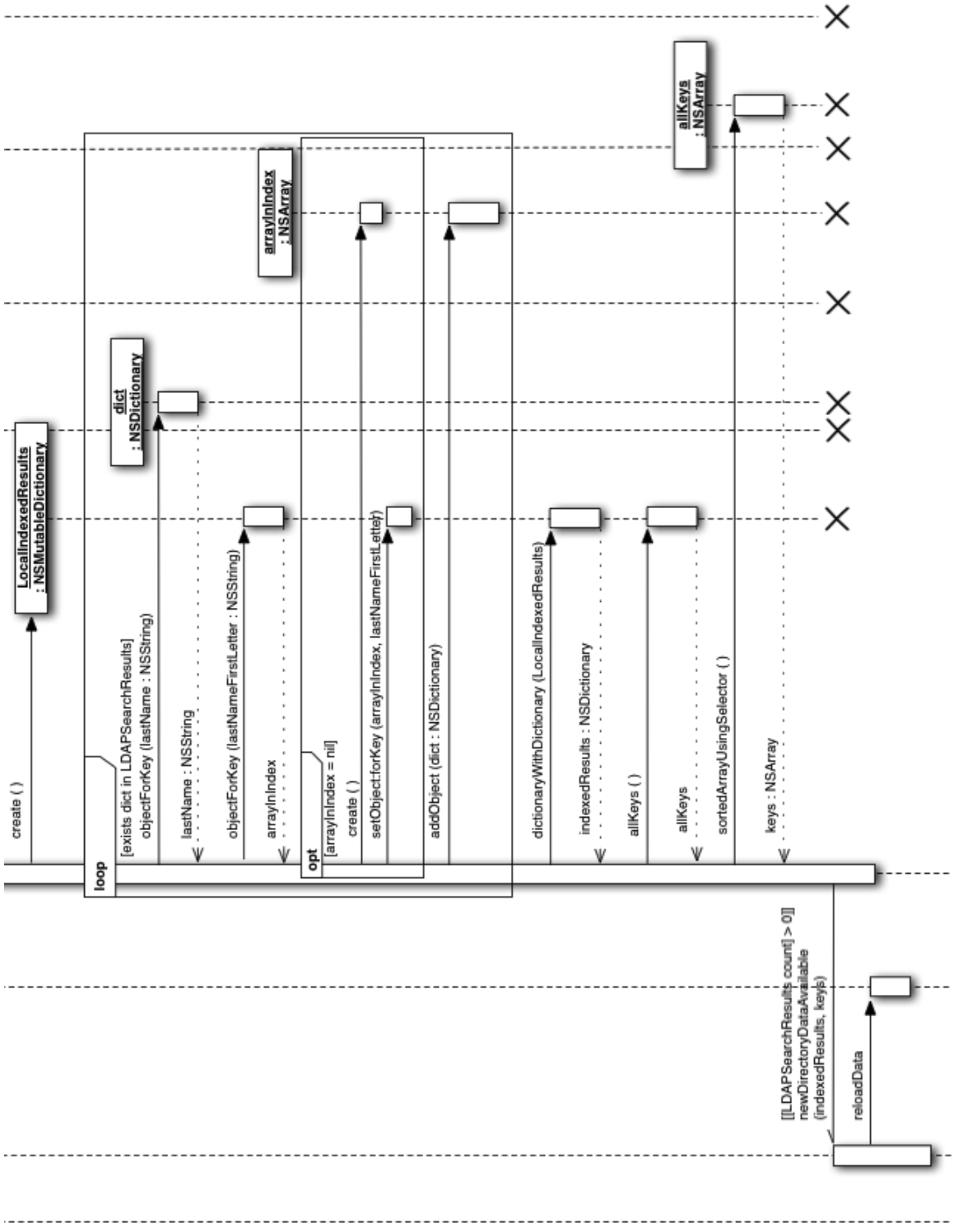

Figura 11.5: Sequenza ricerca docente, caso 3 parte 2.

Visualizzazione di un risultato della ricerca

Questo scenario si verifica quando l'utente seleziona dalla tabella dei risultati il nominativo corrispondente al docente per il quale vuole essere visualizzato il suo profilo.

L'apertura di un profilo comporta, inoltre, l'inserimento dello stesso nominativo nella lista dei recenti. Per cui, il compito delle prossime sequenze sarà quello di descrivere in che modo interagiscono gli oggetti quando viene eseguito il corpo dei metodi che si occupano di svolgere questo scenario.

Di seguito sono elencati i metodi per i quali è stato incluso un diagramma di sequenza specifico:

- Il metodo addRecent( ), la cui responsabilità è quella di inserire nel modello dati persistente l'ultimo profilo consultato;
- A seguire, il metodo fetchExistent( ), di supporto al precedente, si occupa di evitare l'inserimento di recenti duplicati e/o di aggiornare l'ordine del profilo consultato nella lista dei recenti;
- Infine, il metodo loadRecents( ), richiamato più volte nelle sequenze, svolge il caricamento della lista aggiornata dei recenti dal modello persistente.

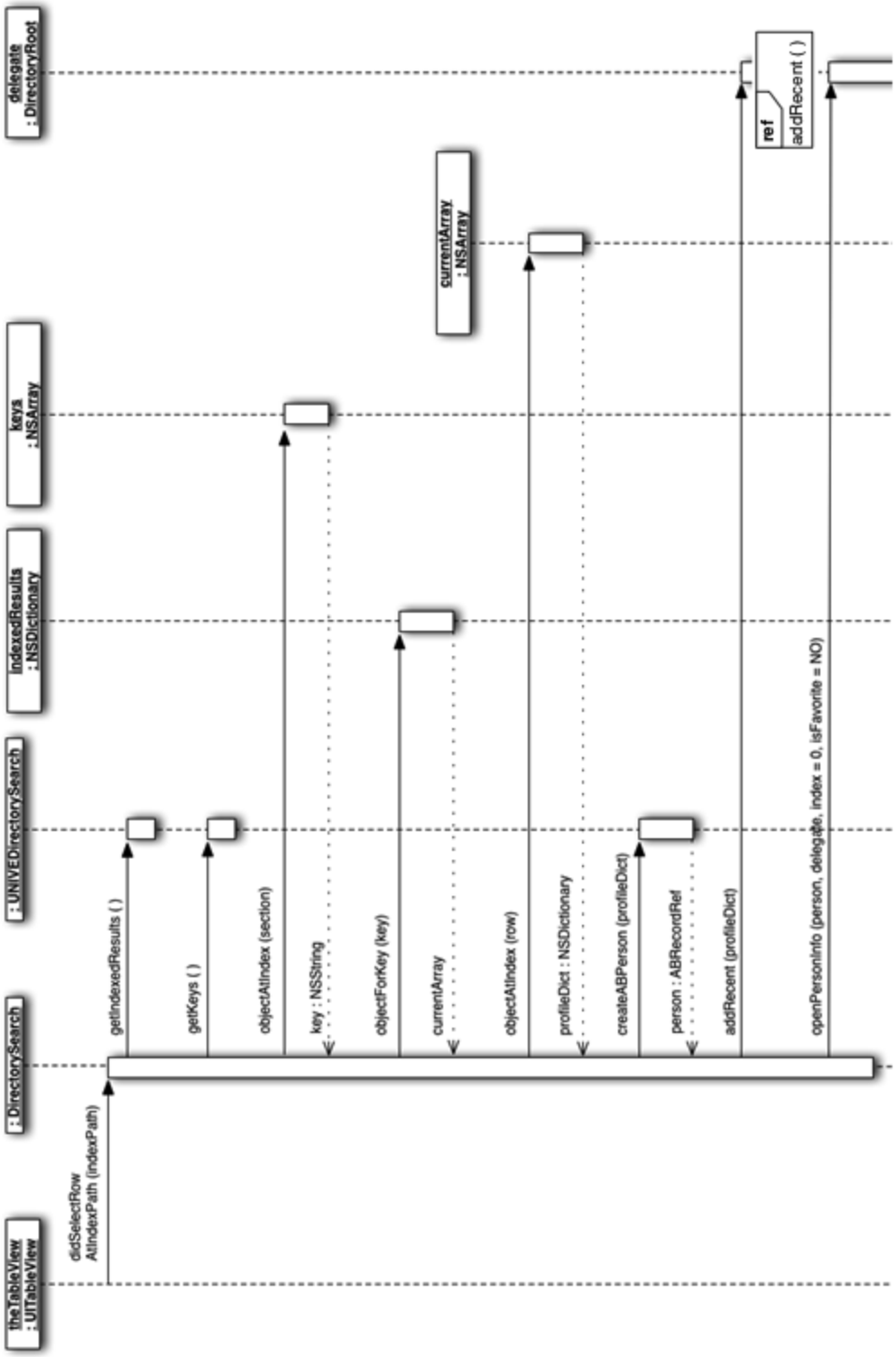

Figura 11.6: Sequenza visualizzazione risultato, parte 1.

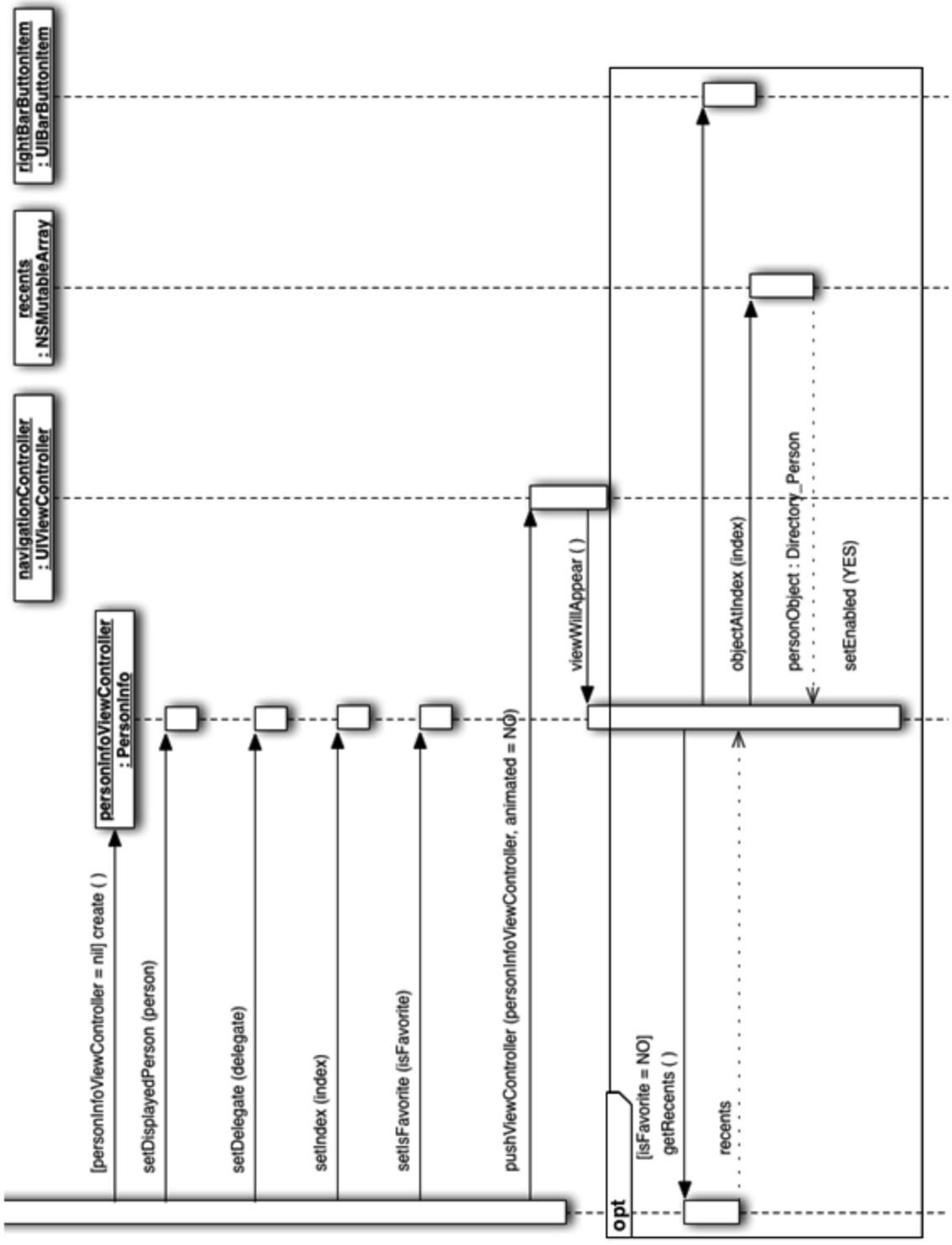

Figura 11.7: Sequenza visualizzazione risultato, parte 2.

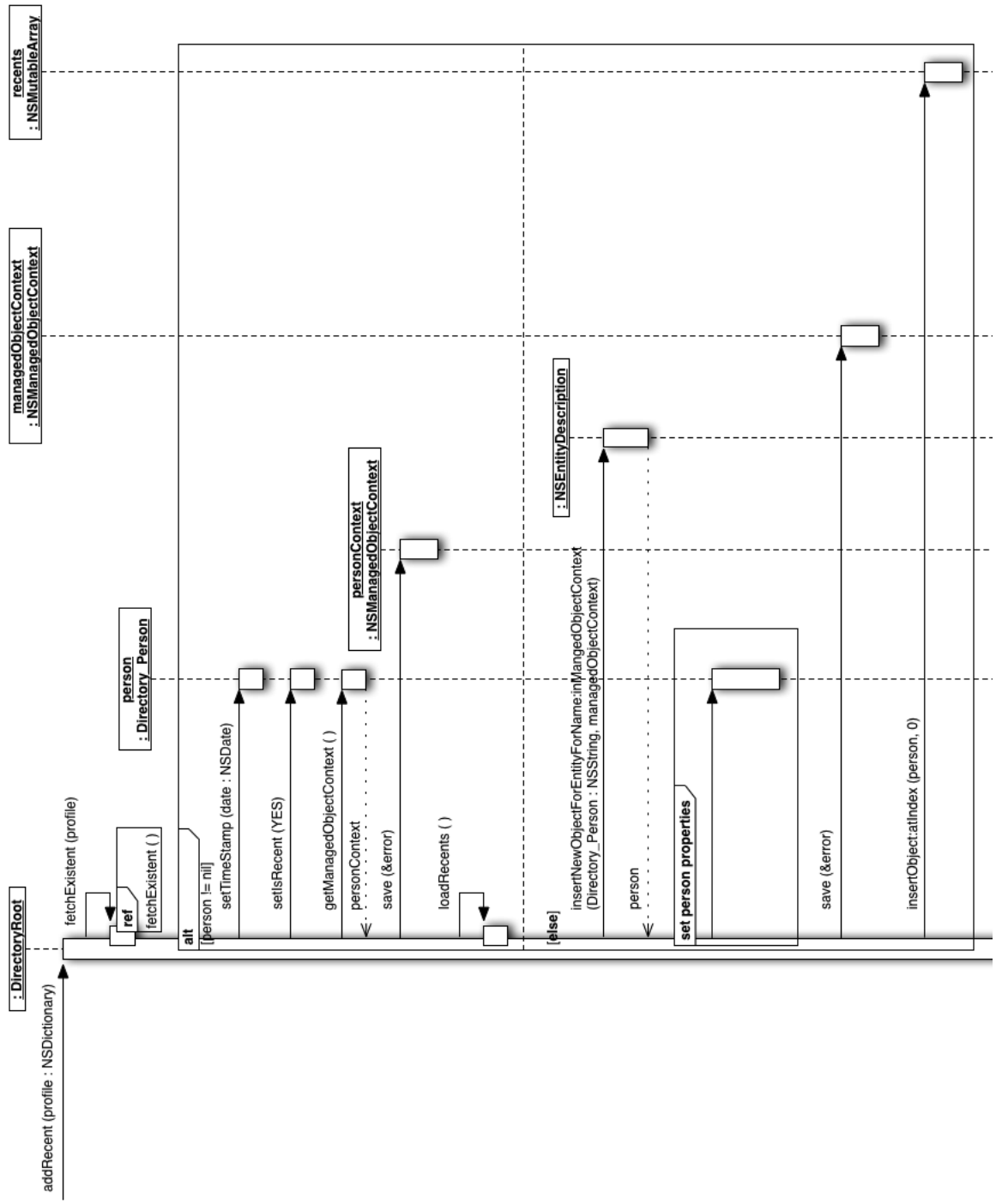

Figura 11.8: Sequenza metodo addRecent( ), parte 1.

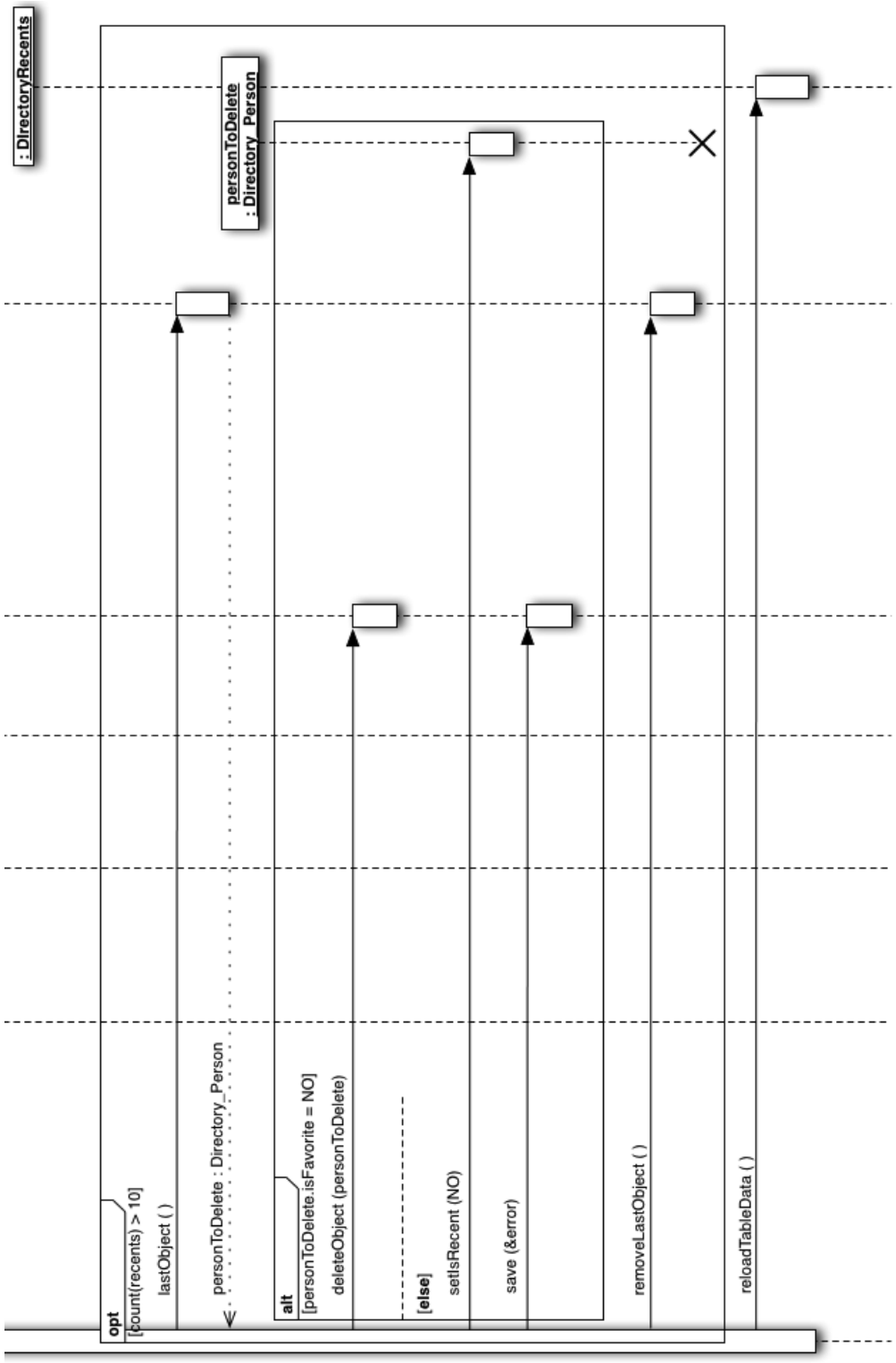

Figura 11.9: Sequenza metodo addRecent( ), parte 2.

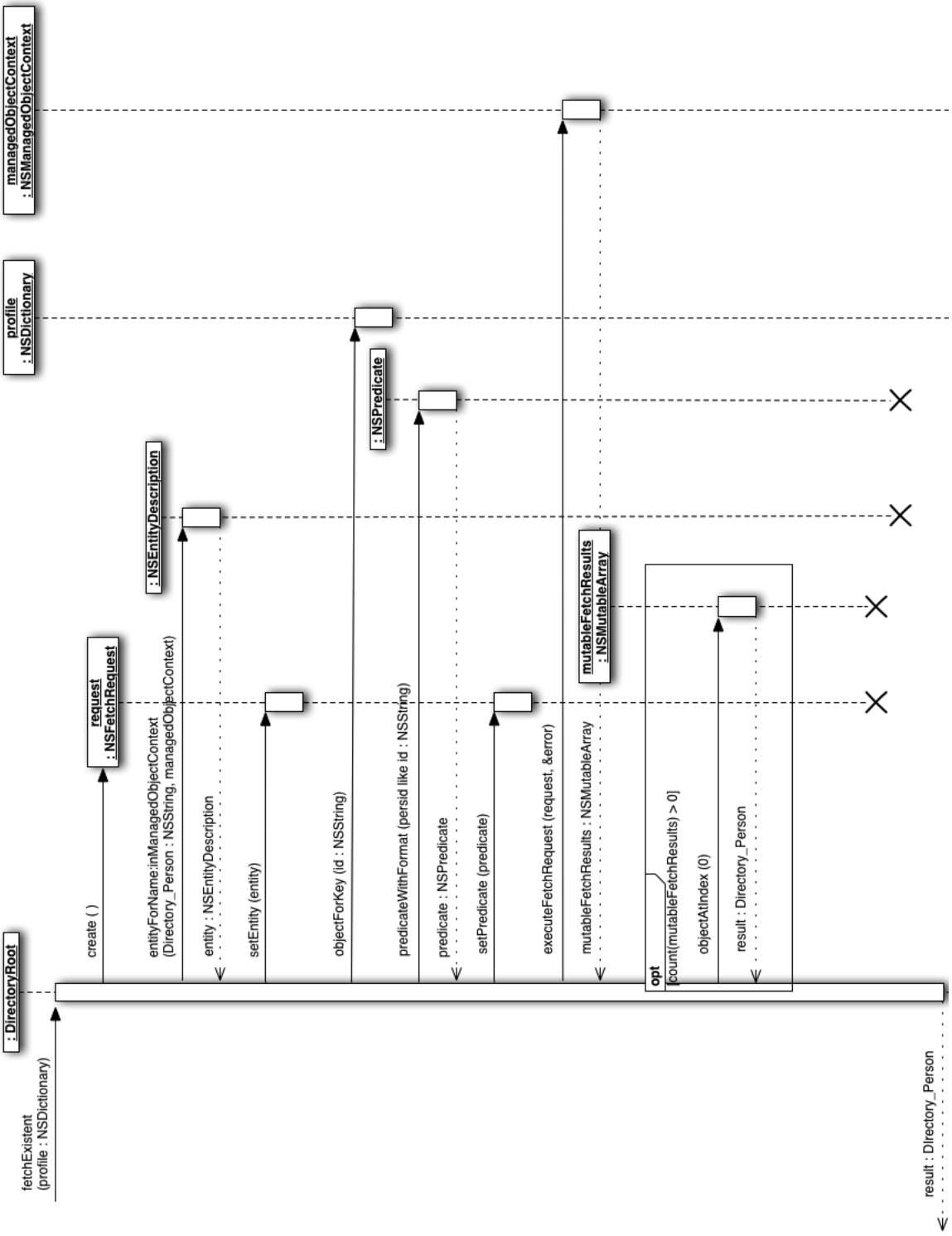

Figura 11.10: Sequenza metodo fetchExistent( ).

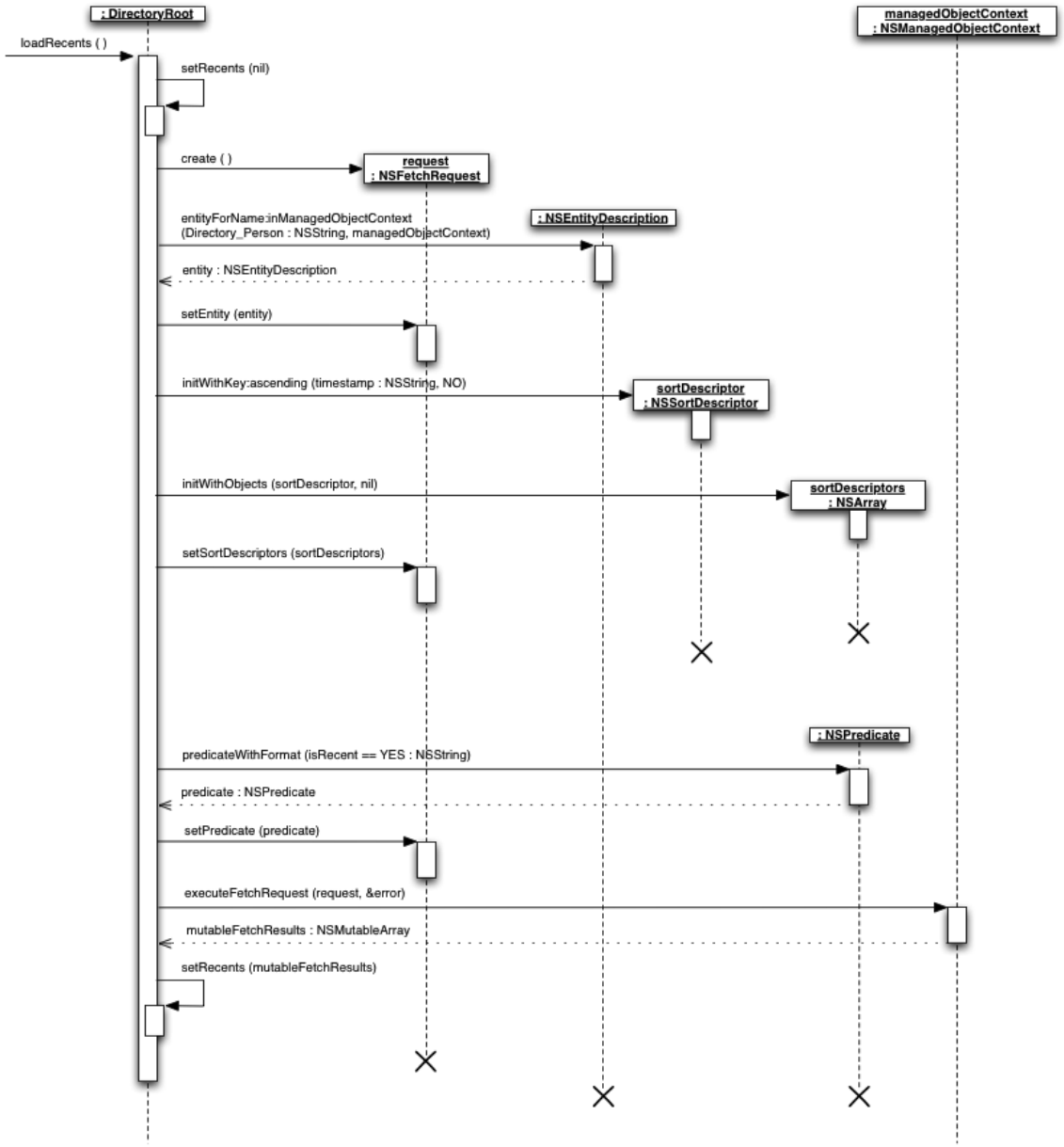

Figura 11.11: Sequenza metodo loadRecents( ).

Aggiunta ed eliminazione di un preferito

L'utente, oltre a visualizzare il profilo di un docente, può salvare il suo nominativo tra i preferiti per consultazioni future. Allo stesso modo, in seguito può rimuoverlo dalla lista dei preferiti.

I diagrammi di sequenza seguenti descrivono questi due scenari, illustrando quanto avviene dopo che l'utente ha premuto il pulsante "aggiungi ai preferiti" dal profilo di un docente, oppure dopo aver eliminato un'occorrenza dalla tabella dei preferiti.

Inoltre, è riportato anche il diagramma del metodo loadFavorites( ), utilizzato in entrambe le operazioni per caricare i preferiti dal modello persistente.

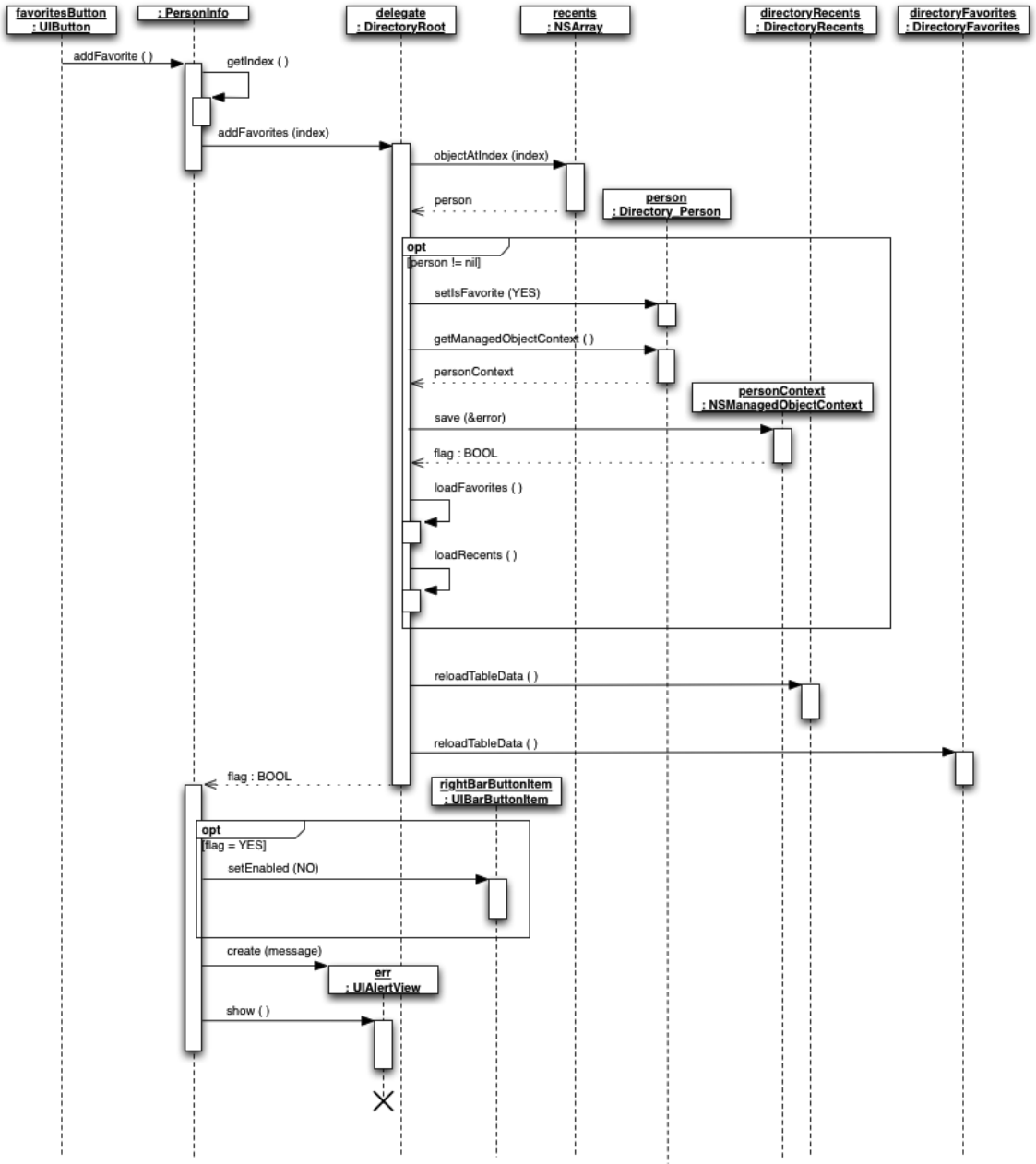

Figura 11.12: Sequenza aggiunta preferito.

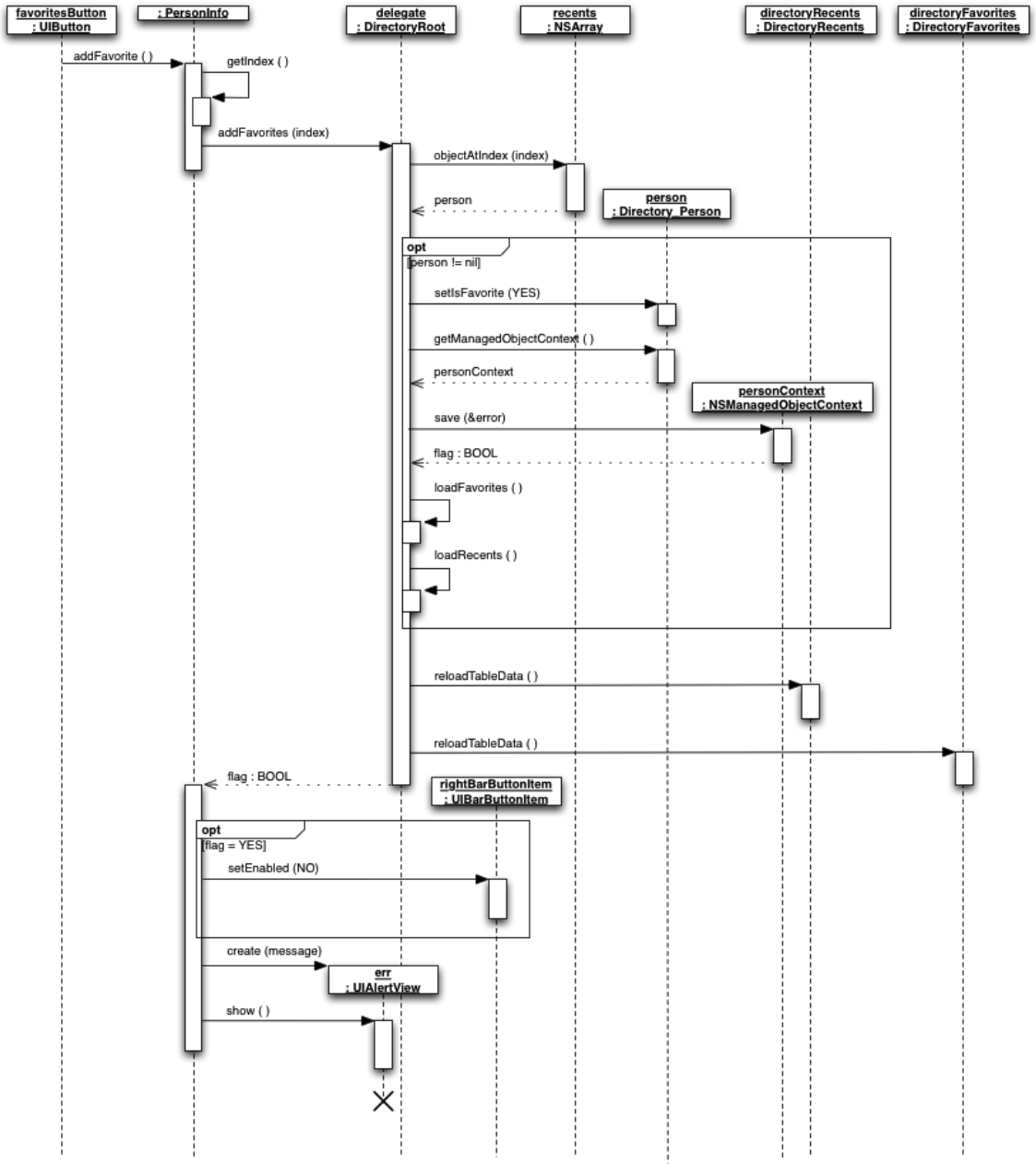

Figura 11.13: Sequenza metodo loadFavorites( ).

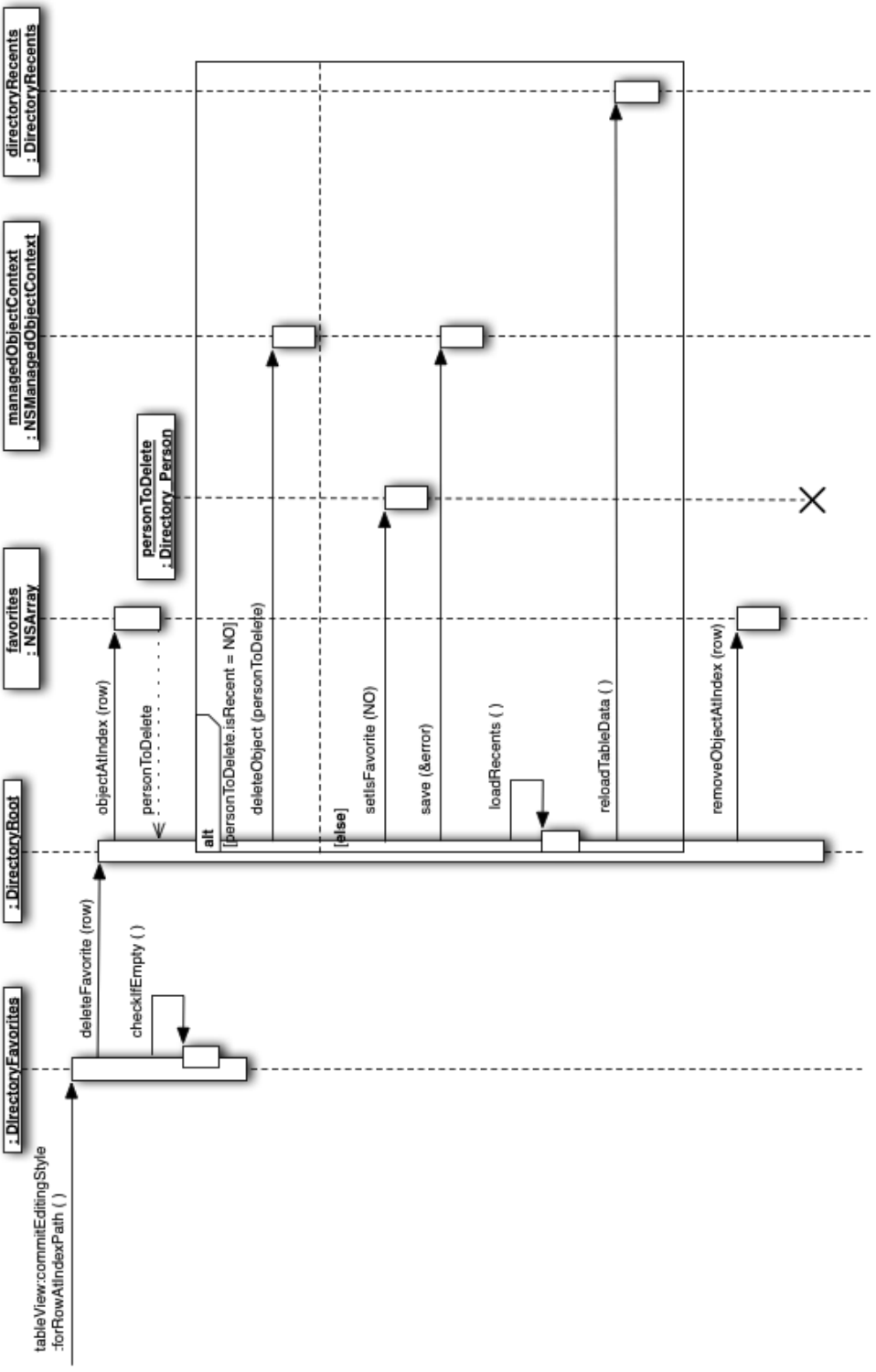

Figura 11.14: Sequenza eliminazione preferito.

Eliminazione della cronologia dei recenti

L'ultima sequenza descrive lo scenario in cui l'utente vuole svuotare la lista dei recenti premendo l'apposito pulsante. All'utente viene presentato un *Alert* di conferma, a cui segue una logica condizionale per l'eliminazione di tutti i recenti dalla persistenza.

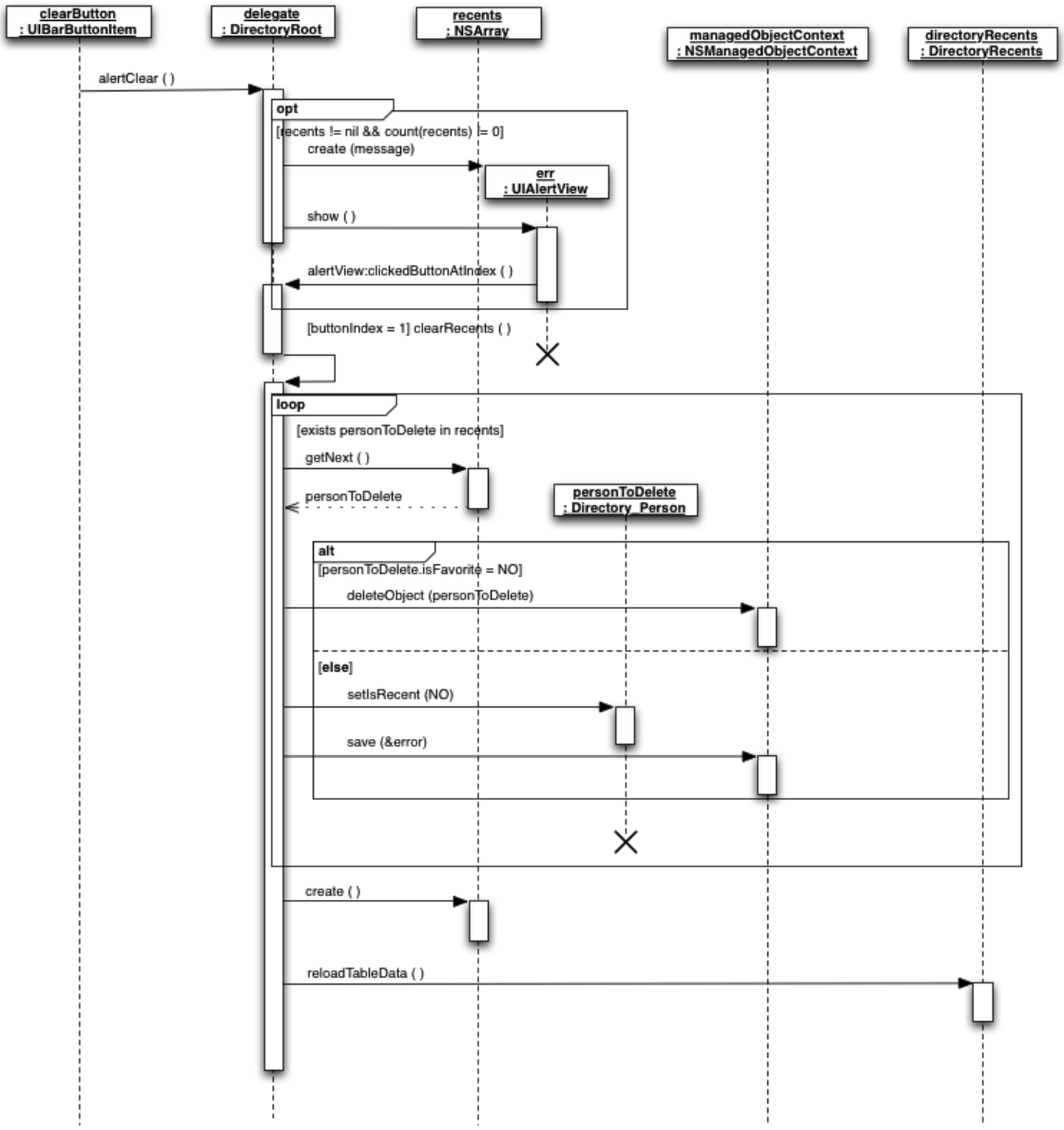

Figura 11.15: Sequenza metodo clearRecents( ).

BEST PRACTICES E OTTIMIZZAZIONI

Nel processo di implementazione di questo requisito funzionale, sono state messe a punto delle pratiche e degli accorgimenti mirati all'ottimizzazione del codice e delle performance dell'applicazione in generale.

I dispositivi *iOS*, essendo dei dispostivi mobili, hanno una potenza di calcolo e una memoria particolarmente limitata. Per questo occorre allocare meno memoria possibile, utilizzandola solo per il tempo strettamente necessario, e testare l'applicazione in condizioni di *low memory warning*. Allo stesso modo, per garantire un funzionamento fluido e dare all'utente l'impressione di un'applicazione sempre reattiva, è necessario effettuare elaborazioni pesanti e caricamenti da sorgenti esterne su *thread* separati.

Di seguito sono allora introdotti alcuni pattern comunemente utilizzati per affrontare questi problemi[16].

Lazy loading

E' un pattern molto diffuso nei framework *Cocoa* e consiste nel *"fare esclusivamente tanto lavoro quanto effettivamente richiesto*", ovvero nell'instanziare risorse in memoria solamente nel momento in cui queste sono invocate, riducendo i tempi di caricamento del sottosistema. Il frammento di codice seguente mostra l'esempio di una risorsa erroneamente instanziata subito nella fase di allocazione di un oggetto:

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

[16]Stanford CS193P Winter 2010 – Lecture 10: Performance

Nel caso precedente l'oggetto myImage è caricato in memoria troppo presto risultando in un utilizzo inefficiente delle risorse, in quanto rischierebbe di non essere mai invocato, oppure di servire molto più tardi nell'esecuzione. L'esempio seguente, invece, mostra come la risorsa dovrebbe essere invece creata solamente nel momento in cui è richiesta:

```
- (UIImage *)myImage
{
      if (myImage == nil) {
            myImage = [self readSomeHugeImageFromDisk];
      }
}
```

Il *lazy loading* è implementabile nella maggior parte dei casi, tutta via esistono situazioni in cui può non essere conveniente e sicuro, ad esempio quando si utilizzano *thread* multipli.

Memory leak

Objective C per *iOS* non supporta la *garbage collection*, di conseguenza è compito dello sviluppatore gestire la vita degli oggetti allocati in memoria.

Ogni oggetto residente in memoria possiede un contatore incrementato ad ogni *alloc* e *retain* e diminuito ad ogni *release*. Spesso, a seconda della complessità del codice, accade che un oggetto viene dereferenziato dallo *stack* prima che il suo contatore sia posto a zero, generando un *memory leak*. Lo sviluppatore dovrà accertare di volta in volta che tutto ciò non si verifichi.

Sebbene questo possa rappresentare un compito arduo, esiste uno strumento incluso nei *developer tools* di *XCode*, chiamato *Instruments*, che permette di individuare facilmente i *leak*, evidenziando le righe di codice direttamente responsabili.

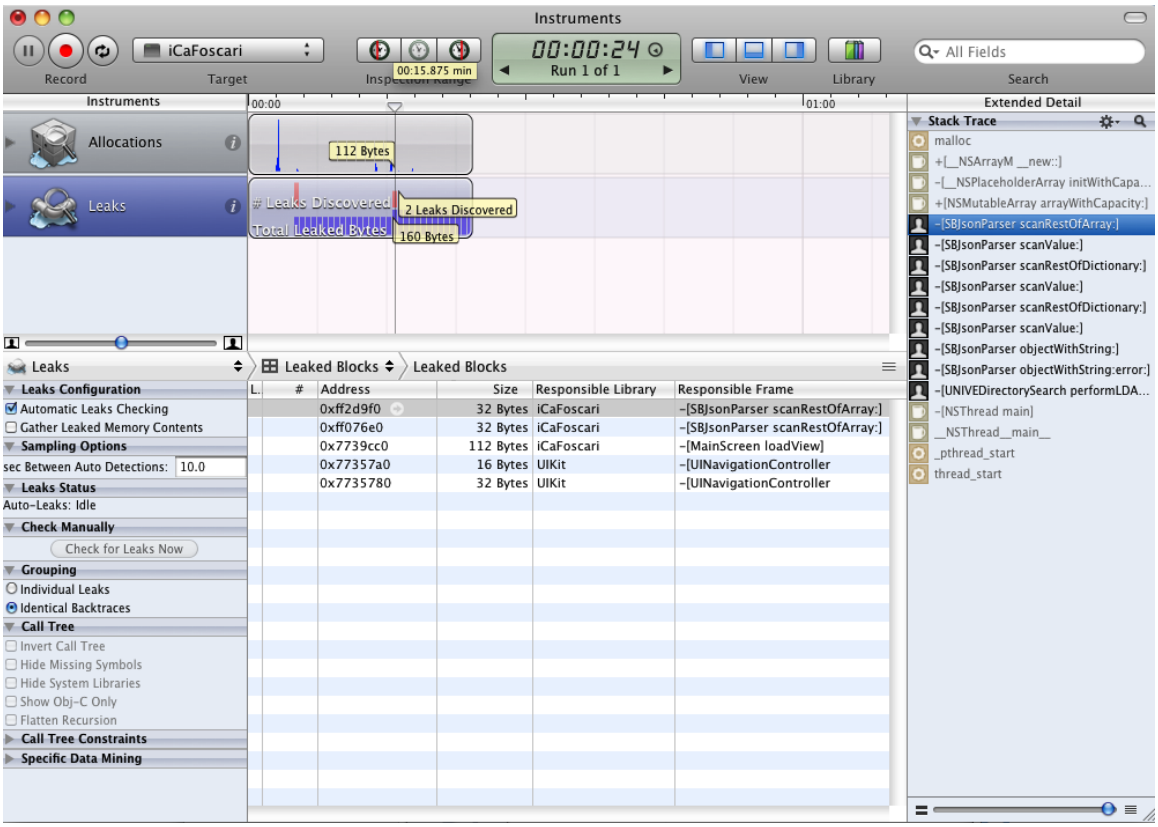

Figura 11.16: Trovare i memory leak con Instruments.

Ogni *leak* è unito a un *backtrace* che indica la porzione di codice responsabile. Esistono dei *leak* anche nel codice di sistema, sebbene questi siano rari, tuttavia è indispensabile considerare prima sempre il proprio codice.

Autorelease e autorelease pool

L'*autorelease* è un modo di allocare gli oggetti che semplifica notevolmente il codice. Ogni oggetto sui cui è effettuato l'*autorelease* viene inserito in un *pool*, quando poi questo viene drenato, esso invoca automaticamente il *release* sugli oggetti che gli appartengono. Infatti, ad ogni iterazione del *run loop* dell'applicazione è creato automaticamente un *autorelease pool*.

Tuttavia, ciò può diventare controproducente nel caso in cui il numero dei oggetti nel *pool* sia eccessivo, ovvero nel caso di un *high water mark*. E' dunque necessario creare e rilasciare manualmente il proprio *pool* nelle porzioni di codice in cui si fa molto uso di *autorelease*, come ad esempio nei *loop*.

```
for (int i = 0; i < someLargeNumber; i++) {
      NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

      NSString *string = ...;
      string = [string lowercaseString];
      string = [string stringByAppendingString:...];
      NSLog(@"%@", string);

      [pool release];
}
```

Generalmente però, è sempre consigliato utilizzare le procedure *alloc/init/release* per l'allocazione degli oggetti, sempre che ciò sia possibile. Per di più è buona pratica riutilizzare oggetti e strutture dati nei *loop*, dichiarandoli all'esterno del ciclo e aggiornando il proprio valore al suo interno, piuttosto che allocarli e deallocarli ad ogni iterazione; cosa che provocherebbe un *overhead* considerevole quanto inutile.

Concorrenza

Il framework *Foundation* di *iOS* include dei *wrapper* di alto livello che consentono di semplificare l'accesso all'*API POSIX* per la gestione dei *thread*.

La creazione di un nuovo *thread* è spesso utile per effettuare operazioni molto lunghe, come il download di dati da una sorgente, ed evitare di interrompere l'interazione dell'utente con l'applicazione. Per lanciare un nuovo *thread* è sufficiente instanziare un oggetto della classe *NSThread*, creare un apposito *autorelease pool* e definire dei metodi di convenienza per lo scambio di messaggi con il *thread* principale; un chiaro esempio è fornito dal codice seguente:

```
- (void)someAction:(id)sender
{
      // Fire up a new thread [NSThread
      detachNewThreadSelector:@selector(doWork:)
      withTarget:self object:someData];
}

- (void)doWork:(id)someData
{
      NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

      [someData doLotsOfWork];

      // Message back to the main thread [self
      performSelectorOnMainThread:@selector(allDone:)
      withObject:[someData result] waitUntilDone:NO];

      [pool release];
}
```

Ciò nonostante, va tenuto in considerazione che l'impiego dei *thread* comporta anche un codice più difficile da mantenere ed un eventuale decadimento delle prestazioni se vengono lanciati molti *thread* concorrenti. Nel peggiore dei casi, possono inoltre verificarsi condizioni di *deadlock* o *racing* tra *thread*, ma tutto ciò può essere evitato con una buona programmazione e l'uso corretto dell'*API*. Infine, esistono numerose classi non *thread safe*, come quelle di *UIKit*, che possono essere instanziate solamente dal *thread* principale.

CONCLUSIONI

L'implementazione discussa in questo capitolo, che chiude la seconda parte del lavoro di tesi, sebbene abbia prodotto un prototipo funzionante del sottosistema Directory, rappresenta esclusivamente una tra le tante soluzioni compatibili con l'architettura prevista. Nella terza e ultima parte, infatti verranno discussi alcuni requisiti, come ad esempio il salvataggio dello stato o la ricezione di notifiche *PUSH*, di elevata

complessità, che richiedono una rivisitazione della progettazione e l'introduzione di classi *factory* e *singleton*, per un maggior riutilizzo del codice e l'eliminazione di metodi ridondanti, da un lato, e l'adesione dei sottosistemi a una superclasse in comune per l'implementazione di questi nuovi requisiti, dall'altro.

PARTE TERZA

Capitolo 12: Future works

L'ultima parte di questo elaborato si prende cura di offrire una panoramica dei lavori necessari per il completamento di iCa'Foscari. Per di più viene aperta la strada alla realizzazione di un servizio web mobile per l'Università che comprenda la maggior parte dei dispostivi esistenti in questo ecosistema.

COMPLETAMENTO DEL PROGETTO

Il risultato di questa tesi corrisponde alla progettazione completa dell'applicazione iCa'Foscari, dall'architettura all'interfaccia grafica, secondo la quale è stato poi prodotto un prototipo che ne dimostra la correttezza e la fattibilità.

Tuttavia, tale progettazione si è basata su dei requisiti puramente iniziali e per nulla ufficiali. Pertanto il proseguimento del progetto richiede innanzitutto che questo sia presentato agli Organi di Ateneo competenti per ottenere la sua approvazione.

Nel caso questi decidessero di approvarlo, dovrà essere costituito un team di progetto che si occuperà, sulla loro supervisione, di raccogliere i requisiti veri e propri e di revisionare la progettazione realizzata in questa tesi, al fine di renderla coerente coi nuovi requisiti e con i dati effettivamente gestiti dal Sistema Informativo.

Il team di progetto come minimo dovrà essere formato dalle seguenti figure:

- *Capo progetto:* ha il compito di gestire e coordinare il team facendo rapporto periodico agli Organi di Ateneo sull'andamento del progetto.
- *Capo sviluppatore:* è responsabile allo sviluppo dell'applicazione, per il quale collabora con un gruppo di sviluppatori;

- *Sviluppatori:* rappresenta un gruppo di programmatori qualificati per lo sviluppo di applicazioni per *iOS*;
- *Interface designer:* collabora con gli Organi di Ateneo per la definizione di uno stile grafico dell'applicazione, sulla base del quale andrà poi a disegnare l'interfaccia grafica implementata dagli sviluppatori;
- *Advisor:* corrisponde ad una persona in possesso di conoscenze specifiche nel campo dei servizi mobili e delle applicazioni per *iOS*. Curerà le operazioni svolte dal team accertando il mantenimento di un livello di qualità adeguato;
- *Deployment e bug testing:* è un gruppo di persone responsabile per la fase finale del progetto che consiste nella messa in opera e il *testing* dell'applicazione per la ricerca di eventuali bug;
- *Sponsor di progetto:* rappresenta un membro degli Organi di Ateneo, eletto per supervisionare il progetto e assistere il capo progetto alla gestione del team. Ha inoltre il compito di ottenere il budget e firmare documenti, come ad esempio il documento dei requisiti e il documento di avvio del progetto.

Al fine di contenere le spese di conduzione, il progetto potrà essere sviluppato internamente dallo staff e dagli studenti dell'Università Ca' Foscari.

Requisiti funzionali aggiuntivi

Il progetto attualmente prevede, nella prima versione distribuibile dell'applicazione, l'implementazione di un set contenuto di funzionalità, che corrisponde a quello presentato in questo documento. Vi sono due motivazioni per questa scelta: in primo luogo un'applicazione *iOS*, essendo un'applicazione mobile, non può implementare un elevato numero di funzionalità, in quanto l'utente si troverebbe

disorientato e impossibilitato a svolgere i suoi task; in secondo luogo è preferibile poter distribuire l'applicazione il prima possibile, al fine di monitorare come viene utilizzata dagli utenti e utilizzare le metriche ottenute per stabilire quali casi d'uso sono da migliorare e quali nuove funzionalità possono essere invece introdotte nelle future *release* del software.

Nell'elenco riportato qui di seguito viene fatto riferimento ad alcune funzionalità che sono state escluse dalla progettazione iniziale, ma che potranno essere oggetto di rivalutazione nello sviluppo futuro dell'applicazione:

- *Libretto universitario:* rappresenta una versione ottimizzata e più funzionale del Libretto accessibile dall'area riservata. Oltre alla normale visualizzazione dei voti degli esami registrati, permette all'utente di verificare istantaneamente la media ottenuta per un voto non ancora registrato e il calcolo del voto finale di laurea, soddisfacendo i bisogni degli studenti che tengono particolarmente al proprio punteggio di laurea;
- *Profilo esteso degli insegnamenti*: prevede l'aggiunta di alcune funzionalità avanzate che consistono nell'iscrizione autenticata agli insegnamenti inseriti nel proprio piano di studi, la visualizzazione di una lista degli studenti iscritti, l'accesso ai materiali riservati, una gestione più avanzata e automatizzata dei task assegnati dai docenti ed infine l'integrazione di un'area di discussione specifica per ogni insegnamento, nella quale possono intervenire sia gli studenti iscritti che i docenti responsabili;
- *Bacheca appelli:* permette di iscriversi agli appelli d'esame del proprio piano di studi, attraverso l'accesso autenticato all'area riservata;
- *Indicatore previsione marea:* fornisce in tempo reale la previsione sul livello della marea di Venezia. Un'apposita icona nella Home screen di iCa'Foscari permette

all'utente di verificare a colpo d'occhio la previsione per le ventiquattrore successive. Premendo la stessa icona, l'utente può accedere direttamente alla pagina Centro Maree per ottenere maggiori informazioni;

Figura 12.1: Indicatore di previsione della marea di Venezia.

- *Ricerca federata:* rappresenta una barra di ricerca presente nella Home screen che consente di effettuare una ricerca federata tra i contenuti di tutti i sottosistemi.

Requisiti non funzionali e aspetti progettuali da approfondire

La progettazione realizzata in questo lavoro di tesi non tiene conto di alcuni requisiti non funzionali, sebbene questi siano già stati previsti, poiché questi richiedono maggiore spazio di approfondimento. A loro volta alcuni aspetti della progettazione stessa non sono stati curati a sufficienza, in quanto dipenderanno dalla gestione del progetto, una volta questo avviato.

Di seguito è riportato un elenco dei lavori di approfondimento consigliati, riservato al team di progetto:

- Supporto delle notifiche *PUSH* per il funzionamento del sottosistema Avvisi;
- Progettazione del codice per il salvataggio dell'ultimo stato di ciascun sottosistema, come richiesto nei requisiti non funzionali, basandosi

sull'archiviazione dello *stack* dei *view controller* caricati e la query di ricerca dell'utente;

- Utilizzo di un oggetto *singleton* per la gestione della persistenza, permettendo di risparmiare righe di codice ridondanti per il salvataggio e l'eliminazione di entità con *Core Data*;
- Definizione di una classe *factory* che tutti i sottosistemi devono estendere per l'implementazione di metodi in comune, che riguardano ad esempio la ricerca federata dei contenuti, l'accesso trasversale alle funzionalità, il salvataggio dell'ultimo stato e la persistenza;
- Caricamento *multi-thread* delle view con un elevato numero di contenuti, per mantenere l'applicazione sempre reattiva;
- Implementazione di classi *factory* e di metodi parametrizzati per l'eliminazione di codice ripetuto, come ad esempio la creazione di *emptyView* e indicatori di attività (vedi sottosistema Directory);
- Introduzione di una classe *singleton* per l'accesso all'*API* dei servizi UNIVE Mobile Web, al fine di separare la logica di *parsing* e la sintassi delle *API call* dall'implementazione di ogni sottosistema;
- Definizione di un modello di *testing* basato sulla metodologia di *unit testing* supportata dall'*IDE XCode*. Può essere valutato, inoltre, l'utilizzo di tecniche come *Mock Objects*[17] per la progettazione del codice di *testing*;
- Supporto al *multi-tasking* di *iOS 4*.

[17]http://www.mockobjects.com

Distribuzione dell'applicazione

Esistono due modi per distribuire un'applicazione per *iOS*: attraverso l'App Store, oppure mediante un'infrastruttura di distribuzione privata *Ad Hoc*.

La distribuzione *Ad Hoc* però ha un limite di condivisione fino a 100 dispositivi *iOS*[18]. Tale limitazione fa si che iCa'Foscari debba essere distribuita esclusivamente per mezzo dell'App Store. Ciò comporterebbe la presenza ufficiale dell'Università Ca' Foscari su un *marketplace* internazionale, con gli impegni e i benefici che ne decorrono.

Il processo di pubblicazione di un'applicazione sull'App Store non è per nulla immediato. Infatti, come testimoniano alcuni sviluppatori[19], l'applicazione può venir rifiutata più volte prima di essere approvata da Apple. Il che vuol dire che dovranno alternarsi frequenti fasi di *testing* e di scrittura di codice, per la correzione preventiva di bug che potrebbero manifestarsi in fase valutazione dell'applicazione.

Una volta approvata dallo staff di Apple, l'applicazione è disponibile sull'App Store e può essere scaricata da chiunque senza limitazioni; pertanto l'infrastruttura di servizio dovrà disporre di un adeguato numero di risorse hardware per sostenere la massa di utenti che installeranno l'applicazione. Nel caso le risorse non dovessero bastare, potrà essere valutato l'inserimento di credenziali per l'utilizzo dell'applicazione.

Ogni nuovo aggiornamento del software dovrà seguire il medesimo processo di approvazione e pubblicazione.

MOBILE WEB PER L'UNIVERSITÀ CA' FOSCARI

Come già introdotto nella prima parte di questo documento, questa tesi si è proposta di progettare un'applicazione *iOS* che possa costituire delle linee guida per la

[18]http://developer.apple.com/programs/ios/distribute.html
[19]http://www.mikeash.com/pyblog/the-iphone-development-story.html

realizzazione di un servizio web mobile che comprenda la maggior parte dei dispositivi capaci di navigare il web utilizzati dagli universitari.

La Mission del progetto si apre, infatti, con questa frase: *"Offrire una varietà di servizi studenteschi e di facoltà in ogni luogo e momento[…]"*. Con questa affermazione si intende realizzare in futuro un nuovo sistema di fornitura dei servizi web dell'Università verso i dispositivi mobili più diffusi o emergenti.

Il ruolo di iCa'Foscari in questo proposito è quello di delineare quali servizi possono essere di maggiore utilità per un utente mobile, le cui esigenze cambiano radicalmente rispetto a quando si trova ad utilizzare il proprio PC desktop o portatile. Per di più, si propone di stabilire una forma di rappresentazione grafica e di navigazione dei contenuti riproducibile su tutti i dispositivi *touch screen* che hanno una risoluzione dello schermo simile. Infine, sarà di aiuto alla definizione di un'*API* per la distribuzione dei contenuti verso tutti i dispositivi mobili.

Il concetto di web mobile

Quando si fa riferimento al *mobile web*[20] o ai siti web mobili, si sta parlando di siti web progettati specificamente per essere utilizzati su un apparecchio mobile, come un cellulare, uno smartphone oppure su un dispositivo *handheld* capace di navigare il web come un iPod Touch. Non si tratta quindi di prendere un sito web, e con esso tutte le sue funzionalità, che è stato progettato per PC desktop e connessioni internet veloci, e adattarlo allo schermo di un dispositivo tascabile. Si tratta, invece, di progettare e sviluppare siti web specificamente per questi dispositivi.

Ma allora ci si domanda, quale valore può aggiungere il *mobile web* ad una comunità universitaria rispetto a quanto già offerto dal *desktop web*?

[20]https://spaces.internet2.edu/download/attachments/4032298/Mobility-at-MIT-2009-11.pdf

Innanzitutto, sempre più persone stanno utilizzando i dispositivi mobili per accedere al web. Infatti, nel 2007 su scala mondiale gli apparecchi mobili hanno superato le vendite della controparte costituita dai computer desktop e portatili, e tutte le proiezioni indicano che questo trend è in costante aumento. Inoltre, un rapporto della *Morgan Stanley*, risalente alla fine del 2009, afferma che l'utenza internet mobile supererà quella desktop nell'arco di cinque anni[21].

In secondo luogo, vi è un numero sempre maggiore di contenuti e funzionalità ottimizzate per il web mobile, come ad esempio i servizi di ricerca, le news, le e-mail, i *social network*, ecc..

Infine, alcuni tipi di contenuti, funzionalità e interazioni sono particolarmente utili e appropriati per l'uso *on-the-go*, come ad esempio la ricerca di un contatto, di un edificio, di una lezione o di un evento.

Infatti, vi sono delle differenze sostanziali tra la navigazione di un sito web da una postazione desktop e quella da un dispositivo mobile. Nel primo caso ci si aspetta che l'utente dedichi particolare attenzione all'atto di navigare il web. Al contrario, un utente mobile può trovarsi allo stesso tempo in condizioni diverse, ad esempio può star camminando, parlando o ascoltando, oppure può essere intento a svolgere altre attività che necessitano della sua attenzione. L'utilizzo del web mobile in queste situazioni rivestirà quindi un ruolo secondario, temporaneo e facilmente interrompibile. Non per niente, un utente desktop è generalmente propenso a passare diversi minuti su un sito web; al contrario un utente mobile solitamente non rimane più di qualche secondo su un sito web mobile comparabile.

[21]http://www.morganstanley.com/institutional/techresearch/mobile_internet_report122009.html

Pertanto, la progettazione e la struttura di un sito web mobile o di una *web application* deve riflettere l'impegno e le limitazioni imposte proprio dal fatto di essere mobile. Ciò significa che, i contenuti devono poter essere consumati velocemente e le funzionalità devono essere poche e scelte con cura. Allo stesso modo, la navigazione deve essere semplice e conforme col tipo di input e la dimensione del dispositivo utilizzato.

La necessità di offrire un servizio web mobile, sta nel fatto che le persone, soprattutto quelle appartenenti al mondo universitario, stanno facendo un uso sempre maggiore delle tecnologie mobili, dalle quali hanno aspettative sempre maggiori.

Il progetto UNIVE Mobile Web

Il progetto UNIVE Mobile Web si propone quindi di costituire un nuovo servizio web per l'Università Ca' Foscari, applicando il concetto di *mobile web* appena introdotto.

Vi sono quattro fasi in cui è articolato il progetto:

1. Definizione dei contenuti e delle funzionalità (già parte integrante di questo lavoro di tesi);
2. Progettazione di un'architettura di generazione dei contenuti e di un'interfaccia comune per i dispositivi mobili;
3. Ricerca e consolidamento delle categorie di dispositivi più diffuse nell'ambiente universitario, design di un sito web mobile compatibile e progettazione di un sistema di *device detection*;
4. Progettazione e sviluppo di applicazioni native per alcune piattaforme specifiche (oggetto di questa tesi è infatti la progettazione e lo sviluppo di un'applicazione nativa per *iOS*).

La struttura dell'architettura risultante rappresenta un'estensione di quella già presentata nella seconda parte di questo documento, in quanto ai dispositivi *iOS* si accostano altre piattaforme mobili, come dimostra la figura seguente:

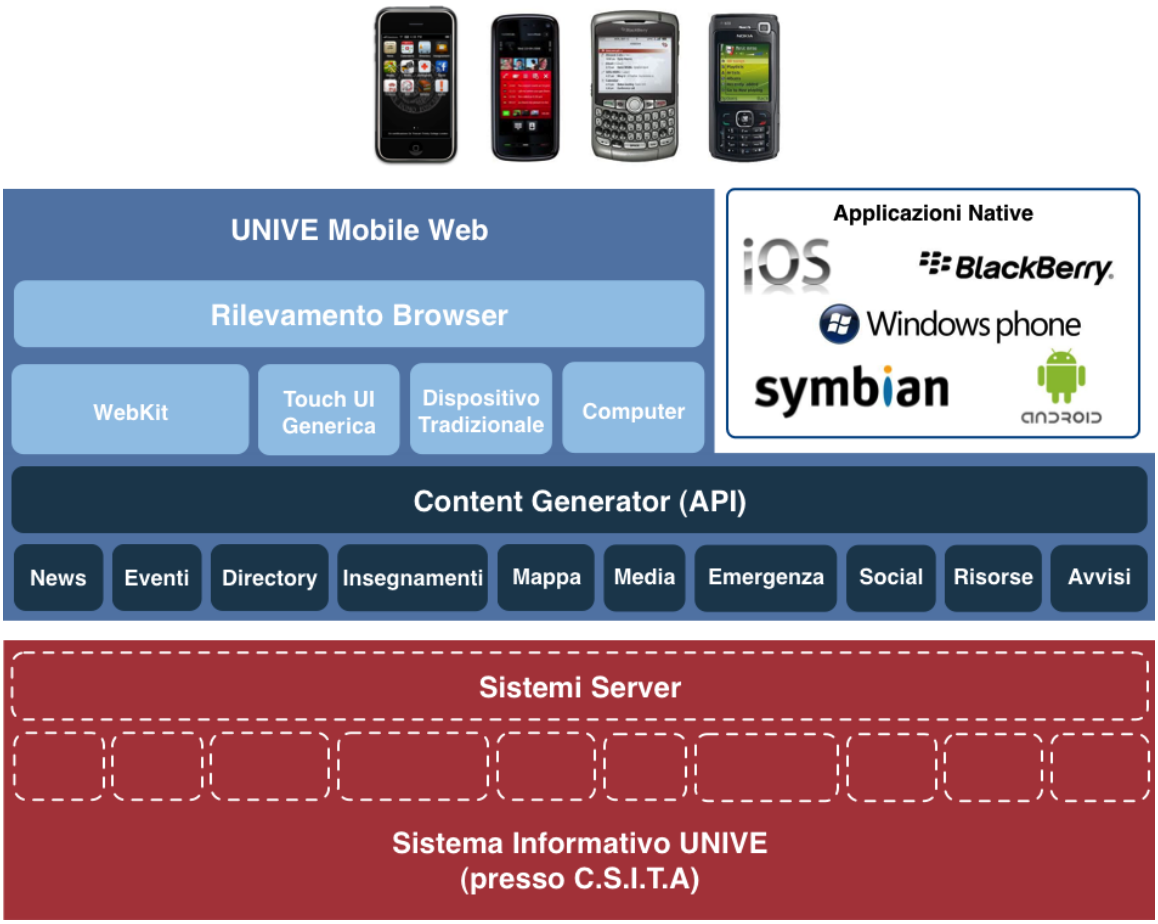

Figura 12.2: Architettura UNIVE Mobile Web.

Nella figura si possono distinguere tre categorie di dispositivi mobili che faranno uso del futuro sito web mobile:

- **Dispositivi tradizionali**: sono degli apparecchi mobili, come ad esempio il Nokia N70, che dispongono di un browser web dalle funzionalità limitate e di una

tastiera fisica come dispositivo di input e puntamento. Rappresentano la categoria maggiormente consolidata;

- **Dispositivi touchscreen**: sono dispositivi caratterizzati da ampi schermi sensibili al tocco, come ad esempio il Nokia 5800 e alcuni modelli di BlackBerry. Questa categoria richiede un sito web ottimizzato per una navigazione di tipo *touch-driven*, ma allo stesso tempo adatto alle capacità limitate del proprio browser;
- **Dispositivi Webkit**: a questa categoria appartengono tutti i dispositivi con *display touchscreen* che dispongono di un browser web avanzato basato su *Webkit*, come ad esempio i dispositivi *iOS* e *Android*.

Le categorie appena elencate sono state definite esclusivamente sulla base dell'offerta corrente del mercato delle tecnologie mobili; tenendo però in considerazione i dispositivi *legacy* che hanno dominato lo scenario del web mobile negli scorsi anni.

Poiché in queste categorie rientra un vasto numero di modelli diversi di apparecchi mobili, sarà di estrema complessità e importanza, fin dall'inizio della progettazione e attraverso tutta la fase di sviluppo, prevedere un piano di *testing* frequente ed esaustivo, al fine di comprendere il maggior numero di dispositivi possibile.

CONCLUSIONI

Nell'ultimo capitolo di questa tesi è stato definito l'iter necessario al completamento e alla distribuzione di iCa'Foscari; aprendo la strada ad un nuovo progetto dalle dimensioni maggiori che si propone di realizzare un servizio web mobile per l'Università Ca' Foscari, in cui coesistono applicazioni native come iCa'Foscari e un sito web mobile per altre categorie di dispositivi capaci di navigare il web.

CONCLUSIONI

Questa tesi descrive la progettazione e lo sviluppo di un'applicazione appositamente studiata e ottimizzata per la fornitura di alcuni servizi, già disponibili dal sito web dell'Università, agli utenti mobili che utilizzano i dispositivi *iOS*.

L'applicazione nativa progettata include numerose funzionalità, che consentono ad esempio di cercare il contatto di un docente, di visualizzare il profilo di un corso, di consultare il calendario degli eventi, di localizzare una sede sulla mappa e di ricevere avvisi in tempo reale. In generale l'applicazione permette ad un utente in movimento di ottenere delle informazioni, in maniera rapida ed efficace, nel momento stesso in cui ne ha bisogno.

Il lavoro svolto in questa tesi, ha permesso per di più di identificare e apprendere le conoscenze teoriche e le migliori pratiche coinvolte nella progettazione e nell'implementazione di un'applicazione sulla piattaforma *iOS*. Oltre alla progettazione dell'architettura dell'applicativo, svolta secondo i criteri dell'Ingegneria del Software, è stato infatti prodotto il prototipo di una parte delle funzionalità definite nei requisiti, al fine di offrire un'anticipazione del futuro prodotto finale di questo progetto.

La tesi, infine, introduce le linee guida e i concetti fondamentali per la futura messa in opera di un servizio web mobile per l'Università Ca' Foscari, il quale prevede lo sviluppo di applicazioni native per alcuni dispositivi più avanzati, proprio come iCa'Foscari per iPhone, e la realizzazione di un sito web mobile per quelli tradizionali. Condividendo però tra tutte le piattaforme le stesse caratteristiche funzionali di base, opportunamente confezionate e ottimizzate per offrire la migliore esperienza di fruizione dei contenuti possibile su ogni dispositivo.

Glossario

Questo glossario non comprende riferimenti all'*SDK* di Apple, la cui documentazione può essere consultata direttamente dalla *Reference Library* ufficiale:

http://developer.apple.com/library/ios/navigation/

Agile: si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente, ottenendo in tal modo una elevata reattività alle sue richieste.

API: è un insieme di regole e specifiche che un programma deve seguire per poter accedere e usufruire alle risorse fornite da un altro particolare programma che le implementa.

badge: notifica degli eventi dell'appliccazione.

bundle: è la cartella dove risiedono tutti i buld dell'applicazione iOS.

caching: è una memorizzazione dei dati in modo trasparente in modo che le future richieste di dati possano essere servite in modo più veloce.

CSS: è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML. Le regole per comporre il CSS sono contenute in un insieme di direttive (*Recommendations*) emanate a partire dal 1996 dal W3C.

debugging: è un'attività che consiste nella individuazione della porzione di software affetta da errore (bug) rilevati nei software a seguito dell'utilizzo del programma.

design pattern: una soluzione progettuale generale a un problema ricorrente.

embedded: si identificano genericamente tutti quei sistemi elettronici di elaborazione a microprocessore progettati appositamente per una determinata applicazione.

factory: è uno dei design pattern fondamentali, definiti originariamente dalla gang of four che fornisce un metodo per istanziare un oggetto senza sapere a priori la sua esatta classe.

feed RSS: è un'unità di informazioni formattata secondo specifiche stabilite precedentemente. Ciò per rendere interoperabile ed interscambiabile il contenuto fra le diverse applicazioni o piattaforme..

firmware: è un programma, inteso come sequenza di istruzioni, integrato direttamente in un componente elettronico nel senso più vasto del termine.

garbage collection: si intende una modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, liberano le porzioni di memoria che non dovranno più essere successivamente utilizzate dalle applicazioni.

hard coding: si riferisce alla pratica di sviluppo software di inserimento dati di input o di configurazione direttamente nel codice sorgente di un programma eseguibile.

high water mark: indica il numero di oggetti contenuti in un pool di memoria.

iOS: è il sistema operativo sviluppato da Apple per iPhone, iPod touch e iPad. Come Mac OS X è una derivazione di FreeBSD, usa un kernel Mach e Darwin.

ISO: è la più importante organizzazione a livello mondiale per la definizione di norme tecniche.

jailbreak: è il processo che permette ai possessori di iPhone, iPod touch e iPad di modificare i file di sistema originali di Apple, di poter accedere alle cartelle di sistema del proprio dispositivo e di installare meccanismi di distribuzione di applicazioni e pacchetti alternativi a quello ufficiale dell'App Store.

look and feel: descrive le caratteristiche percepite dall'utente di una interfaccia grafica, sia in termini di apparenza visiva (il *look*) che di modalità di interazione (il *feel*).

memory leak: è un particolare tipo di consumo non voluto di memoria dovuto alla mancata deallocazione dalla memoria di variabili/dati non più utilizzati da parte dei processi.

mobile application: sono applicazioni per piccoli dispositivi a bassa potenza, come PDA o telefoni cellulari.

mobile edutainment (mobile learning): forma di entertainment educativo svolto su dispositivo mobile.

Model View Controller: è un pattern architetturale molto diffuso nello sviluppo di interfacce grafiche di sistemi software object-oriented.

multi-tasking: un sistema operativo che permette di eseguire più programmi contemporaneamente.

OAuth: è un protocollo aperto, per permettere l'autorizzazione API di sicurezza in un metodo standard e semplice per applicazioni portatili, per pc fisso e per il web.

overhead: serve per definire le risorse accessorie, richieste in sovrappiù rispetto a quelle strettamente necessarie, per ottenere un determinato scopo in seguito all'introduzione di un metodo o di un processo più evoluto o più generale.

parser: è un programma atto ad analizzare uno stream continuo in input (letto per esempio da un file o una tastiera) in modo da determinare la sua struttura grammaticale grazie ad una data grammatica formale.

podcast: è un file (generalmente audio o video) in internet e messo a disposizione di chiunque si abboni ad una trasmissione periodica.

POI: sono dei punti specifici che risultano essere utili e interessanti.

pool: è un insieme di risorse che possono essere riutilizzate.

porting: è un *adattamento* o una *modifica* del software, volto a consentirne l'uso in un ambiente di esecuzione diverso da quello originale.

Post-PC device: sono gli apparecchi che andranno a sostituire computer desktop e laptop permettendoci di avere sempre con noi i nostri dati e di connetterci a internet da qualunque punto del pianeta.

raster: è una tecnica utilizzata in computer grafica per descrivere un'immagine.

release: è una specifica versione di un software resa disponibile ai suoi utenti finali.

Responder Chain:.

responsiveness: la capacità specifica di un'unità funzionale per completare i compiti assegnati all'interno di un dato periodo di tempo.

runtime: indica il momento in cui un programma per computer viene eseguito, in contrapposizione ad altre fasi del ciclo di vita del software.

screen reader: è un'applicazione software che identifica ed interpreta il testo mostrato sullo schermo di un computer, presentandolo ad un utente affetto da handicap visivo tramite sintesi vocale o attraverso un display braille.

Scrolling Toolbar: tipo di toolbar a scorrimento orizzontale che estende la toolbar nativa di iOS.

SDK: è un insieme di strumenti per lo sviluppo e la documentazione di software.

singleton: è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Smalltalk: è un linguaggio di programmazione orientato agli oggetti con gestione dinamica dei tipi e con un paradigma di programmazione riflessivo.

social media: è un termine generico che indica tecnologie e pratiche online che le persone adottano per condividere contenuti testuali, immagini, video e audio.

Social Network: consiste di un qualsiasi gruppo di persone connesse tra loro da diversi legami sociali, che vanno dalla conoscenza casuale, ai rapporti di lavoro, ai vincoli familiari.

Tab Bar: barra delle schede, utile nelle situazioni in cui si desidera fornire prospettive diverse dello stesso set di dati o in situazioni dove si vuole organizzare l'applicazione lungo le linee funzionali.

UML: è un linguaggio di modellazione e specifica basato sul paradigma object-oriented.

Universal Access: Essa descrive la capacità di fornire capacità di calcolo per i ciechi, sordi e portatori di handicap.

web services: è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete.

WebKit: è un *framework* per applicazioni disponibile come aggiornamento per Mac OS X a partire dalla versione 10.2.7 che permette a sviluppatori terzi di includere con facilità nelle loro applicazioni molte delle funzioni proprie di Safari.

wireframe: indica un tipo di rappresentazione grafica da computer di oggetti tridimensionali, detta anche *vettoriale*.

wrapper: è un *design pattern* utilizzato in informatica nella programmazione orientata agli oggetti.

Riferimenti

[Campbell, W. G. 1990. Form and Style in Thesis Writing, a Manual of Style. Chicago: The University of Chicago Press.

Turabian, K. L. 1987. A Manual for Writers of Term Papers, Theses, and Dissertations. 5th ed. Chicago: The University of Chicago Press.]