Università degli Studi di Pisa

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica Vecchio Ordinamento

Mabon:
l'opensource e il web al servizio dei beni culturali

Candidato

Massimiliano della Rovere

Relatori

Chiar.mo Prof. Andrea Domenici

Chiar.mo Prof. Giovanni Stea

Chiar.ma Dott.ssa Clara Baracchini

Chiar.mo Umberto Parrini

Anno accademico 2007/2008

a Mamma, Papà, i nonni e Mariateresa!

La versione più aggiornata di questo file è disponibile all'indirizzo

`http://www.massimilianodellarovere.eu/mabon/`

Indice

I Introduzione **1**

1 Premessa **3**

1.1 Cosa è Mabon 3

1.2 Open source e l'aderenza agli standard 4

1.2.1 Motivazioni per la scelta di aderire agli standard 5

1.2.2 Motivazioni per la scelta di strumenti opensource 6

1.2.3 Strumenti usati 6

1.3 Flessibilità e adattabilità alle esigenze dell'utente e al dispositivo di lettura 8

1.4 Convenzioni in uso e traduzioni effettuate 8

2 Cenni su Zope **11**

2.1 Python 11

2.2 Zope 11

2.3 Flusso delle informazioni 14

2.4 Organizzazione dei componenti 15

2.4.1 Generale 15

2.4.2 Interfacce 15

2.4.3 Componenti 17

2.4.4 Componenti di contenuto 17

2.4.5 Oggetto 17

2.4.6 Adattatori 18
2.4.7 Viste 18
2.4.8 L'applicazione utente 19
2.4.9 Utilità 20
2.4.10 Registro dei componenti 20
2.4.11 Sicurezza, permessi e ruoli 21
2.4.12 Eventi 21
2.4.13 La skin o tema 22
2.4.14 ZPT, i modelli di pagina e il linguaggio TAL 23
2.5 Il paradigma Modello Vista Controllore 24

3 L'analisi dello scenario 27

3.1 Il problema della digitalizzazione 27
3.1.1 La necessità di un archivio informatizzato 27
3.1.2 Struttura della scheda 28
3.1.2.1 Informazioni catalografiche 28
3.1.2.2 Informazioni geografiche (sul soggetto rappresentato nel materiale fotografico) 29
3.1.2.3 Informazioni sull'autore e sul soggetto 30
3.1.2.4 Informazioni tecniche sul supporto fisico 30
3.1.3 Metodo di archiviazione 31
3.2 Lo stato della scheda 31
3.2.1 Le azioni effettuabili su una scheda 34
3.3 Utenti 35
3.4 Internazionalizzazione 36

II La struttura dell'applicazione 37

4 Quadro generale 39

4.1 Organizzazione dei file e struttura gerarchica 39

5 I componenti pricipali e secondari 45

5.1 I componenti secondari 45

5.1.1 Contenitori 45

5.1.1.1 La relazione di contenimento in Zope3 45

5.1.1.2 I contenitori di Mabon 46

5.1.1.3 Le classi `IContenitore*` 46

5.1.1.4 Le classi `Contenitore*` 46

5.1.1.5 L'adattatore `SincronizzaNome*` 49

5.1.2 L'oggetto etichetta 50

5.1.2.1 Generale 50

5.1.2.2 La classe `IEtichetta` 50

5.1.2.3 La classe `Etichetta` 50

5.1.3 I vocabolari 51

5.1.3.1 Generale 51

5.1.3.2 Il vocabolario Ruoli 51

5.1.3.3 Il vocabolario Fondi 52

5.1.3.4 Il vocabolario Supporti 52

5.1.3.5 Il vocabolario Etichette 52

5.2 I componenti principali 52

5.2.1 Introduzione 52

5.2.2 L'oggetto sito Mabon 52

5.2.2.1 Generale 52

5.2.2.2 La classe `ISitoMabon` 53

5.2.2.3 La classe `SitoMabon` 53

5.2.3 L'oggetto EventoNuovoSitoMabon 54

5.2.3.1 Generale 54

5.2.3.2 La classe IEventoNuovoSitoMabon 54

5.2.3.3 La classe EventoNuovoSitoMabon 55
5.2.4 L'oggetto scheda 55
5.2.4.1 Generale 55
5.2.4.2 La classe `IScheda` 55
5.2.4.3 La classe `Scheda` 57
5.2.5 L'oggetto utente 58
5.2.5.1 Generale 58
5.2.5.2 La classe IUtente 58
5.2.5.3 La classe `Utente` 59

6 Altri componenti **61**

6.1 Gestori di eventi, adattatori, viste, visualizzatori e generatori . . . 61
6.1.1 Generale 61
6.1.2 I gestori di eventi 61
6.1.2.1 I gestori di aggiunta di utilità 62
6.1.2.2 La funzione `autenticazione_e_principali` . . . 62
6.1.2.3 La funzione `cookie_e_sessioni` 62
6.1.2.4 La funzione `catalogo_e_indici` 64
6.1.2.5 I gestori degli eventi delle istanze di scheda e utente 66
6.1.2.6 La funzione `generaTitoloScheda` 66
6.1.2.7 La funzione `aggiornaTitoloScheda` 67
6.1.2.8 La funzione `generaTitoloUtente` 67
6.1.2.9 La funzione `aggiornaTitoloUtente` 67
6.1.2.10 La funzione `creazioneAreaUtente_e_principale` 67
6.1.2.11 La funzione `cancellazioneAreaUtente_e_principale` 68
6.1.2.12 Le funzioni `aggiornaDCda*` 68
6.1.3 Adattatori 70
6.1.3.1 Generale 70
6.1.3.2 Le classi `SincronizzaNome*` 70

6.1.3.3 Le classi `Dimensione*` 71
6.1.3.4 La classe `TestoSchedaPerRicerca` 71
6.1.3.5 L'adattatore con nome mabon.moduloscheda" . . 72
6.1.3.6 L'adattatore con nome "mabon.moduloutente" . . 72
6.1.4 Viste 72
6.1.4.1 Generale 72
6.1.4.2 La classe `VisualizzaAzioni` 73
6.1.4.3 La classe `VisualizzaScheda` 73
6.1.4.4 La classe `ModuloModificaScheda` 74
6.1.4.5 La classe `ModuloAggiungiScheda` 76
6.1.4.6 La classe `VisualizzaUtente` 78
6.1.4.7 La classe `ModuloModificaUtente` 79
6.1.4.8 La classe `ModuloAggiungiUtente` 81
6.1.4.9 La classe `PaginaDiRicerca` 82
6.1.4.10 La classe `ModuloRicercaAvanzata` 84
6.1.4.11 La classe `PaginaDiRicercaAvanzata` 86
6.1.4.12 La classe `ModificaStatoScheda` 87
6.1.4.13 La classe `ElencaImmagini` 88
6.1.4.14 La classe `ImportaImmagini` 89
6.1.4.15 La classe Azioni 91
6.1.5 I visualizzatori 92
6.1.5.1 Generale 92
6.1.5.2 Il visualizzatore di ricerca 92
6.1.5.3 La classe `OpzioniModificaStato` e il relativo visualizzatore 93
6.1.6 I generatori 96
6.1.6.1 Generale 96
6.1.6.2 La classe `generatoreScheda` 96

6.1.6.3 La classe `generatoreUtente` 96
6.1.6.4 La funzione `vocabolarioRuoli` 97
6.1.6.5 La funzione `vocabolarioFondi` 97
6.1.6.6 La funzione `vocabolarioImmaginiNuove` 97
6.1.6.7 La funzione `vocabolarioImmaginiImportate` . . 97
6.1.6.8 La funzione `vocabolarioSupporti` 97
6.1.6.9 La funzione `vocabolarioEtichette` 98
6.2 Codice e componenti client-side 98
6.2.1 La visualizzazione dell'immagine associata alla scheda . . . 98

7 Localizzazione dei componenti nei files 101
7.1 Generale 101

8 Relazioni e vincoli fra le classi e le istanze 109
8.1 Generale 109
8.2 Le relazioni imperniate su `Utente` 109
8.3 Le relazioni imperniate su `Scheda` 110

9 Interazioni fra i componenti 115
9.1 Generale 115
9.2 Le interazioni asincrone 115
9.2.1 I gestori di eventi: SitoMabon 115
9.2.2 I gestori di eventi: Schede 117
9.2.3 I gestori di eventi: Utenti 118
9.2.4 La generazione dell'identificatore intero per un istanza contenuta 119
9.2.5 La generazione dell'indice per gli oggetti scheda 119
9.2.6 I componenti di aggiunta e modifica 120
9.2.6.1 Il componente di aggiunta 120
9.2.6.2 Il componente di modifica 123

9.2.7 La ricerca delle schede 123
9.2.7.1 La ricerca semplice 123
9.2.7.2 La ricerca avanzata 124
9.2.8 L'importazione delle immagini 126
9.3 Le interazioni sincrone 126
9.3.1 Generale 126
9.3.2 L'autenticazione dell'utente 126
9.3.3 La modifica dello stato della scheda 128
9.3.4 Il calcolo della dimensione di un oggetto 128
9.3.5 La visualizzazione di un oggetto 128

III Installazione ed uso del programma 131

10 Installazione 133

10.1 Requisiti 133
10.2 Installazione 134
10.3 Aggiunta di una nuova traduzione dell'interfaccia 134

11 Uso 135

11.1 Generale 135
11.2 Opzioni per l'Amministratore 135
11.2.1 Generale 135
11.2.2 Opzioni e casi d'uso 135
11.3 Effettuare una ricerca nel database 136
11.4 Opzioni per il Ricercatore 137
11.4.1 Generale 137
11.4.2 Opzioni e casi d'uso 137
11.5 Opzioni per lo Schedatore 138
11.5.1 Generale 138

11.5.2 Opzioni e casi d'uso 138
11.6 Opzioni per lo Scansionatore 140
11.6.1 Generale 140
11.6.2 Opzioni e casi d'uso 141
11.7 Opzioni per il Validatore 142
11.7.1 Generale 142
11.7.2 Opzioni e casi d'uso 142

IV Considerazioni finali e Ringraziamenti 145

12 La documentazione di Zope 147

13 Ringraziamenti 149

A Elenchi 151

B Bibliografia 161

Parte I

Introduzione

Capitolo 1

Premessa

1.1 Cosa è Mabon

Mabon è un'applicazione web–based per la catalogazione e consultazione del materiale fotografico, creato secondo le esigenze e parametri della Soprintendenza per i Beni Architettonici e Paesaggio e per il Patrimonio Storico-Artistico Etnoantropologico (BAPPSAE)[1] per le Province di Pisa e Livorno, dove ho svolto il servizio civile.

La progettazione di Mabon II, che costituisce questa tesi, nasce dalla volontà di applicare le conoscenze maturate durante il mio corso di studi in Ingegneria Informatica – Ingegneria del Software, Basi di Dati, Progettazione Multimediale e Sicurezza delle Reti in particolare – al programma Mabon I, realizzato in PHP[2]/MySQL[3] e attualmente in uso nella Soprintendenza[4], concepito più tramite l'uso di una tecnica di *Extreme Programming*[5] in coppia con gli utenti di Mabon I che con metodologie più razionali e da me preferite.

Mabon II, di seguito Mabon, prevede che l'amministratore del programma

[1] http://www.sbappsae-pi.beniculturali.it/
[2] http://www.php.net/
[3] http://it.wikipedia.org/wiki/MySQL
[4] Mabon I è accessibile solo dalla rete interna della Soprintendenza.
[5] http://it.wikipedia.org/wiki/Extreme_Programming

crei o cancelli gli account dei vari utenti secondo l'iter burocratico e le necessità dei responsabili, dato che la attribuzione dei livelli di privilegio degli utenti è in corrispondenza biunivoca con le mansioni svolte. Tramite l'interfaccia web offerta dal programma è possibile, in funzione del proprio livello di privilegio, creare bozze di schede, modificarle, cancellarle, e promuovere tali bozze in documenti effettivi, importare una rappresentazione digitale del materiale fotografico del quale la scheda è una descrizione e collegarla alla scheda, gestire gli account utenti e, ovviamente, effettuare delle ricerche.

Tutte le operazioni sono effettuabili tramite web, quindi solo con l'ausilio di un browser, tranne una: la copia delle immagini scansionate in un'apposita cartella del programma. Date le dimensioni delle immagini utilizzate in Soprintendenza, ho preferito che la copia dei files avvenga in una cartella esportata pubblicamente nella rete locale (la sicurezza e i permessi di tale cartella sono di responsabilità dell'amministratore di rete e delle sistema operativo delle macchine coinvolte) e quindi l'operazione di copia effettuata con strumenti offerti dal sistema operativo, mentre il collegamento tra l'immagine e la scheda viene effettuato tramite Mabon.

1.2 Open source e l'aderenza agli standard

Quando si presentò il momento di scrivere questa tesi, il primo interrogativo che mi posi riguardò gli strumenti da utilizzare e se, data la sua natura di applicazione web, utilizzare o meno delle estensioni proprietarie o delle ottimizzazioni per una specifica piattaforma.

Data l'eterogeneità dell'hardware e software con i quali mi sono confrontato durante il servizio civile, ho preferito utilizzare strumenti che utilizzano e rispettano gli standard suggeriti dal consorzio W3C[6] e che spesso richiedono anche una limitata potenza di calcolo. Mabon I infatti è stato usato anche da PC con

[6]http://www.w3.org/TR/#Recommendations

sistema operativo FreeBSD[7] o Linux[8] e browser Opera[9] e ELinks[10] (con supporto grafico) e processori a 266÷400MHz...

1.2.1 Motivazioni per la scelta di aderire agli standard

Nella mia esperienza quotidiana riscontro spesso problemi funzionali ed estetici nella navigazione in Internet a causa dei frequenti errori di sintassi o estensioni e ottimizzazioni per uno specifico browser presenti nelle pagine web. Pur utilizzando browsers quali Opera, Konqueror[11] e Firefox[12], perfettamente adempienti alle specifiche di (X)HTML[13] e CSS[14] promulgate dal consorzio W3[15], spesso la visualizzazione della pagina è palesemente errata con elementi fuori posto o mal dimensionati o, peggio, il codice in ECMAscript[16] è mal scritto o non correttamente integrato col codice (X)HTML o, più comunemente, il codice (X)HTML contiene errori.

Ho, quindi, cercato uno strumento per l'implementazione di Mabon che, pur garantendomi la massima flessibilità e libertà in fase di programmazione, produca codice compatibile con il formato XHTML e CSS. In quest'ambito la scelta è ricaduta su Zope 3[17] poiché le pagine web prodotte vengono renderizzate[18] a partire da un file XML[19], e superano quindi i tests dei validatori[20] offerti dal consorzio W3C.

[7] http://www.freebsd.org/it/
[8] http://www.linux.org/
[9] http://www.opera.com/
[10] http://elinks.cz/
[11] http://www.konqueror.org/
[12] http://www.mozilla-europe.org/it/firefox/
[13] http://www.w3.org/TR/xhtml11/
[14] http://www.w3.org/Style/CSS/
[15] http://www.w3.org/
[16] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[17] http://it.wikipedia.org/wiki/Zope_3 e http://www.zope.org/Products/Zope3
[18] Cioè il contenuto XHTML viene generato a partire da codice sorgente XML.
[19] http://it.wikipedia.org/wiki/XML e http://www.w3.org/XML/
[20] http://validator.w3.org/ e http://www.w3.org/QA/Tools/
Il browser Opera integra un'opzione per l'invio della pagina ai suddetti validatori.

1.2.2 Motivazioni per la scelta di strumenti opensource

Data la intrinseca esposizione del programma al web, ho preferito utilizzare strumenti e componenti che vengono sviluppati e aggiornati da una comunità estremamente attenta alla ricerca di falle di sicurezza ed errori nel codice ma, soprattutto, reattiva in caso di scoperta e successiva applicazione delle correzioni. Ero sostanzialmente alla ricerca di un ambiente di sviluppo OpenSource[21] e Zope 3, distribuito sotto licenza Zope Public License[22], lo è.

Per loro stessa natura, i programmi closed source richiedono che sia la stessa ditta produttrice a scoprire, opzionalmente rendere noti e, infine, correggere eventuali errori e falle di sicurezza; data la sensibilità e l'importanza dei dati gestiti da Mabon, questo è un rischio che non volevo correre.

Ulteriori motivazioni che mi hanno guidato nella scelta sono state anche quelle dei costi degli strumenti software di sviluppo – tutti quelli da me utilizzati sono gratuiti – e la loro frequente e continua evoluzione.

Ulteriori tesi a supporto dell'opensource possono essere trovate sul web e nel manifesto del progetto GNU[23].

1.2.3 Strumenti usati

Da quanto esposto precedentemente, risulta evidente che già il sistema operativo non può essere closed source. Nel corso dello sviluppo della tesi sono stati impiegati – alternandosi – Linux (distribuzione Kubuntu[24]) e FreeBSD.

L'ambiente di sviluppo e lavoro per tutto lo sviluppo della tesi è stato KDE, nelle due successive versioni 3 e 4. KDE[25] è un ambiente per l'utilizzo e la gestione del desktop con un supporto integrato e trasparente alle risorse di rete, disponibile per tutti i sistemi Unix[26] e compatibili. Per questa sua caratteristica è, quindi,

[21] http://it.wikipedia.org/wiki/Open_source
[22] http://en.wikipedia.org/wiki/Zope_Public_License
[23] http://it.wikipedia.org/wiki/GNU
[24] http://it.wikipedia.org/wiki/Kubuntu e http://www.kubuntu.org/
[25] http://it.wikipedia.org/wiki/KDE e http://www.kde.org/
[26] http://it.wikipedia.org/wiki/Unix

esistente sia per Linux sia per FreeBSD.

I diagrammi UML[27] che illustrano il funzionamento e le interazioni tra i vari moduli e oggetti scritti per la tesi, e le interazioni di questi con le componenti di Zope sono stati creati con il programma gratuito UMLet[28], un editor sia visuale sia testuale multipiattaforma scritto in Java[29].

I ritocchi alle immagini, le icone, gli screenshot e tutte le parti grafiche sono stati creati o eseguiti con il programma di grafica Gimp[30].

L'editor per effettuare la stesura della tesi è invece Lyx[31], il popolare document processor con impostazione "What You See Is What You Mean"[32] nella sua incarnazione lyx–qt[33] per integrarsi meglio con KDE, affiancato da Kbibtex[34] per la gestione della bibliografia.

Zope è una piattaforma – nonché un ambiente di sviluppo web – che permette a sviluppatori con differenti competenze di costruire applicazioni e componenti web. Zope 3(.4) è la nuova e più aggiornata edizione di Zope e nasce come una riscrittura da zero di un framework a moduli sfruttando tutta l'esperienza maturata in anni di utilizzo di Zope 2[35] e Zope CMF[36]. Zope 3 assolve contemporaneamente a due funzioni:

- l'essere una piattaforma per la creazione di sistemi per la gestione di contenuto, fruibili, gestibili e utilizzabili in particolare attraverso il web (questo scopo, infatti, viene raggiunto dal modulo web server di Zope, che ne mantiene tutte le funzionalità) e

[27] http://it.wikipedia.org/wiki/Unified_Modeling_Language
[28] http://www.umlet.com/
[29] http://it.wikipedia.org/wiki/Java_(linguaggio)
[30] http://www.gimp.org/
[31] http://it.wikipedia.org/wiki/LyX e http://www.lyx.org/
[32] http://it.wikipedia.org/wiki/WYSIWYM e http://it.wikipedia.org/wiki/WYSIWYG
[33] http://www.qt-apps.org/content/show.php/LyX?content=57243 e http://it.wikipedia.org/wiki/Qt_(toolkit)
[34] http://www.unix-ag.uni-kl.de/~fischer/kbibtex/
[35] http://wiki.zope.org/zope2/Zope2Wiki
[36] http://plone.org/documentation/faq/what-is-cmf

- l'implementare un'architettura basata su componenti che consente la creazione di applicazioni.

1.3 Flessibilità e adattabilità alle esigenze dell'utente e al dispositivo di lettura

La creazione di pagine che utilizzano solo codice XHTML e CSS valido rende possibile l'utilizzo di programmi che consentano o facilitino la fruizione dei contenuti anche a persone con disabilità o una più semplice conversione per dispositivi di input/ouput diversi da quelli tradizionali, quali terminali braille o ancora la possibilità di visualizzare la pagina così come viene percepita da chi è affetto da uno dei vari tipi di daltonismo.

1.4 Convenzioni in uso e traduzioni effettuate

Per comodità di redazione del testo e per un maggiore utilizzo della lingua italiana ove possibile, ho deciso di tradurre alcuni termini riguardanti i componenti di Zope3 e di utilizzare in maniera particolare le lettere maiuscole.

Quando nel testo si incontra una parola con lettera maiuscola, vi è un riferimento al codice del programma, mentre con lettera minuscola si intende una parola comune: ad esempio "amministratore" indica in senso lato una persona con compiti di amministrazione del sito, mentre "Amministratore" si riferisce ad un utente che dispone di un account con privilegi di ammistratore.

La traduzione si riferisce ai seguenti termini:

- adapter = adattattore
- component = componente
- component registry = registro dei componenti

- content component = componenti di contenuto
- factory = generatore
- interface = interfaccia
- packages = pacchetti
- site manager = gestore del sito
- skin = tema
- traversing = attraversare o attraversamento
- utility = utilità
- view = vista
- viewlet = visualizzatore o area di visualizzazione
- viewlet manager = raggruppatore di visualizzatori

Capitolo 2

Cenni su Zope

2.1 Python

L'intero framework Zope è implementato in Python[1] e, sempre tramite il linguaggio di Guido van Rossum[2], è estendibile e programmabile. I moduli di Mabon sono infatti scritti in Python, con le dovute eccezioni costituite dai file di configurazione e le pagine web scritti invece in XML.

2.2 Zope

Come illustrato in [Wei08] Zope è una collezione di software gratuiti, rilasciato nel 2004, e continuamente sviluppato in sinergia dalla Zope Corporation[3] e da una vasta comunità di programmatori, che può essere usato in toto o anche modularmente, per gestire la complessità di progettare e far lavorare insieme svariati componenti, e pubblicare in sicurezza degli oggetti[4] sul web o altri sistemi, pur mantendo semplice la possibilità di effetuare dei Controlli sulla Qualità del software scritto.

[1] Sebbene al momento della scrittura della tesi sia disponibile Python 3.0.1, la versione di Python utilizzata di Zope 3.3 è la 2.5 - http://www.python.org/ e http://it.wikipedia.org/wiki/Python

[2] http://www.python.org/~guido/

[3] Il cui CEO è Jim Fulton.

[4] Zope è infatti l'acronimo di "Z Object Publishing Environment".

Zope è un software multipiattaforma (essendo basato su Python che a sua volta lo è) in grado di girare su sistemi Unix, Linux, Mac OS X[5] e Windows[6]. Può cooperare con webservers e database esterni, anche se ne implementa di propri.

Parti di Zope sono la Architettura a Componenti (illustrata nella prossima sezione), un database ad oggetti e non relazionale – lo ZOpe DataBase[7], una libreria per la gestione e la creazione e l'utilizzo di modelli HTML/XML, librerie per la generazione e la validazione di form di input[8] in XHTML, supporto per l'internazionalizzazione del software e librerie e sistemi per la gestione della sicurezza e dei permessi flessibili, modulari ed espandibili dal programmatore, un motore di ricerca ed altro.

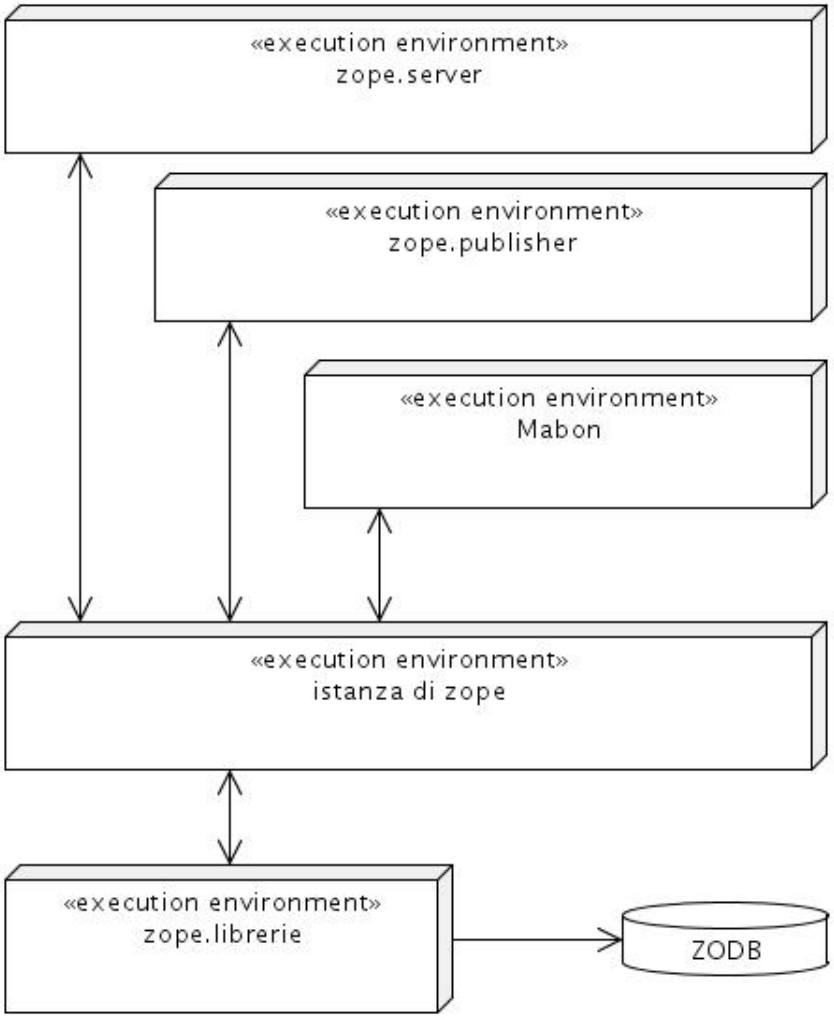
Lo ZODB, in particolare, soddisfa tutti i requisiti delle specifiche ACID[9]: atomicità, consistenza, isolamento (di ciascuna transazione dalle altre) e durabilità (persistenza su disco). Inoltre, il suo utilizzo è totalmente trasparente al programmatore (si veda più avanti la descrizione della classe Persistent), supporta l'Undo ed è configurabile con differenti backends.

Uno schema a grandi linee dell'interazione tra Zope e Mabon è illustrato in Figura 2.1.

Date le premesse è adesso possibile fornire un'altra motivazione per la quale ho preferito Zope 3 rispetto ad un'implementazione tramite un server L.A.M.P.[10] per una tesi di progettazione in UML: la natura stessa di Zope 3, quella di framework basato sull'interazione di oggetti, che ben si sposa alla strutturazione dei diagrammi UML.

[5] http://it.wikipedia.org/wiki/Mac_OS_X
[6] http://it.wikipedia.org/wiki/Microsoft_Windows
[7] Lo ZODB non è un database relazionale (come MYSQL usato in Mabon I) ma specializzato nello gestione di oggetti.
[8] http://www.w3schools.com/html/html_forms.asp
[9] http://it.wikipedia.org/wiki/ACID
[10] Linux Apache (webserver) MySQL PHP

Figura 2.1: Rappresentazione macroscopica del framework Zope 3 e dell'applicazione Mabon.



2.3 Flusso delle informazioni

Il flusso dei dati e delle azioni di un'applicazione come Mabon che utilizza il webserver integrato di Zope (Zope.server), può essere schematizzato come segue:

- i dati dell'utente vengono ricevuti e reinviati a quest'ultimo dal server che supporta clients HTTP[11] e FTP[12].
- le richieste e i dati ricevuti dal server vengono analizzati e vagliati dal Publisher, il quale istanzia un oggetto Request del tipo consono alla richiesta e ai dati; viene così generata una HTTP/WebDav[13]Request o FTPRequest o BrowserRequest ecc, tutte classi che implementano l'interfaccia IPublisherRequest. La classe Request interagisce con gli altri componenti dell'applicazione e/o del framework tramite operazioni di *object publishing*, caratteristica che rende Zope unico. Un'operazione di *object publishing* è composta da due parti:
  - il traversing: ovvero, la ricerca dell'oggetto "indirizzato" dalla richiesta proveniente dal browser;
  - il publishing: ovvero, l'invocazione di tale oggetto e le interazioni con lo ZODB;
- sempre il Publisher si occuperà di reinviare al server l'output generato nella fase di publishing.
- l'applicazione: ovvero, il codice scritto dall'utente che interagisce direttamente o indirettamente con le parti suddette e con le librerie offerte da Zope.

[11] http://it.wikipedia.org/wiki/Hyper_Text_Transfer_Protocol
[12] http://it.wikipedia.org/wiki/File_Transfer_Protocol
[13] http://en.wikipedia.org/wiki/WebDAV

L'applicazione può ridefinire, estendere o limitarsi ad utilizzare tutte quelle parti offerte da Zope che gestiscono vari aspetti cruciali per un'applicazione web–based quali i permessi, la sicurezza, l'autenticazione, ecc.

Un diagramma di attività che illustra le fasi di traversing e publishing si trova in Figura 2.2.

2.4 Organizzazione dei componenti

2.4.1 Generale

Come già premesso, tutte le entità che verranno illustrate nel resto della sezione sono implementate in Python e costituite da classi (`class`[14] in Python) o funzioni (`def`).

2.4.2 Interfacce

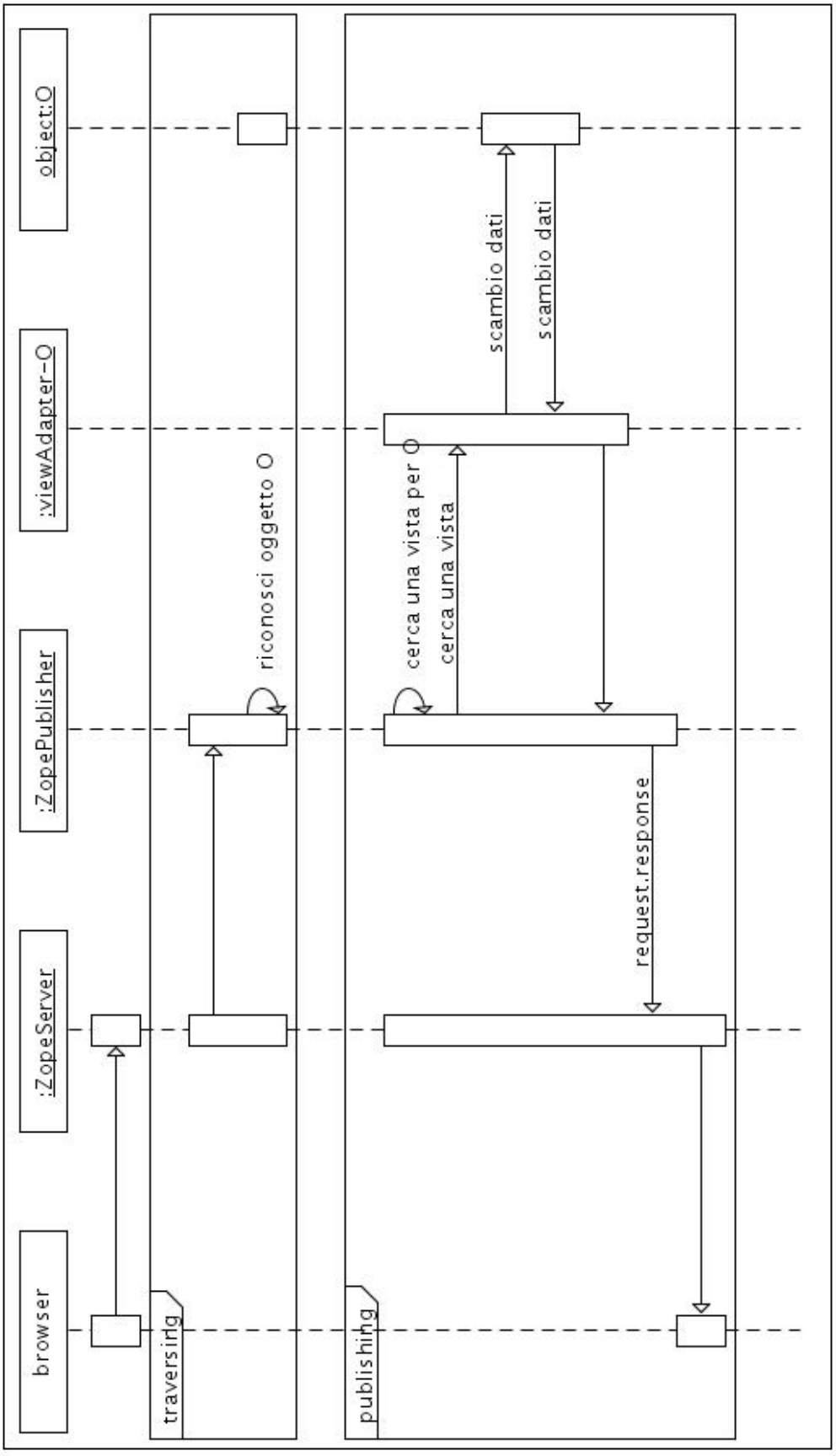
Un'interfaccia rappresenta una sorta di contratto che ciascun componente che l'implementa deve rispettare e specifica quali attributi, metodi e/o funzionalità saranno, quindi, incorporate nell'oggetto o cosa un componente può aspettarsi da un altro componente.

Un oggetto può essere caratterizzato – ovvero *offrirne* nel gergo di Zope – da più di una interfaccia e quindi un componente può implementare più di una interfaccia. È prassi che un componente possa implementare un'interfaccia che non specifichi alcun attributo o metodo; tali interfacce sono definite come Interfacce Etichetta, *marker interfaces*, e vengono utilizzate tra l'altro – ma non solo – per raggruppare oggetti o esprimere quale sia l'interfaccia principale tra tutte quelle offerte.

Python non contiene un costrutto per le interfacce, quindi Zope utilizza un

[14] http://docs.python.org/tutorial/classes.html

Figura 2.2: Diagramma di attività con le fasi di traversing e publishing.



proprio stereotipo definito nel pacchetto Zope.interface. Inoltre, Zope non fornisce un meccanismo di controllo a garanzia del fatto che un componente implementi tutte le parti – attributi e metodi – specificate dalla o dalle interfacce implementate. Per convenzione tutti gli identificativi di un'interfaccia iniziano per I maiuscola.

Esempi di interfacce in Mabon sono IUtente e IScheda entrambi nel pacchetto interfacce.

2.4.3 Componenti

Dal punto di vista di un programmatore Zope, l'unità fondamentale è il componente.

Un componente può essere costituito da una funzione o una classe e quest'ultima può essere creata da zero o implementare un'interfaccia definita dall'utente o presente nelle librerie di Zope.

2.4.4 Componenti di contenuto

I componenti di contenuto sono gli oggetti che contengono i dati usati dall'applicazione e devono offrire tutte le funzionalità necessarie all'aggiornamento e alla conservazione persistente degli stessi. Non è responsabilità di un componente di contenuto fornire una presentazione intellegibile per l'utente dei dati o eseguirvi delle operazioni, funzionalità rispettivamente delegate alle viste e agli adattatori in genere.

Esempi di componenti di contenuto in Mabon sono la classe Scheda e la classe Oggetto.

2.4.5 Oggetto

Nel gergo di Zope 3[15] la parola *oggetto* è un termine che si adatta al contesto in cui viene usato: può riferirsi a istanze di una classe, a moduli scritti in Python

[15][Wei08] pagina 48

(quindi le classi stesse) o alle interfacce stesse (alcune Interfacce Etichetta possono essere utilizzate per fornire funzionalità).

Frequentemente un oggetto è un'entità derivante dall'associazione logica di uno o più componenti ed è il risultato dell'astrazione informatica di un concetto o di una cosa realmente esistente.

Esempi di oggetti in Mabon sono le schede e gli utenti, intesi sia come istanze sia come classi; per distinguere i due casi, i riferimenti alle istanze sono scritte con lettera minuscola (scheda) mentre classi e moduli con lettera maiuscola (Scheda).

2.4.6 Adattatori

Gli adattatori sono il mezzo tramite il quale componenti non progettati per lavorare insieme possono essere resi in grado di interagire. Un adattatore viene caratterizzato dalla frase «*fornisce un interfaccia e adatta un componente*», ovvero fa sì che il componente adattato possa agire in conformità alle richieste (interfaccia implementata dall'adattatore) o specifiche di un altro oggetto che si aspetta di interagire secondo i dettami di una certa interfaccia.

Un adattatore può operare e anche modificare un oggetto adattato.

Un adattatore può avere un identificativo unico, definito come *nome* nel resto della tesi, ed è consentito che adattatori diversi abbiano lo stesso nome, perché identificabili in base alle interfacce che adattano e implementano.

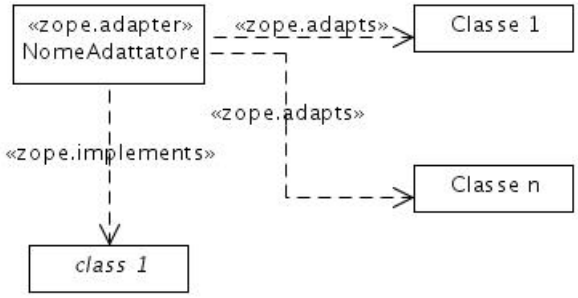
Un diagramma di classe che schematizza la struttura di un multiadattatore è visibile in Figura 2.3.

Un adattatore Zope agisce come l'omonimo elettrico: può interagire con altri componenti elettrici (si innesta nella presa) poiché offre una struttura che rispetta ed è compatibile con "l'interfaccia" implementata dalla presa nel muro, ma al contempo la adatta a quella della spina che viene inserita nell'adattatore stesso.

2.4.7 Viste

Le viste sono dei multiadattatori, ovvero degli adattatori che adattano più

Figura 2.3: Diagramma di classe di un adattatore.



interfacce.

La loro funzione è quella di consentire la fase di publishing, ovvero di adattare un oggetto alle richieste (rappresentate dall'interfaccia implementata direttamente o indirettamente dall'adattatore) del `Publisher` e quindi consentire una presentazione dell'oggetto adattato compatibile con l'utente.

2.4.8 L'applicazione utente

In Zope 3 un'applicazione utente è un'entità costituita da due componenti:

- la richiesta effettuata dalla applicazione su richiesta dell'utente;
- la risposta che verrà ritornata a quest'ultimo.

La scomposizione in componenti garantisce la massima flessibilità verso vari tipi di richieste (HTTP, FTP, ecc), che saranno implementate e contenute in componenti caratterizzati da interfacce diverse. Ciascuno di questi componenti verrà poi adattato all'istanza della classe `Request` dal componente `Publisher`, in maniera da offrire un'interfaccia standardizzata ma flessibile a vari metodi di input.

Zope supporta nativamente richieste HTTP e FTP, ma è possibile estendere il framework per aggiungere dei componenti in grado di gestirne ulteriori tipi.

Da quanto suddetto si evince che il flusso dei dati di un'applicazione Zope è

incentrata sul componente `Publisher`, che "traduce" le azioni richieste e i dati inviati dall'utente in interazioni fra componenti, e l'istanza della classe `Request`, all'interfaccia della quale gli oggetti coinvolti da Publisher devono "adattarsi".

Il componente Publisher, sebbene estendibile e modificabile dal programmatore, non è stato modificato per la scrittura di questa tesi dato che tutte le richieste sono di tipo `Browser` e quindi pienamente supportate dal framework Zope.

2.4.9 Utilità

Un'utilità è un componente che fornisce delle funzionalità ad altri componenti ma non opera su di essi, quindi potremmo dire che permette un accesso in sola lettura.

Un'utilità che dipende dal contesto, ossia dai dati dell'istanza dell'oggetto adattato, viene implementata come un adattatore, mentre quelle autarchiche sono utilità propriamente dette.

Anche le utilità possono avere un nome.

2.4.10 Registro dei componenti

Il registro dei componenti è un'entità che contiene le informazioni presenti nei vari files di configurazione forniti da Zope e dall'applicazione. Tali files di configurazione terminano con l'estensione `.zcml` e sono scritto in ZCML (Zope Configuration Markup Language[16]), un linguaggio basato su XML.

Nel registro dei componenti i vari componenti vengono registrati, viene loro opzionalmente fornito un nome e i permessi per le varie funzioni che un altro componente deve avere o ereditare potervi interagire; inoltre, il registro consente la ricerca di un componente in base alle interfacce implementate o adattate.

[16]http://wiki.zope.org/zope3/ZCML

2.4.11 Sicurezza, permessi e ruoli

Come accennato nella sottosezione precedente, ciascun componente può specificare quali Permessi deve avere un componente che voglia interagire con esso. È possibile utilizzare i Permessi predefiniti di Zope e/o definirne di nuovi, come avviene in Mabon. È possibile specificare un Permesso per accedere ai dati di una classe e uno differente per modicarli, ma anche dare dei Permessi differenti a singoli attributi o metodi rispetto alla classe stessa.

Qualsiasi oggetto che venga elaborato dal `Publisher` deve specificare quali Permessi siano necessari per il suo utilizzo; nel caso in cui ciò non sia stato programmato nel registro dei componenti, Zope "avvolge" il componente non protetto con un componente detto *SecurityProxy*, che ne vieta l'accesso.

Un Ruolo (`Role`) è un identificativo che raggruppa più Permessi; l'attribuzione di tale Ruolo ad un componente, garantisce l'accesso ad altri componenti protetti dai suddetti Permessi. Così come i Permessi, è possibile utilizzare i Ruoli predefiniti di Zope e/o definirne di nuovi; quelli implementati in Mabon sono elencati nella sottosezione «Utenti» 3.3. Il Ruolo predefinito da Zope `Zope.anybody` viene attribuito ad un utente non autenticato e quindi la vista Login dovrà permettere la visualizzazione anche a tale Ruolo.

Un utente che interagisce con un'applicazione Zope viene rappresentato con un oggetto `Principal`; a questi oggetti può essere associato un oggetto Ruolo, in caso contrario l'oggetto `Principal` erediterà il Ruolo predefinito. Gli oggetti che rappresentano la Richiesta effettuata dall'utente erediteranno questi Permessi che verranno confrontati con quelli richiesti dai componenti utilizzati, ad esempio le viste.

I Permessi e i Ruoli vengono specificati nei files `.zcml`.

2.4.12 Eventi

I componenti di Zope che agiscono su istanze di oggetti generano degli eventi

che possono essere e vengono intercettati da altri componenti, sia forniti da Zope sia definibili dall'utente. Al programmatore è consentito creare nuovi eventi, ma è suo onere far generare ai componenti di un'applicazione gli eventi relativi alle modifiche effettuate dai componenti stessi.

Gli eventi vengono definiti dall'interfaccia che ha un nome descrittivo dell'azione che lo ha generato; ad esempio, l'evento associato alla creazione di un oggetto è `IObjectAddedEvent`, quello relativo alla modifica `IObjectModifiedEvent`, ecc.

Un gestore di eventi è un adattatore che adatta sia l'interfaccia dell'evento sia quello dell'istanza (se l'evento si riferisce ad un'azione che agisce su un'istanza); ad esempio, un gestore di eventi «*è stato creato un oggetto scheda*» è un adattattore per le interfacce `IObjectAddedEvent` e `IScheda`.

2.4.13 La skin o tema

La skin rappresenta l'aspetto ovvero il tema dell'applicazione.

Senza dilungarci in una trattazione sulle skin che esula dagli scopi di questo testo, una skin è sostanzialmente costutita da vari componenti che interagiscono fra loro, definendo l'interfaccia utente, permettendone la estensione o la modifica e fornendo dei collegamenti al e per il codice scritto dal programmatore. Questi componenti possono essere rimpiazzati o estesi dal programmatore e tra essi possono essere definiti dei raggruppatori di visualizzatori o *viewlet managers*. Compito di quest'ultimi è quello fornire dei "punti di aggancio" e dei raggruppatori per altri componenti – detti visualizzatori o *viewlet* – in grado di mostrare all'utente particolari contenuti o funzioni contestualizzati al Ruolo o all'azione correntemente svolta dall'utente.

Mabon estende e adatta alle proprie esigenze la skin di default di Zope3, il cui nome è "Rotterdam".

Un esempio di visualizzatore in Mabon è quello che consente la modifica dello stato di una scheda da *bozza* a *documento*, e viceversa, tramite una classe in Python che genera del codice XHTML. È contestualizzato perché il contenuto del

visualizzatore varia in funzione del Ruolo dell'utente e dello stato della scheda. Il suo raggruppatore ne visualizza il contenuto all'interno della propria area, una zona lungo il bordo sinistro dello schermo.

2.4.14 ZPT, i modelli di pagina e il linguaggio TAL

Generare del codice in Python è un'operazione lenta e tediosa, soprattutto se si desidera mantenere almeno una parvenza di indentazione nel codice prodotto. Inoltre, è quasi impossible chiedere ad un grafico o web designer di creare per noi la pagina come file a sé: sarebbe molto più comodo poter disporre della possibilità di lavorare separatamente sulle pagine XHTML o sulle porzioni di esse che ci servono e poi collegarle ai vari moduli del programma.

Zope 3 offre per questa ragione due metodi per la generazione del codice XHTML di una pagina web: tramite classi o funzioni in Python oppure per mezzo di *Page Templates*, ovvero Modelli di Pagina, costituiti da files con estensione `.pt`.

Il codice contenuto in un modello di pagina può essere un'intera pagina HTML (come nel caso dei files collegati alle viste) o solo porzioni di essa (come nel caso dei files collegati ai visualizzatori). Lo *Zope Page Templating* (ZPT), consente di utilizzare dei files XML, contenenti non solo la struttura della pagina ma anche delle istruzioni che verranno eseguite quando la pagina viene invocata (come vedremo più avanti) offrendo la possibilità di creare un contenuto dinamico, similmente a quanto avviene con linguaggi server-side quali PHP. Poiché queste istruzioni risiedono in un *namespace*[17] XML separato[18], non vanno in conflitto con le istruzioni XHTML e consentono di lavorare con programmi XML-compatibili sull'aspetto della pagina.

Queste istruzioni si chiamano *Template Attribute Language*[19] (TAL), linguaggio degli attributi dei modelli. Le istruzioni TAL hanno accesso alle variabili

[17] http://www.w3.org/TR/REC-xml-names/
[18] http://xml.zope.org/namespaces/tal
[19] http://en.wikipedia.org/wiki/Template_Attribute_Language

globali definite in Zope e a quelle locali delle istanze delle classi alle quali sono collegate.

2.5 Il paradigma Modello Vista Controllore

Zope 3 incoraggia un approccio secondo il paradigma di programmazione[20] Modello-Vista-Controllore[21].

La struttura di Zope rende molto semplice creare degli oggetti che rappresentino i Modelli progettati, la cui struttura viene rispecchiata dalla classe interfaccia. La classe IScheda in particolare è il prodotto delle richieste catalografiche degli utenti e degli esperti scientifici della Soprintendenza e la conseguente rappresentazione di tali dati tramite un Modello informatico. Anche gli oggetti Utente, Etichetta e i vocabolari sono Modelli.

La parte di Vista, ovvero la rappresentazione dei Modelli, è svolta dalle viste, la cui unica funzione è appunto quella di fornire una rappresentazione intelligibile per l'utente dei Modelli.

Il Controllore è rappresentato da elementi in parte forniti da Zope e in parte programmati da me. In particolare, il componente `Publisher`, responsabile della scomposizione delle URL e delle richieste che giungono dall'utente e dell'invio delle risposte, fa parte di tali elementi. All'interno dei moduli di Mabon, i gestori di visualizzatori, il componente Mabon con tutti i suoi gestori di eventi, i generatori, i componenti di ricerca e parecchi adattatori sono riconducibili al ruolo di Controllore.

Riassumento per grandi linee, ecco una tabella di corrispondenze:

- Modello = gli oggetti principali e i vocabolari;
- Vista = alcuni adattatori, i visualizzatori e le viste;

[20] http://it.wikipedia.org/wiki/Design_pattern
[21] http://it.wikipedia.org/wiki/Model-View-Controller

- Controllore = il componente Publisher, i generatori, i moduli di ricerca, sessione e login, alcuni adattatori.

Capitolo 3

L'analisi dello scenario

3.1 Il problema della digitalizzazione

3.1.1 La necessità di un archivio informatizzato

Nella mia esperienza da informatico presso Soprintendenza ho potuto constatare come una consistente parte delle attività lavorative degli uffici Catalogo Monumenti (schedatura dei beni immobili), Catalogo Gallerie (schedatura dei beni mobili) e Fototeca sia imperniata sulla consultazione e digitalizzazione di archivi in origine cartacei.

Al momento della creazione di Mabon I, gli archivi del Catalogo Monumenti e Catalogo Gallerie erano stati informatizzati almeno in parte, mentre gli archivi della Fototeca erano ancora in formato unicamente cartaceo. Da qui l'idea di sviluppare un software in grado di aiutare e sveltire il processo di consultazione del materiale fotografico, eterogeneo per quel che concerne i supporti fisici (lastre in vetro[1], negativi, diapositive, immagini digitali) e i diritti legali (alcuni negativi sono di proprietà della Soprintendenza, altri di enti esterni).

[1] http://www.photogallery.it/storia/ichimica.html

3.1.2 Struttura della scheda

Insieme alla dott.ssa Severina Russo, responsabile del catalogo Gallerie e della Fototeca, abbiamo stilato una lista di parametri che fossero in grado di contenere tutte le informazioni necessarie a digitalizzare il contenuto dei registri della Fototeca. Il risultato è illustrato dai seguenti insiemi:

3.1.2.1 Informazioni catalografiche

Le informazioni catalografiche sono costituite dai campi:

- codice di inventariazione progressivo del "cartoncino"[2], che contiene il codice di archiviazione in Fototeca. Per le schede in Mabon che sono una digitalizzazioni degli archivi già esistenti, questo codice è la trascrizione di quello presente sul formato cartaceo, mentre per le schede nuove viene assegnato in maniera progressiva. Come suggerisce il titolo, in genere questo codice è un numero (anche se talvolta mischiato a lettere come in "435/bis");

- codice di inventariazione del negativo che è assegnato dalla Soprintendenza se quest'ultima detiene i diritti o la proprietà del negativo, oppure dall'Ente esterno proprietario;

- data del negativo, che contiene la data della presa in carico del negativo da parte della Soprintendenza;

- tipo di fondo archivistico, contenente un valore fra quelli di un vocabolario chiuso, costituito dalla lista dei dispositivi hardware dove può risiedere la versione digitalizzata del materiale fotografico, con le seguenti opzioni: "non digitalizzata", "CD", "DVD", "hard disk", "server 'Cristina'[3]" che si trova in

[2]Supporto cartaceo sul quale è incollato il positivo, il negativo imbustato o la lastra in vetro.

[3]È un server interno della Soprintendenza con funzione di database delle schede catalografiche digitalizzate prima del 2005. Una scheda catalografica http://www.iccd.beniculturali.it/Catalogazione/standard-catalografici/normative/nomative è un tracciato contenente i dati realativi alla schedatura di un oggetto d'arte secondo regole stabilite dall'Istituto Centrale Catalogo e Documentazione (ICCD):

Soprintendenza, "server del progetto 'ArtPast'[4]", "USB data pen Drive", "Altro";

- campo FTAN[5] è il codice che, in sostanza, permette il collegamento tra la scheda nel database di Mabon e il file contenente l'immagine digitalizzata;
- fotografo, ovvero nome e cognome del fotografo;
- supporto fisico contenente un valore fra quelli di un vocabolario chiuso, costituito dalla lista dei supporti fisici del materiale fotografico, con le seguenti opzioni: "lastra fotografica" (vari formati), "diapositiva", "pellicola" (vari formati), "stampa fondo riservato" (ovvero la stampa di un negativo del quale la Soprintendenza non detiene i diritti), "digitale" e "digitale fondo riservato".

3.1.2.2 Informazioni geografiche (sul soggetto rappresentato nel materiale fotografico)

Le informazioni geografiche sono costituite dai campi:

- indirizzo,
- comune,

http://www.iccd.beniculturali.it/

[4]http://www.artpast.org/

[5] **Da Normativa v. 2.00 per le schede OA e D:**
http://www.iccd.beniculturali.it/Catalogazione/standard-catalografici/normative/scheda-oa-d

FTAN: Negativo → *Indicazione del numero di negativo delle fotografie eseguite dai laboratori fotografici premettendo ai singoli numeri di negativo la sigla delle Soprintendenze o Istituti competenti (si veda Appendice C) o il nome di altri enti o privati. Esempi:*

SBAS PR 3254
ICCD E 2576
Alinari 3280

- provincia, dove risiede il soggetto raffigurato nel materiale fotografico;
- numero di archiviazione dell'oggetto rappresentato, che contiene il numero o codice sotto il quale l'oggetto rappresentato indicato nell'omonimo campo è stato catalogato;
- ubicazione, ovvero un'indicazione catalografica, archivista e geografica più specifica (ad esempio il numero di inventariazione di un oggetto in un museo o l'interno e il piano per un capitello all'interno di un palazzo).

3.1.2.3 Informazioni sull'autore e sul soggetto

Le informazioni autore–soggetto sono costituite dai campi:

- l'autore che ha realizzato il soggetto fotografato;
- una breve descrizione del soggetto fotografico stesso.

3.1.2.4 Informazioni tecniche sul supporto fisico

Le informazioni tecniche sono costituite dai campi:

- la datazione del supporto fisico;
- l'altezza in cm del supporto fisico;
- la larghezza in cm del supporto fisico;
- la profondità in cm del supporto fisico;
- il massimo quantitativo di lux ai quali il supporto fisico può essere esposto;
- la massima umidità alla quale il supporto fisico può essere esposto;

- il codice della vetrina o contenitore in cui il supporto fisico è conservato.

Una scheda inoltre possiede un proprio status che specifica se si tratta di una bozza, ossia non ancora approvata e validata, o di un documento con validità legale.

3.1.3 Metodo di archiviazione

L'inventariazione del materiale fotografico si basa su due parametri: il supporto fisico e il numero di inventariazione del negativo. Ciascun tipo di supporto fisico identifica un diverso archivio con propria numerazione della Soprintendenza e il numero di negativo la posizione all'interno dell'archivio.

Vi è però il caso in cui un negativo non abbia un numero di inventariazione relativo ad un archivio interno della Soprintendenza o perché non è di proprietà della Soprintendenza (ma ne è proprietario un Ente diverso da essa) o perché non esiste la presa in carico[6] (come nel caso di materiale derivante da una donazione).

In questo caso si tratta di un supporto di tipo "*fondo riservato*" (digitale o stampa) e per l'identificazione univoca viene utilizzato il numero di inventariazione progressivo del cartoncino.

Riassumendo: se la Soprintendenza detiene i diritti legali delle immagini l'identificazione è basata sulla coppia <NumeroInventariazioneNegativo, SupportoFisico>; in caso contrario la coppia è <NumeroInventariazioneScheda, SupportoFisico>.

3.2 Lo stato della scheda

All'interno dell'applicazione una scheda può trovarsi in uno stato dal seguente

[6] L'attribuzione del numero di inventariazione al negativo è condizionata alla presa in carico.

insieme: "bozza", "proposta", "documento", "obsoleta".

La transizione tra gli stati è illustrata il figura 3.1, che rappresenta il diagramma di stato per una scheda. Sui nodi sono descritti gli stati in cui si può trovare una scheda, mentre sugli archi le azioni che vengono effettuate.

Gli stati evidenziati in blu sono quelli stabili, ossia quelli nei quali la scheda resta se non interviene uno Schedatore o un Validatore a modificarli; gli stati in rosso sono quelli transitori nei quali la scheda entra su richiesta di uno Schedatore e Validatore e ne esce quando viene elaborata dall'altra delle due precedenti figure.

Gli stati con il testo scritto in bianco contrassegnano una scheda che si trova nell'area di lavoro dello Schedatore che ne è proprietario, ed è trovabile con una ricerca semplice effettuata anche da utenti diversi dallo Schedatore; è però mascherabile per mezzo di una ricerca avanzata.

Gli stati col testo in nero sono associati a schede presenti nel contenitore delle schede approvate (unico per tutta l'applicazione).

Lo stato di *bozza* implica che una scheda si trovi nell'area di lavoro di uno Schedatore che può modificarla o cancellarla o proporla per promozione a *documento*.

Lo stato di *documento* implica che la scheda non è più modificabile, anche se rimane cancellabile da un Validatore. Per consentirne la modifica, ovvero l'aggiornamento, un Validatore deve modificarne lo stato in *obsoleta*.

Lo stato *proposta* indica che lo Schedatore ha proposto una sua scheda per il vaglio di un Validatore. Quest'ultimo può decidere se promuoverla a *documento* o respingerla e riportarla in stato di *bozza*.

Lo stato *obsoleta* indica un documento selezionato da un Validatore per l'aggiornamento che verrà effettuato da uno Schedatore. Quest'ultimo trasformerà la scheda in *bozza*, operazione che implica l'automatico trasferimento nella sua area

Figura 3.1: Stati di una scheda

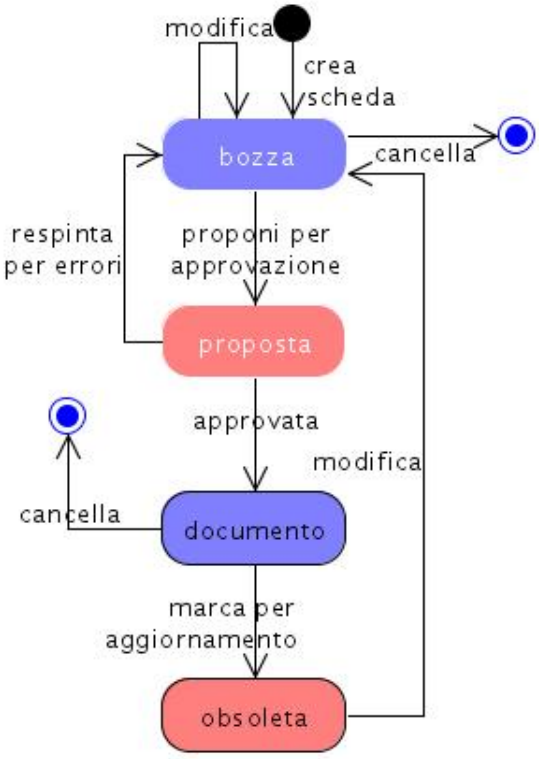
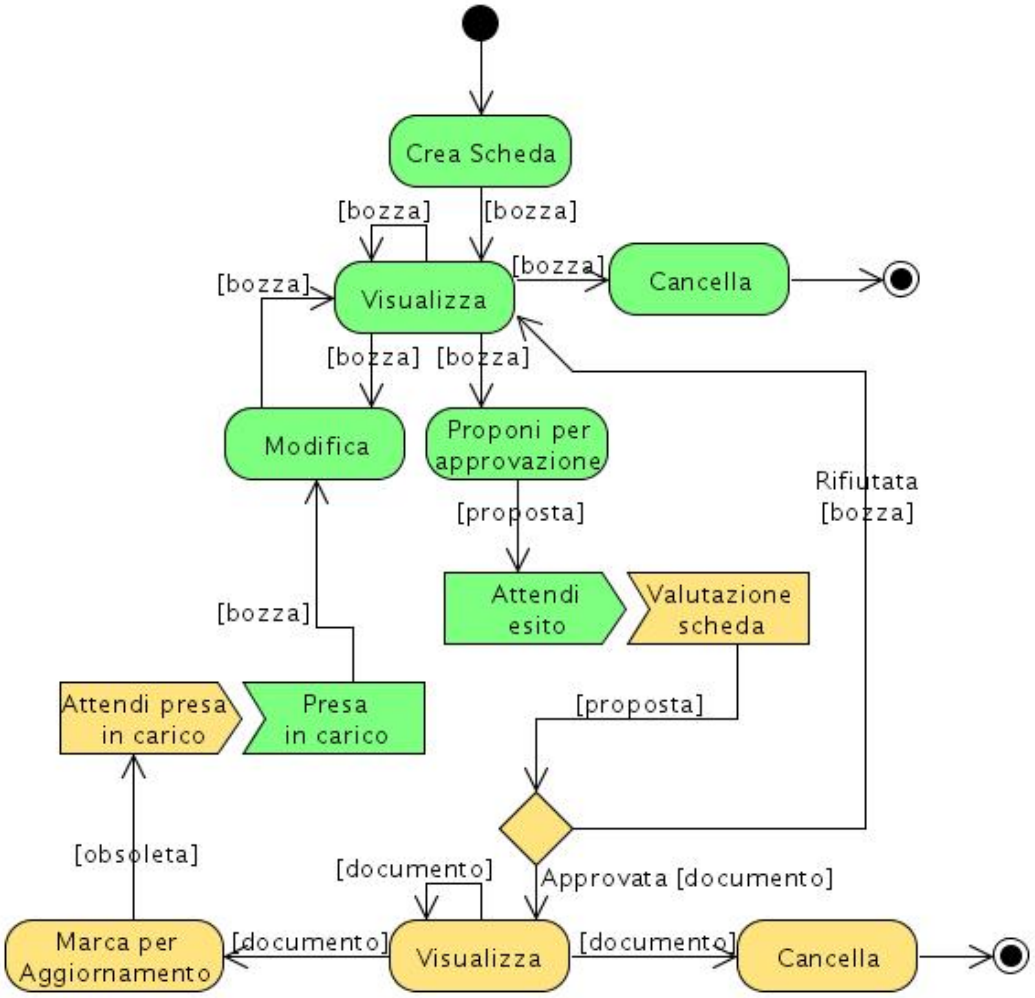


Figura 3.2: Attività effettuabili su una scheda



di lavoro.

3.2.1 Le azioni effettuabili su una scheda

Duale al diagramma in figura 3.1 è quello in figura 3.2, che contiene il diagramma di attività di Schedatori e Validatori relativamente a una Scheda. Sui nodi sono riassunte le azioni che vengono effettuate sulla scheda mentre sugli archi gli stati in cui si trova la scheda.

In verde sono evidenziate le azioni di pertinenza dello Schedatore, mentre in

arancione quelle di un Validatore.

3.3 Utenti

Nella tesi si considera la persona che utilizza il programma da un punto di vista funzionale tenendo conto dell'effettivo livello di privilegio o Ruolo associato all'account. Ciascun utente del sistema può ricoprire un solo Ruolo; nel caso in cui si desideri che un utente possa ricopire più Ruoli, si dovranno creare un numero equivalente di account con differente campo login.

Di seguito i Ruoli definiti per Mabon:

- Ricercatore: colui il quale ha solo diritto di consultazione del materiale presente sul sito;
- Schedatore: colui il quale può creare nuove schede, apporre correzioni a quelle presenti nel sistema (che però dovranno poi ripassare al vaglio di un Validatore), modificare le proprie bozze, proporre una scheda ad un Validatore.
- Validatore: esperto scientifico del settore della catalogazione dei beni culturali al quale è riservato il compito di valutare la correttezza delle schede compilate dagli schedatori e l'eventuale promozione dello status di una scheda da "bozza" a "documento approvato";
- Scansionatore: l'addetto alla produzione e importazione in Mabon del materiale digitale raffigurante il materiale fotografico schedato;
- Amministratore: il responsabile della creazione e cancellazione degli account per gli utenti e le attribuzioni dei Ruoli, ma non ha alcuna competenza in senso storico-artistico, quindi non ha accesso alle schede.

3.4 Internazionalizzazione

Sebbene i contenuti dell'applicazione saranno scritti in lingua italiana, Mabon è stato progettato in maniera da consentire una facile traduzione di tutto il testo presente nell'applicazione e non inserito dagli utenti.

Il testo facente parte dell'interfaccia di Mabon e tutti i messaggi inviati all'utente vanno marcati così che sia possibile estrarli dall'applicazione con un opportuno strumento fornito da Zope e utilizzarli per la creare una tabella di traduzione. Quest'ultima verrà generata in un file nel formato `.pot` (`.po t`emplate, modello `po`) del pacchetto di utilità GNU *gettext*[7].

Il file .pot generato può essere rinominato .po e consegnato ai vari traduttori madrelingua che per mezzo di programmi quali KBabel[8], poEdit[9] o gtranslator[10] potranno fornire la traduzione di tutto il testo necessario a generare la versione straniera dell'interfaccia di Mabon; la generazione è un azione estremamente semplice: basta compilare il file ricevuto e copiare la versione binaria in un'apposita cartella dell'applicazione.

[7]Pagina del progetto GNU: http://www.gnu.org/software/gettext.
[8]KBabel: http://kbabel.kde.org
[9]poEdit: http://poedit.sourgeforge.net
[10]gtranslator: http://gtranslator.sourceforge.net

Parte II

La struttura dell'applicazione

Capitolo 4

Quadro generale

4.1 Organizzazione dei file e struttura gerarchica

Le classi contenenti i vari componenti di Mabon sono raggruppati con un criterio tematico in files, i quali a loro volta sono contenuti in pacchetti; la lista dei quali è mostrata in Figura 4.1.

Delle normali cartelle vengono promosse al ruolo di pacchetti tramite la presenza dello speciale file `__init__.py` (non rappresentato nelle immagini successive).

Le seguenti immagini mostrano in quali files sono contenuti i componenti descritti nei capitoli successivi.

All'interno della cartella `mabon` (Figura 4.2) si trovano tutti i componenti di contenuto e le relative interfacce, nonché i componenti e gli adattatori necessari alla interazione con le librerie offerte da Zope e tra i vari componenti stessi.

La cartella `immagini` contiene la cartella `importate` con le immagini importate in Mabon dagli scansionatori e collegabili alle schede e la cartella `nuove` con le immagini da importare. La struttura delle sotto cartelle di `nuove` e `importate` è identica: un primo livello di cartelle i cui nomi sono gli stessi dei supporti fisici e un secondo livello, ripetuto per ciascuna cartella del primo livello, contenente

Figura 4.1: Diagramma dei pacchetti

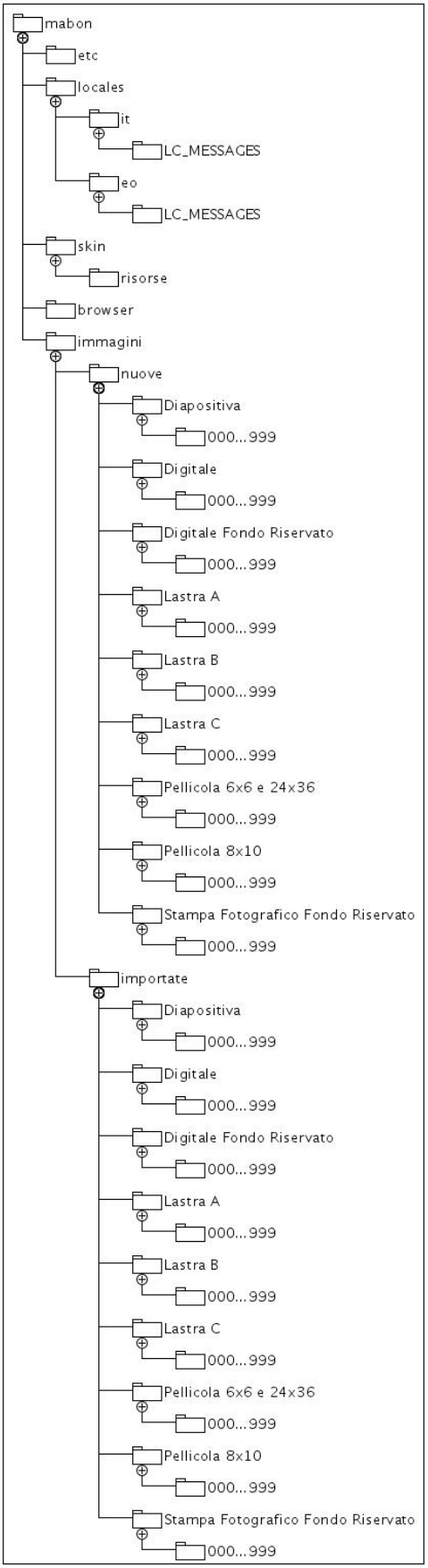
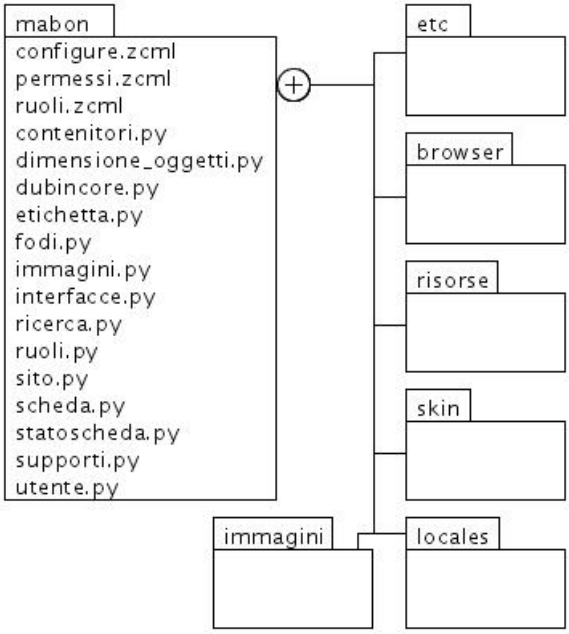


Figura 4.2: Pacchetto `mabon`



1000 cartelle numerate da 000 a 999. La spiegazione del perché esistono queste cartelle si trova nel testo in 6.1.4.14 a pagina 90.

La cartella `browser` (Figura 4.3) contiene adattatori – o per essere più precisi multi–adattatori – che svolgono il ruolo di Vista nel pattern MVC e porzioni di codice XHTML, contenuti nei files `.pt`, utilizzati per una visualizzazione ad hoc dei componenti di contenuto.

La cartella `skin` (Figura 4.4) contiene le modifiche, aggiunte e personalizzazioni al tema Rotterdam offerto da Zope3, presenti nei files `.pt` sotto forma di codice XHTML. Tra gli altri si annoverano i tre raggruppatori di visualizzatori, i visualizzatori stessi e l'ordinatore della posizione di apparizione dei visualizzatori nelle aree dei raggruppatori.

La cartella `risorse` (Figura 4.5) raccoglie tutto ciò che non è una vista, ma fa comunque parte del tema: tali componenti vengono definiti nel gergo di Zope3 come *risorse*. Fogli di stile, icone per i componenti di contenuto e files con codice javascript sono esempi di risorse.

Il pacchetto `etc` (Figura 4.6) contiene due files di configurazione che andranno

Figura 4.3: Pacchetto `browser`

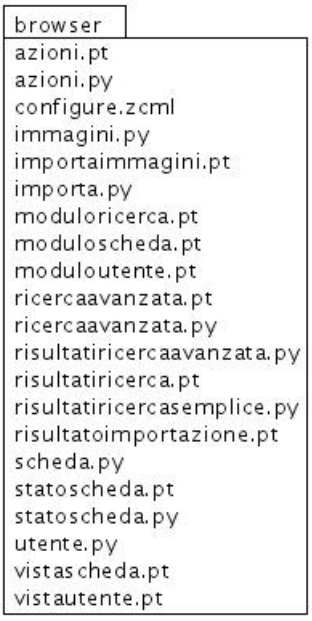


Figura 4.4: Pacchetto `skin`

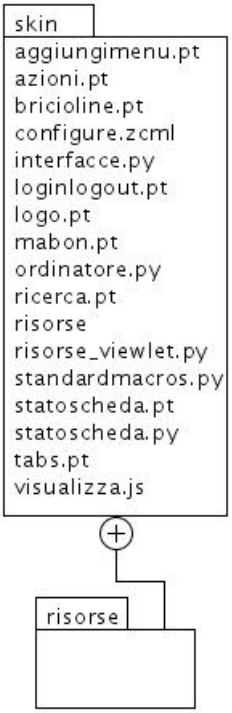
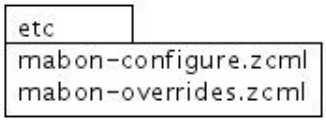


Figura 4.5: Pacchetto `risorse`

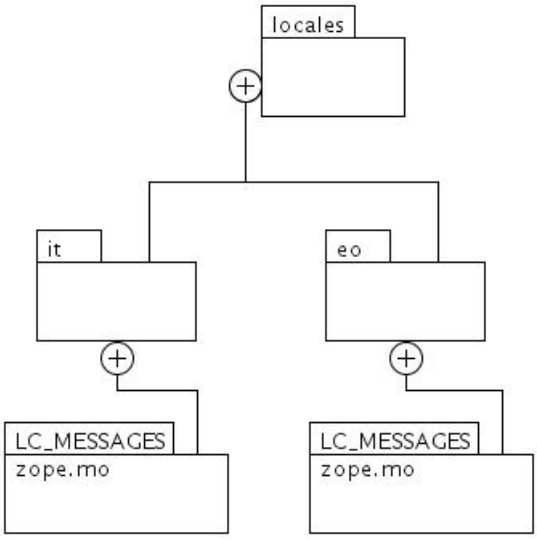
Figura 4.6: Pacchetto `etc`



copiati nella cartella `etc` dell'istanza di Zope, come illustrato in dettaglio nel capitolo 10.

La cartella `locales` (Figura 4.7) contiene le traduzioni di tutte le stringhe di Mabon e le informazioni necessarie per l'internazionalizzazione e la localizzazione del programma. Tali files sono nel formato gettext `.mo`/`.po` della GNU; maggiori dettagli si trovano nel capitolo 10. Nella figura 4.7 è rappresenta la struttura dei pacchetti per supportare due lingue: Esperanto e Italiano.

Figura 4.7: Pacchetto `locales`



Capitolo 5

I componenti pricipali e secondari

5.1 I componenti secondari

5.1.1 Contenitori

5.1.1.1 La relazione di contenimento in Zope3

Un contenitore è un oggetto che consente di aggiungervi altri oggetti offrendo la possibilità di indicizzarli con un identificativo univoco (si comporta quindi come un vocabolario Python ma non lo è per motivi di efficienza), e di imporre alcuni limiti o condizioni sugli oggetti contenuti. L'indicizzazione offre quindi la possibilità di scorrere verso il contenuto la catena del contenimento. La classe contenuta deve avere gli attributi `__name__` e `__parent__` che rispettivamente conterranno l'indice per l'oggetto contenuto nel contenitore e un riferimento all'oggetto contenitore stesso, nello stesso modo in cui vengono utilizzati i file '.' e '..' nel file system. Con questo artificio si costituisce un collegamento dal contenuto al contenitore, generando quindi la possibilità di scorrere verso il contenitore la catena del contenimento.

La creazione di questo doppio concatenamento è fondamentale perché gli adattatori `IPublishTraverse` siano in grado, dopo aver scomposto una URL in ele-

menti atomici, di "attraversare" un oggetto per arrivare al successivo o al precedente. Per oggetti semplici, non contenitori, l'attraversamento viene gestito con una vista, mentre per un oggetto che contenga altri oggetti o referenze ad essi, deve esservi un modo per distinguere ciascuna "strada" percorribile, l'indice appunto e una vista dell'oggetto indicizzato.

Ciascuna interfaccia di un oggetto contenitore e ciascuna interfaccia di un oggetto contenuto può anche specificare dei vincoli rispettivamente su quali interfacce contenere e da quali interfacce essere contenute. Tali vincoli vengono espressi tramite le funzioni `containers` e `contains` presenti nel pacchetto `constraints`[1].

5.1.1.2 I contenitori di Mabon

In Mabon sono utilizzati tre differenti contenitori: uno per le istanze degli oggetti scheda, uno per quelle degli utenti e uno per quelle delle etichette.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 5.1.

Ognuno dei tre oggetti contenitori implementati in Mabon è composto da un'interfaccia `IContenitore*`, da un componente di contenuto `Contenitore*` e dispone di un adattatore `SincronizzaNome*` dove al posto dell'asterisco va sostituito il nome dell'oggetto contenuto secondo la Tabella 5.1.

5.1.1.3 Le classi `IContenitore*`

L'interfaccia del contenitore non aggiunge alcun attributo o metodo a quelli ereditati da `IContainer`.

5.1.1.4 Le classi `Contenitore*`

Ciascuna classe `Contenitore*` fornisce il componente di contenuto per l'oggetto `Contenitore*`.

La derivazione dalla classe `BTreeContainer` consente un'interazione con lo

[1] `zope.app.container.constraints`.

Figura 5.1: Diagramma delle classi `Contenitore*`

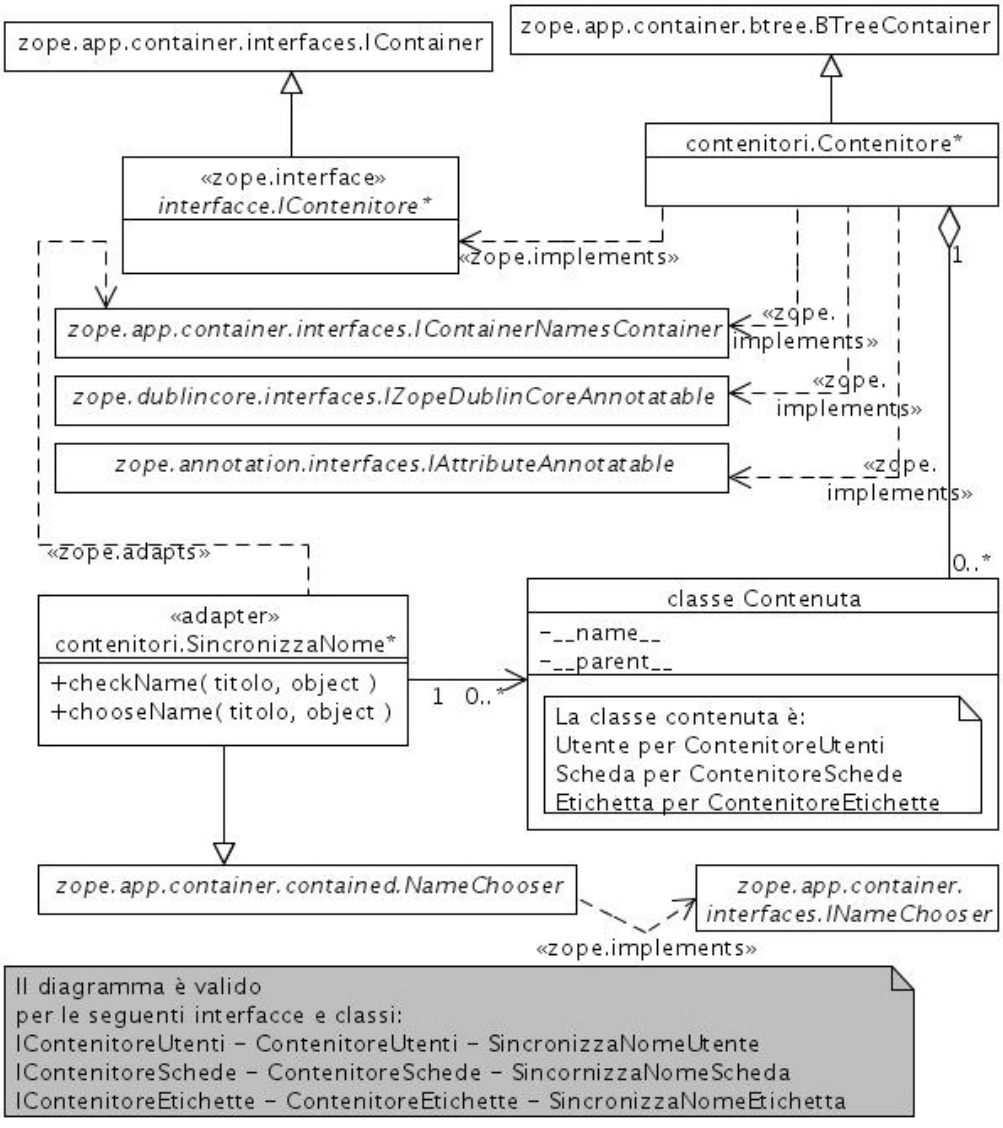


Tabella 5.1: Corrrispondenze tra contenitori, contenuti e adattatori.

interfaccia	componente	classe Contenuta	adattatore
IContenitoreSchede	ContenitoreSchede	Scheda	SincronizzaNomeScheda
IContenitoreUtente	ContenitoreUtenti	Utente	SincronizzaNomeUtente
IContenitoreEtichette	ContenitoreEtichette	Etichetta	SincronizzaNomeEtichetta

ZODB automatica e trasparente per il programmatore: la creazione, cancellazione e modifica delle istanze del contenitore saranno automaticamente inoltrate al database senza che il programmatore debba aggiungere ulteriori linee di codice, sebbene sia possibile modificare questo comportamento in caso di necessità.

Come suggerisce il nome, nella classe `BTreeContainer` le istanze degli oggetti contenuti sono organizzati secondo un albero B^+Tree[2] offrendo tempi di ricerca proporzionali al logaritmo della quantità degli oggetti contenuti.

Poiché il componente di contenuto dei tre contenitori implementati in Mabon – `ContenitoreSchede`, `ContenitoreUtenti` e `ContenitoreEtichette` – implementa anche l'interfaccia `IContainerNamesContainer`, i contenitori sono in grado di scegliere in maniera intelligente un indice per ciascun istanza contenuta partendo dai dati stessi presenti nell'oggetto. L'algoritmo da seguire nella scelta di tale indice viene fornito da un adattatore specifico per ciascuna interfaccia.

L'interfaccia caratterizzante l'oggetto utente è, come anticipato, IUtente, ma ne sono fornite anche altre: `IAttributeAnnotatable`, `IZopeDublinCoreAnnotatable` e `IPersistent`. Le due interfacce `I*Annotatable` vengono sfruttate tramite degli adattatori presenti nelle librerie di Zope per registare informazioni non associabili ai campi propri di un contenitore, ma che sono tuttavia utili, quali i commenti dell'Amministratore e le date di creazione e modifica.

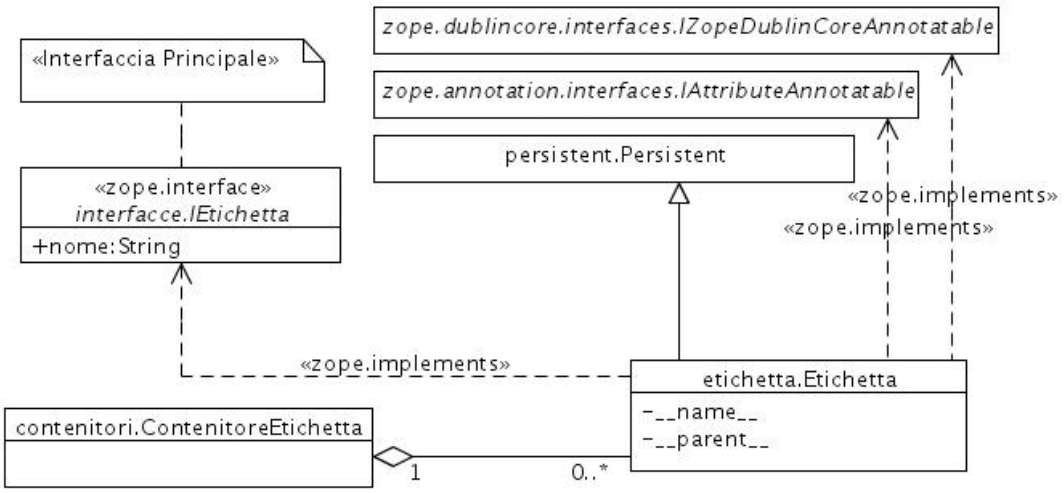
5.1.1.5 L'adattatore `SincronizzaNome*`

Ciascuna interfaccia IContenitore* ha associato un adattatore SincronizzaNome* secondo la tabella precedente.

Le classi SincronizzaNome* verranno descritte più avanti, nella sottosezione dedicata agli adattatori.

[2] http://it.wikipedia.org/wiki/B-Albero

Figura 5.2: Diagramma della classe `Etichetta`



5.1.2 L'oggetto etichetta

5.1.2.1 Generale

Le istanze dell'oggetto etichetta rappresentano le etichette che lo schedatore può applicare alle istanze dell'oggetto scheda di sua competenza. Ogni schedatore dispone delle proprie etichette private e non ha accesso a quelle di altri schedatori.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 5.2.

5.1.2.2 La classe `IEtichetta`

L'interfaccia `IEtichetta` è estremamente semplice, infatti consta di un solo attributo nome, contenente il nome dell'etichetta.

`IEtichetta` è anche l'interfaccia primaria – cioè caratterizzante – un oggetto che la implementi.

5.1.2.3 La classe `Etichetta`

La classe Etichetta fornisce il componente di contenuto per l'oggetto etichetta e implementa quanto specificato nell'interfaccia `IEtichetta`.

La derivazione da `Persistent` consente un'interazione con lo ZODB automatica e trasparente per il programmatore: la creazione, cancellazione e modifica

delle istanze di `Etichetta` saranno automaticamente inoltrate al database senza che il programmatore debba aggiungere ulteriori linee di codice, sebbene sia possibile modificare questo comportamento in caso di necessità.

Vengono aggiunti due nuovi attributi a quelli forniti dall'interfaccia: `__name__` e `__parent__`. Tali attributi sono necessari per le relazioni di contenimento tra l'oggetto `Etichetta` e l'oggetto `ContenitoreScheda` ed in generale per poter scorrere la catena delle relazioni di contenimento in entrambi i sensi come più dettagliatamente illustrato in 5.1.1.1 nella pagina 45.

L'interfaccia caratterizzante l'oggetto etichetta è, come anticipato, `IEtichetta`, ma ne sono fornite anche altre: `IAttributeAnnotatable`, `IZopeDublinCoreAnnotatable` e `IPersistent`. Le due interfacce `I*Annotatable` vengono sfruttate tramite degli adattatori presenti nelle librerie di Zope per registare informazioni non associabili ai campi propri di un'etichetta, ma che sono tuttavia utili, quali commenti dello schedatore e le date di creazione e modifica dell'etichetta stessa.

5.1.3 I vocabolari

5.1.3.1 Generale

La classe `SimpleVocabulary` offerta da Zope rende disponibile un componente estremamente utile ogniqualvolta sia necessario scegliere una o più opzioni da un insieme. La classe `SimpleVocabulary` costituisce un componente che offre le stesse caratteristiche del tipo `dict` in Python, ovvero un dizionario con indici.

Mabon implementa tre istanze della classe `SimpleVocabulary` che sono sempre utilizzati da un campo di tipo `Choice` presente in un'interfaccia, come illustrato successivamente.

5.1.3.2 Il vocabolario Ruoli

Contiene tutti i ruoli che può assumere un utente; viene utilizzato dal campo `IUtente.ruolo` e creato dal generatore `vocabolarioRuoli`.

5.1.3.3 Il vocabolario Fondi

Contiene tutti i fondi archivistici che possono contenere la versione digitalizzata del materiale fotografico descritto da una scheda; viene utilizzato dal campo `IScheda.fondo` e creato dal generatore `vocabolarioFondi`.

5.1.3.4 Il vocabolario Supporti

Contiene tutti i supporti fisici che possono che possono essere stati utilizzati per la realizzazione del materiale fotografico descritto da una scheda; viene utilizzato dal campo `IScheda.supporto` e creato dal generatore vocabolarioSupporti nel file supporti.py.

5.1.3.5 Il vocabolario Etichette

Contiene tutte le etichette che possono essere applicate a una scheda in stato bozza; viene utilizzato dal campo `IScheda.etichette` e creato dal generatore `vocabolarioEtichette`.

5.2 I componenti principali

5.2.1 Introduzione

Tutti i componenti principali di Mabon fanno largo uso delle tecniche di subclassing e delegazione proprie di Zope3, inoltre sono sempre costituiti dalle coppie <Interfaccia, Content Component>.

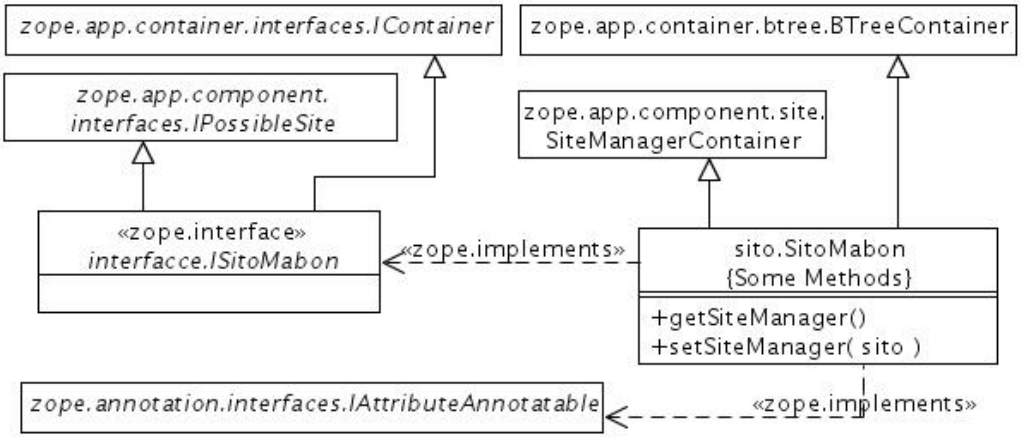
5.2.2 L'oggetto sito Mabon

5.2.2.1 Generale

Essenzialmente il sito Mabon è un'istanza di contenitore contenuto a sua volta nell'istanza del contenitore–"sito principale" definito nell'istanza di Zope.

Un sito consta di un contenitore e di un nuovo registro dei contenuti, chiamato gestore del sito. Quest'ultimo offre la possibilità di registrare dei componenti

Figura 5.3: Diagramma della classe `SitoMabon`



locali, ossia visibili solo all'interno del nuovo registro dei contenuti e non in quello generale dell'istanza di Zope – in tal caso verrebbero definiti componenti globali.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 5.3.

5.2.2.2 La classe `ISitoMabon`

Come suggerisce la derivazione da IContainer, l'interfaccia indica che il sito dovrà essere un contenitore nel quale verranno aggiunte le istanze dei vari componenti necessari al funzionamento del sito, tra cui i contenitori per gli utenti, quelli per le schede e quelli per le etichette, nonché le utilità quali l'indice delle istanze presenti nel sito, il catalogo per le ricerche, il servizio di autenticazione e le specifiche per le sessioni e i cookies da trasmettere agli utenti che si autenticheranno; tutte le suddette utilità saranno quindi locali e non globali.

L'interfaccia `IPossibleSite` indica a Zope che l'interfaccia `ISitoMabon` ha la potenzialità di essere un sito, (ma non lo è ancora dato che non offre l'interfaccia `ISite` che caratterizza i siti; l'istanziazione della classe che implementa ISitoMabon trasformerà l'interfaccia `IPossibleSite` in `ISite`).

5.2.2.3 La classe `SitoMabon`

L'interfaccia `IPossibleSite` richiede che la classe che la implementa ridefini-

Algorithm 5.1 da `Mabon.sito`

```
def setSiteManager(self, sm):
        super( SitoMabon, self ).setSiteManager( sm )
        notify( EventoNuovoSitoMabon(self) )
```

sca i due metodi `getSiteManager` e `setSiteManager` per consentire l'attraversamento dell'oggetto SitoMabon sia verso i contenuti più semplici sia verso il registro locale dei componenti. La classe `SiteManagerContainer` fornisce già un'implementazione di tali metodi che viene estesa per generare un `EventoNuovoSitoMabon` che un oggetto SitoMabon è stato creato e aggiunto a un contenitore. Un gestore per tale evento provvederà a creare il nuovo registro dei componenti e ad aggiungervi e registrare tutte le utilità suddette. Il nuovo registro deve essere creato dopo che il suo contenitore associato è disponibile per consentirne la raggiungibilità: infatti il componente `Publisher` di Zope deve poter attraversare un oggetto contenitore per poter agganciare il registro locale a quello globale. Il suddetto concatenamento tra i gestori avviene nel metodo `SitoMabon.setSiteManager` tramite la chiamata della metodo setSiteManager della classe madre `SiteManagerContainer`, come mostrato nell'Algoritmo 5.1:

La classe `SitoMabon` genera quindi un evento definito dal programmatore.

5.2.3 L'oggetto EventoNuovoSitoMabon

5.2.3.1 Generale

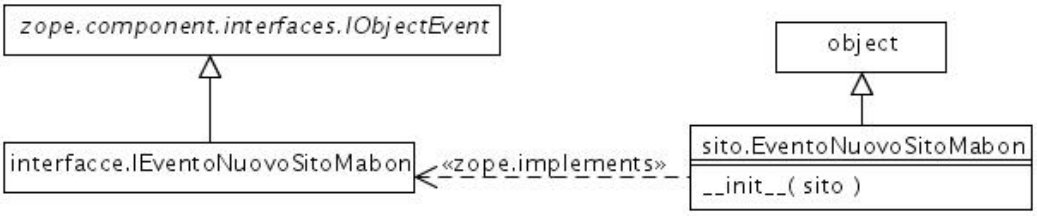
L'oggetto è un evento utilizzato per segnalare ad alcuni gestori di evento che un'istanza della classe `SitoMabon` è stata aggiunta al contenitore principale di Zope.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 5.4.

5.2.3.2 La classe IEventoNuovoSitoMabon

L'interfaccia deriva dall'interfaccia `IObjectEvent`, base per tutte le interfacce

Figura 5.4: Diagramma della classe `SitoMabon`



di eventi.

5.2.3.3 La classe EventoNuovoSitoMabon

Il costruttore della classe riceve un riferimento all'istanza della classe `SitoMabon` appena creata.

5.2.4 L'oggetto scheda

5.2.4.1 Generale

Le schede sono il cuore dell'applicazione Mabon, la cui natura è appunto quella di essere un software di catalogazione.

Un'oggetto scheda non rappresenta una scheda intesa come documento ufficiale nel gergo storico–artistico, ma l'insieme dei dati che descrivono l'oggetto a cui la scheda si riferisce; la validità giuridica dei dati in esso contenuti, come si vedrà, viene espresso dal valore di un apposito attributo modificabile solo da un validatore.

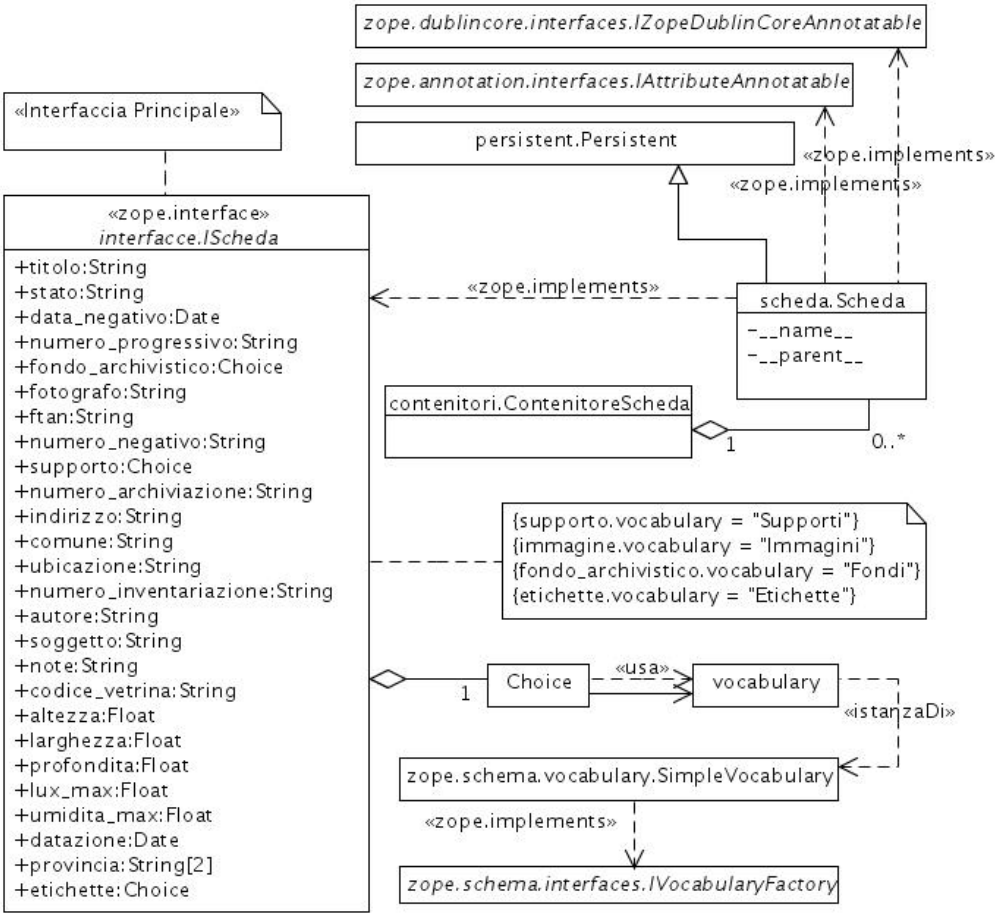
Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 5.5.

5.2.4.2 La classe `IScheda`

Come già illustrato in 3.1.2 nella pagina 28, una scheda è un oggetto che consta di quattro categorie di campi definiti nell'interfaccia IScheda come in figura precedente.

L'Interfaccia IScheda presenta anche due ulteriori campi – titolo e stato –

Figura 5.5: Diagramma della classe `Etichetta`



necessari al funzionamento del programma ma non propriamente facenti parte della categorizzazione storico–artistica:

- lo stato indica se una scheda è ancora una bozza, cioè non approvata da un validatore, o un documento effettivo;
- il titolo è una rappresentazione compatta della scheda nel formato "stato – autore – soggetto" che viene utilizzata ogni qualvolta sia necessaria una rappresentazione breve dell'oggetto, ad esempio in un elenco.

I tre campi etichette, fondo_archivistico e supporto di tipo `Choice` utilizzano rispettivamente i vocabolari "Etichette", "Fondi" e "Supporti" per offire all'utente le possibili opzioni da scegliere per compilare il campo. Tali vocabolari vengono creati da Zope tramite opportuni generatori specificati nel registro dei componenti.

`IScheda` è anche l'interfaccia primaria – cioè caratterizzante – un oggetto che la implementi.

5.2.4.3 La classe `Scheda`

La classe Scheda fornisce il componente di contenuto per l'oggetto scheda e implementa quanto specificato nell'interfaccia IScheda.

La derivazione da `Persistent` consente un'interazione con lo ZODB automatica e trasparente per il programmatore: la creazione, cancellazione e modifica delle istanze di Scheda saranno automaticamente inoltrate al database senza che il programmatore debba aggiungere ulteriori linee di codice, sebbene sia possibile modificare questo comportamento in caso di necessità.

Vengono aggiunti due nuovi attributi a quelli forniti dall'interfaccia: `__name__` e `__parent__`. Tali attributi sono necessari per le relazioni di contenimento tra

l'istanza dell'oggetto Scheda e l'istanza dell'oggetto ContenitoreScheda ed in generale per poter scorrere la catena delle relazioni di contenimento in entrambi i sensi come più dettagliatamente illustrato in 5.1.1.1 nella pagina 45.

L'interfaccia caratterizzante l'oggetto Scheda è come anticipato `IScheda`, ma ne sono fornite anche altre: `IAttributeAnnotatable`, `IZopeDublinCoreAnnotatable` e `IPersistent`. Le due interfacce `I*Annotatable` vengono sfruttate tramite degli adattatori presenti nelle librerie di Zope per registare informazioni non associabili ai campi propri di una scheda, ma che sono tuttavia utili, quali commenti dello schedatore e le date di creazione e modifica dell'istanza.

5.2.5 L'oggetto utente

5.2.5.1 Generale

L'oggetto utente contiene tutte le informazioni caratterizzanti un utente del sistema: anagrafica, informazioni prettamente informatiche, ruolo e i parametri per un'identificazione univoca e sicura da parte del programma.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 5.6.

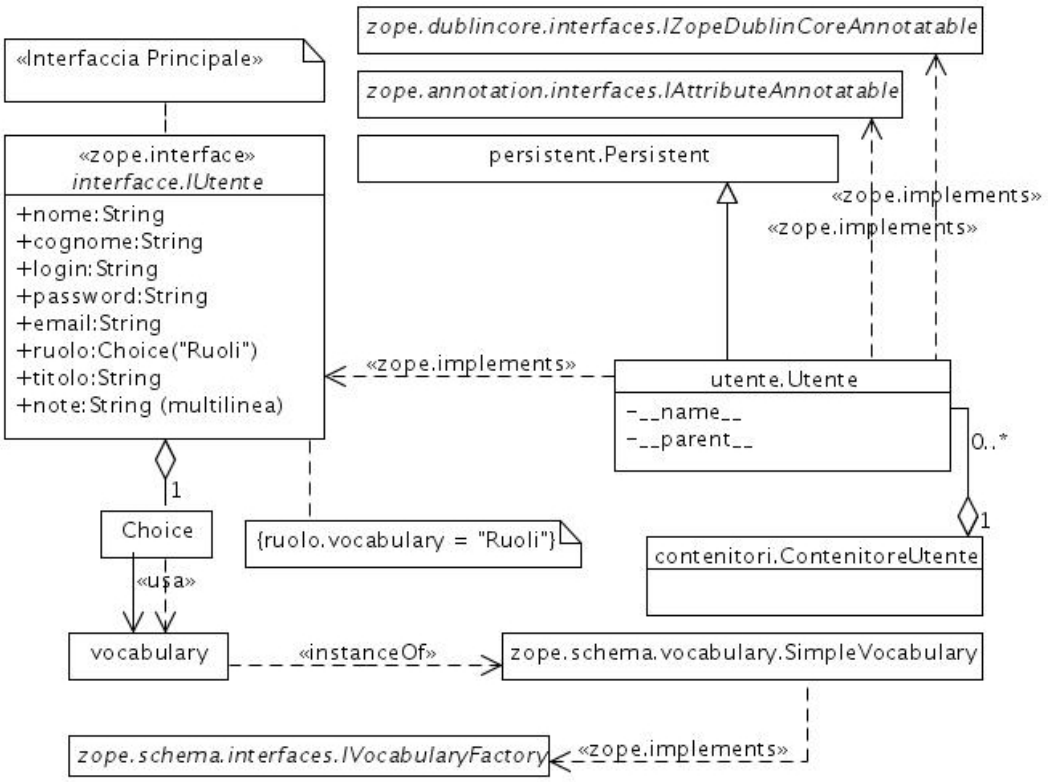
5.2.5.2 La classe IUtente

La sezione concernente l'anagrafica dell'utente è costituita dagli attributi nome e cognome, quella informatica da email, quella identificativa da login e password e infine il ruolo ricoperto dall'omonimo campo.

L'Interfaccia `IUtente` presenta anche l'ulteriore campo titolo necessario al funzionamento del programma:

- il titolo è una rappresentazione compatta dell'utente nel formato "login – cognome, nome" che viene utilizzata ogni qualvolta sia necessaria una rappresentazione breve dell'oggetto, ad esempio in un elenco.

Figura 5.6: Diagramma della classe `Utente`



Il campo ruolo di tipo `Choice` utilizza il vocabolario "Ruoli" per mostrare all'Amministratore le possibili scelte tra le quali selezionare il ruolo da far ricoprire all'utente. Tale vocabolario viene creato da Zope tramite un opportuno generatore specificato nel registro dei componenti.

`IUtente` è anche l'interfaccia primaria – cioè caratterizzante – un oggetto che la implementi.

5.2.5.3 La classe `Utente`

La classe Utente fornisce il componente di contenuto per l'oggetto utente e implementa quanto specificato nell'interfaccia `IUtente`.

La derivazione da `Persistent` consente un'interazione con lo ZODB automatica e trasparente per il programmatore: la creazione, cancellazione e modifica delle istanze di `Utente` saranno automaticamente inoltrate al database senza che

il programmatore debba aggiungere ulteriori linee di codice, sebbene sia possibile modificare questo comportamento in caso di necessità.

Vengono aggiunti due nuovi attributi a quelli forniti dall'interfaccia: `__name__` e `__parent__`. Tali attributi sono necessari per le relazioni di contenimento tra l'istanza dell'oggetto Utente e l'istanza dell'oggetto ContenitoreScheda ed in generale per poter scorrere la catena delle relazioni di contenimento in entrambi i sensi come più dettagliatamente illustrato in 5.1.1.1 nella pagina 45.

L'interfaccia caratterizzante l'oggetto utente è come anticipato `IUtente`, ma ne sono fornite anche altre: `IAttributeAnnotatable`, `IZopeDublinCoreAnnotatable` e `IPersistent`. Le due interfacce `I*Annotatable` vengono sfruttate tramite degli adattatori presenti nelle librerie di Zope per registare informazioni non associabili ai campi propri di un utente, ma che sono tuttavia utili, quali commenti dello amministratore e le date di creazione e modifica dell'istanza ovvero dell'account utente.

Capitolo 6

Altri componenti

6.1 Gestori di eventi, adattatori, viste, visualizzatori e generatori

6.1.1 Generale

L'interazione dei vari componenti principali e secondari del programma fra essi stessi e con quelli offerti da Zope è svolta da vari adattatori e gestori di eventi. In alcuni sono necessarie istanze di oggetti che vengono create da appositi generatori.

6.1.2 I gestori di eventi

Come esposto nell'introduzione un gestore di eventi è un adattatore – classe o funzione – tra un particolare interfaccia che definisce l'evento e l'interfaccia che offerta dall'istanza che ha subito l'azione rappresentata dall'evento.

Un gestore di eventi è anche uno speciale generatore poiché non ritorna alcuna istanza di un oggetto; quando un normale generatore non torna nulla al chiamante, Zope interpreta questa situazione come un errore.

Il programma prevede due gruppi di gestori di eventi:

- quelli adibiti alla aggiunta di utilità al gestore del sito e
- quelli dedicati alle istanze degli oggetti scheda e utente.

6.1.2.1 I gestori di aggiunta di utilità

6.1.2.2 La funzione `autenticazione_e_principali`

Questo gestore è un adattatore delle interfacce `ISitoMabon` e `IObjectAddedEvent`, filtra cioè gli eventi di "oggetti creati" i quali offrono l'interfaccia `ISitoMabon`.

Il suo compito è l'aggiunta al gestore del sito dei componenti critici per il funzionamento del sito Mabon nonché l'istanziazione di molti oggetti Contenitore*.

Vengono infatti create un'instanza di `ContenitoreUtenti`, una di `ContenitoreSchede` (per le schede con lo status di Documento) e una di un semplice `Folder` per contenere le aree degli utenti.

Viene creata e registrata l'utility `PAU`, il cui compito è gestire i vari moduli adibiti alla richiesta e all'estrazione dei dati (plugin delle credenziali) e all'autenticazione dell'utente (plugin di autenticazione); viene creato, registrato e attivato un contenitore per i principali (plugin di autenticazione) e vengono selezionate due plugin delle credenziali: una per gestire le sessioni `SessionCredential`, e l'altra per non far apparire il form di login a un utente che si sia già autenticato `NoChallengeIfAuthenticated`.

In Figura 6.1 sono rappresentate graficamente le attività svolte da questa funzione.

6.1.2.3 La funzione `cookie_e_sessioni`

Le aggiunte apportate da questo gestore di eventi al gestore di Mabon sono la creazione e registrazione di un gestore di cookie e la creazione e registrazione di

Figura 6.1: Diagramma di attività della funzione `autenticazione_e_principali`

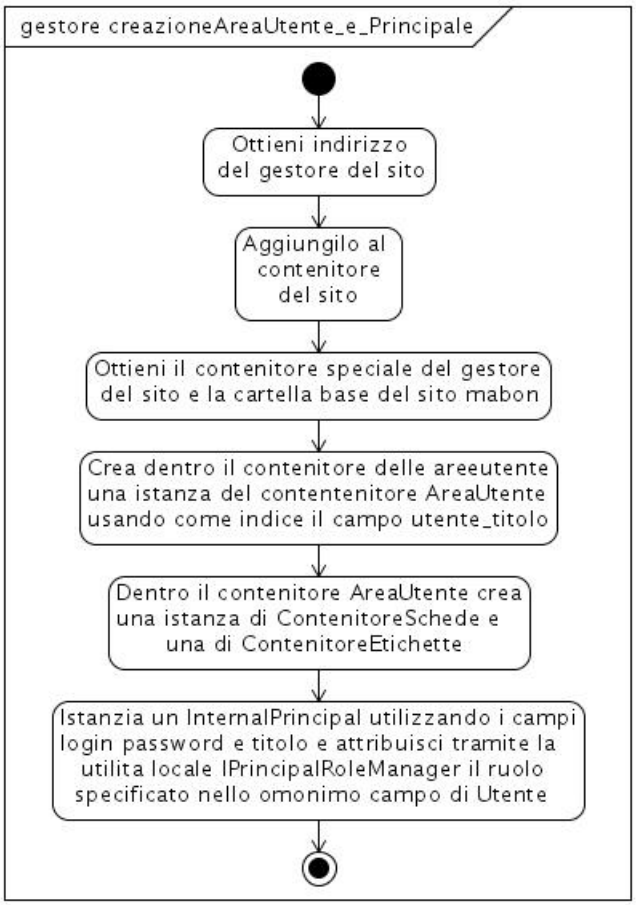


Figura 6.2: Diagramma di attività della funzione `cookie_e_sessioni`

un contenitore `Persistent` per i dati di sessione, entrambi necessari alla plugin per le Credenziali `SessionCredential` .

Sebbene tali aggiunte potrebbero essere inserite in coda al codice del precedente gestore `autenticazione_e_principali`, sono state poste in un gestore separato perché dipendono dalla configurazione del gestore del sito, ma non ne fanno propriamente parte. Ciò si evince anche dalla interfaccia adattata `INuovoSitoMabon`, per la quale viene appositamente generato un segnale dal metodo `SitoMabon.setSiteManager`, dopo la creazione del gestore del sito stesso.

In Figura 6.2 sono rappresentate graficamente le attività svolte da questa funzione.

6.1.2.4 La funzione `catalogo_e_indici`

L'ultimo gestore di questo gruppo è quello adibito all'aggiunta delle utilità che forniscono la possibilità di compiere ricerche all'interno del sito: utilità di identificazione numerica, il catalogo e la creazione degli indici: uno specifico per la ricerca semplice e una serie per la ricerca avanzata ciascuno dei quali dedicato all'indicizzazione di un campo espresso in `IScheda`[1].

[1]La ricerca avanzata non indicizza i campi relativi alle "Informazioni tecniche sul supporto

Algorithm 6.1 Indice per la ricerca semplice

```
catalogo[u'semplice'] = TextIndex(
        interface = ISearchableText,
        field_name = 'getSearchableText',
        field_callable = True
)
```

Algorithm 6.2 Indice per la ricerca avanzata

```
campi_da_omettere = ('__name__', '__parent__', 'titolo',
        'altezza', 'larghezza', 'profondita',
        'lux_max', 'umidita_max')
for campo in getFieldNames(IScheda):
        if not campo in campi_da_omettere:
                catalogo[campo] = TextIndex(
                        interface = IScheda,
                        field_name = campo,
                        field_callable = False
                )
```

In Figura 6.3 sono rappresentate graficamente le attività svolte da questa funzione.

Anche questo gestore come il precedente adatta l'interfaccia `INuovoSitoMabon` e per tale motivo il suo codice è separato da `autenticazione_e_principali`.

Vi è anche un altro motivo per questa separazione: nello stesso file `ricerca.py` che contiene l'adattatore, si trova anche la classe `TestoPerRicerca` che estrae il testo dalle istanze create e modificate di utente e scheda e aggiorna l'indice per la ricerca semplice. Il codice per l'istanziazione dell'indice semplice è mostrato nell'Algoritmo 6.1.

La generazione degli indici per la ricerca avanzata non necessita di un adattatore come quella semplice poiché Zope consente di specificare interfaccia e campo da utilizzare come modello e sorgente dei dati come rappresentato nell'Algoritmo 6.2.

Nel caso in cui si volessero creare ulteriori indici e quindi adattatori correlati,

fisico".

Figura 6.3: Diagramma di attività della funzione `catalogo_e_indici`

entrambe le porzioni di codice da modificare si troverebbero già nello stesso file.

In Figura 6.3 sono rappresentate graficamente le attività svolte da questa funzione.

6.1.2.5 I gestori degli eventi delle istanze di scheda e utente

6.1.2.6 La funzione `generaTitoloScheda`

Questo gestore viene attivato quando Zope genera un evento di oggetto creato `IObjectCreatedEvent` relativo ad un'istanza che offra l'interfaccia `IScheda`.

Compito del gestore è il corretto riempimento del campo titolo "stato – autore, soggetto" dell'oggetto scheda, utilizzato dall'adattatore `SincronizzaNomeScheda` usato da `ContenitoreSchede`; il titolo deve essere unico per l'intero sito, dato che una scheda promossa a "documento" viene spostata nel `ContentitoreSchede` principale: per questo motivo viene postfisso al titolo della scheda l'identificatore

univoco associato dell'istanza generato dall'adatattore preposto alla generazione degli `IntIds`[2].

6.1.2.7 La funzione `aggiornaTitoloScheda`

Similmente all'adattatore precedente, `aggiornaTitoloScheda` viene attivato quando Zope genera un evento di oggetto modificato `IObjectModificatoEvent` relativo ad un'istanza che offra l'interfaccia `IScheda`.

Compito del gestore è il corretto riempimento del campo titolo dell'oggetto Scheda, utilizzato dall'adattatore `SincronizzaNomeScheda` usato da ContenitoreScheda.

6.1.2.8 La funzione `generaTitoloUtente`

Questo gestore viene attivato quando Zope genera un evento di oggetto creato `IObjectCreatedEvent` relativo ad un'istanza che offra l'interfaccia `IUtente`.

Compito del gestore è il corretto riempimento del campo titolo "login – cognome, nome" dell'oggetto utente, utilizzato dall'adattatore SincronizzaNomeUtente usato da ContenitoreUtente.

6.1.2.9 La funzione `aggiornaTitoloUtente`

Similmente all'adattatore precedente, `aggiornaTitoloUtente` viene attivato quando Zope genera un evento di oggetto modificato `IObjectModificatoEvent` relativo ad un'istanza che offra l'interfaccia `IUtente`.

Compito del gestore è il corretto riempimento del campo titolo dell'oggetto utente, utilizzato dall'adattatore `SincronizzaNomeUtente` usato da `ContenitoreUtente`.

6.1.2.10 La funzione `creazioneAreaUtente_e_principale`

Questo gestore si attiva quando viene creato un nuovo Utente da parte del-

[2]A ciascun oggetto presente nello ZODB viene associato un Identificatore numerico, *IntID* o *IntegerID*, unico nell'intero database. Questa associazione avviene automaticamente se nel registro dei componenti del sito è presente l'utilità di generazione degli `IntIds`, che in Mabon viene aggiunta dal gestore `catalogo_e_indici`.

l'Amministratore, dato che adatta le interfacce `IUtente` e `IObjectAddedEvent`.

L'adattatore interagisce sia con la plugin di autenticazione sia con l'istanza "area utenti" della classe `Folder` creata dal gestore di eventi `autenticazione_e_principali`.

Alla plugin di Autenticazione – che ricordo è un'istanza di `PrincipalFolder`, quindi un contenitore – aggiunge un nuovo oggetto InternalPrincipal utilizzando i valori degli attributi dell'istanza `Utente` appena creata usando come indice il valore di `Utente.login`, scelta obbligata dal funzionamento di `PAU`. Al nuovo principale, tramite l'apposita utilità `PrincipalRoleManager`, assegna il livello di privilegio indicato dal valore dell'attributo ruolo dell'istanza di `Utente`.

All'istanza "area utenti" viene aggiunta un'istanza della classe `AreaUtente` e all'interno di quest'ultima un'istanza di `ContenitoreEt`ichette, per contenere le etichette private dell'utente, e una di `ContenitoreSchede`, per contenere le Schede create dall'Utente e non ancora approvate dal Validatore.

In Figura 6.4 sono rappresentate graficamente le attività svolte da questa funzione.

6.1.2.11 La funzione `cancellazioneAreaUtente_e_principale`

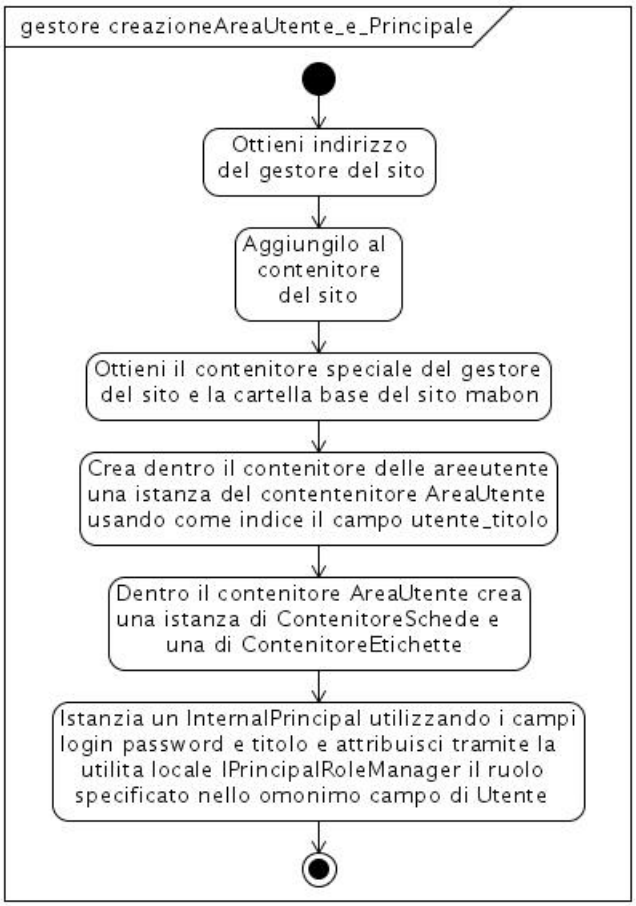
Questo gestore svolge le azioni opposte a quelle del precedente, come suggerito dall'interfaccia evento adattata `IObjectRemovedEvent`.

Vengono rimossa l'intera istanza dell'area utente con tutto il suo contenuto: le schede in stato di bozza e le etichette proprie dell'utente. Inoltre dalla plugin di autenticazione viene cancellato l'istanza di `InternalPrincipal` associata all'Utente.

6.1.2.12 Le funzioni `aggiornaDCda*`

I tre gestori `aggiornaDCdaScheda`, `aggiornaDCdaEtichetta` e `aggiornaDCdaUtente`, adattano un `IObjectModifiedEvent` a rispettivamente `IScheda`, `IEtichetta` e `IUtente`.

Figura 6.4: Diagramma di attività della funzione `catalogo_e_indici`



La loro funzione è quella di sincronizzare il valore del campo titolo della suite *DublinCore* con quello omonimo dell'interfaccia associata.

6.1.3 Adattatori

6.1.3.1 Generale

Le classi contenute in questa sottosezione sono degli adattatori che non svolgono anche la funzione di gestori di eventi, ma consentono di creare un sorta di via di comunicazione tra classi non progettate per dialogare fra loro.

L'oggetto o gli oggetti da adattare vengono passati al costruttore della classe se presente; nel caso in cui si abbia un solo parametro, per convenzione si usa nome "context".

6.1.3.2 Le classi `SincronizzaNome*`

Gli adattatori `SincronizzaNome*` derivano dalla classe `NameChooser`, che a sua volta implementa l'interfaccia `INameChooser`, e adattano una interfaccia `IContenitore*` secondo la tabella nella sezione 5.1.1.2.

La loro funzione è quella di permettere ad una istanza di un contenitore che implementi l'interfaccia `IContainerNamesContainer` di calcolare l'indice delle istanze contenute – al momento dell'inserimento nel contenitore – a partire da alcuni dati da ciascuna istanza.

La classe `NameChooser` impone che la classe derivata ridefinisca i metodi `checkName` e `chooseName`. Il compito del metodo `chooseName` in particolare è quello di implementare la logica di scelta della stringa dell'indice (il primo parametro passato alla funzione, valore di default) dell'oggetto contenuto (il secondo parametro) e ritornarla al chiamante.

È compito del codice presente in `IContainerNamesContainer` ricercare tramite il registro dei componenti un adattatore che implementi `INameChooser` e che adatti l'interfaccia relativa all'istanza dell'oggetto che sta per essere inserito nel contenitore.

6.1.3.3 Le classi `Dimensione*`

I tre adattatori `DimesioneUtente`, `DimensionePrincipale` e `Dimensione-Scheda` implementano l'interfaccia `ISized` e adattano rispettivamente le interfacce `IUtente`, `IInternalPrincipal` e `IScheda`.

Nelle visualizzazioni dei contenuti dell'istanza di un contenitore Zope offre la possibilità, ma non l'obbligo, di mostrare la grandezza e l'unità di misura di ciascuna istanza contenuta. Ciò viene realizzato tramite la ricerca automatica di un adattatore per ISized e l'interfaccia di ciascuna istanza contenuta.

Un adattatore per `ISized` deve ridefinire due metodi: `sizeForSorting` e `sizeForDisplay`. Il primo deve ritornare una tupla composta dalla stringa che specifica l'unità di misura (`byte`/`item`/`line`) e valore numerico della grandezza, per consentire la comparazione tra grandezze e quindi la possibilità di un algoritmo di ordinamento; il secondo deve ritornare una stringa per la visualizzazione dei suddetti dati in forma comprensibile per un essere umano.

6.1.3.4 La classe `TestoSchedaPerRicerca`

L'adattatore `TestoSchedaPerRicerca` consente di indicizzare un'istanza dell'oggetto scheda per la ricerca semplice.

Il componente viene cercato tramite il registro dei componenti ed utilizzato dal catalogo del sito ogni qual volta un'istanza di un oggetto, che offre la stessa interfaccia adattata dall'adattatore, viene aggiunta, modificata o cancellata.

La classe infatti adatta `IScheda` e implementa `ISearchableText`: `IScheda`, perché vogliamo effettuare delle ricerche sulle istanze della classe scheda presenti nel sito, e `ISearchableText` perché è l'interfaccia degli indici di testo utilizzati dal catalogo, come specificato dal gestore di eventi `catalogo_e_indice`.

I suddetti collegamenti sono evidenziati dal codice che li implementa, trascritto in Algoritmo 6.1.

6.1.3.5 L'adattatore con nome mabon.moduloscheda"

È un adattatore con nome "mabon.moduloscheda". La sua funzione è chiamare il metodo `__call__()` della classe `NamedTemplateImplementation` e adattarne il valore ritornato alle due classi `ModuloModificaScheda` e `ModuloAggiungiScheda`.

Il metodo `__call__`[3] a sua volta chiama il metodo `__call__` della classe `ViewPageTemplateFile` che ritorna il contenuto del file `modulo.pt`, file XML contenente tags TAL per la generazione del <form>[4] per la modifica dei valori dei campi di una istanza di un oggetto qualsiasi.

6.1.3.6 L'adattatore con nome "mabon.moduloutente"

È un adattatore con nome "mabon.moduloutente". La sua funzione è chiamare il metodo `__call__()` della classe `NamedTemplateImplementation` e adattarne il valore ritornato alle due classi)`ModuloModificaUtente` e `ModuloAggiungiUtente`.

Il metodo `__call__` a sua volta chiama il metodo `__call__` della classe `ViewPageTemplateFile` che ritorna il contenuto del file `modulo.pt`, file XML contenente tags TAL per la generazione del <form> per la modifica dei valori dei campi di una istanza di un oggetto qualsiasi.

6.1.4 Viste

6.1.4.1 Generale

Le viste sono i componenti responsabili della presentazione di un oggetto in maniera comprensibile per l'applicazione; data la definizione di applicazione utente in Zope, essi sono pertanto dei multi–adattatori con nome proprio tra la l'interfaccia IRequest dell'istanza dell'oggetto rappresentante la richiesta dell'utente, e l'interfaccia di ciascun oggetto coinvolto.

[3] `http://docs.python.org/reference/datamodel.html` e `http://www.python.org/doc/2.5.2/ref/callable-types.html`

[4] http://www.w3schools.com/tags/tag_form.asp

6.1.4.2 La classe `VisualizzaAzioni`

La classe `VisualizzaAzioni` è una vista per il contenitore SitoMabon. Tramite l'interazione col codice TAL presente nel file `azioni.pt` viene presentata all'utente, che si è appena autenticato, una lista di azioni condizionata al Ruolo dell'utente stesso.

Essendo un adattatore, alla classe `VisualizzaAzioni` viene passato il parametro context che contiene nel campo principal l'identificatore del pricipale autenticato.

Nel metodo `__call__`, partendo dall'identificatore del `Principale` autenticato contenuto nell'istanza globale della classe `Request` (`request.principal`), la funzione estrae il Ruolo assegnato al principale e ne assegna il valore alla variabile ruolo. Il codice TAL presente nel file `azioni.pt` tramite opportuni tags `tal:define` e `tal:condition` riferiti a `view/ruolo`, seleziona il blocco di azioni relativo al Ruolo.

Un diagramma di casi d'uso contenente le azioni offerte da Mabon è presente nel Capitolo 11, il manuale d'uso del programma.

6.1.4.3 La classe `VisualizzaScheda`

La classe `VisualizzaScheda` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente. L'unica condizione necessaria è che la classe derivata da BrowserPage implementi il metodo `__call__`. Quest'ultimo chiama il metodo `__call__` della classe `ViewPageTemplateFile` passando come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `vistascheda.pt`. `ViewPageTemplateFile` imposterà il Content–Type[5] per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

[5] http://mgrand.home.mindspring.com/mime.html

Il metodo `preparaNote` è necessario per elaborare il testo multilinea contenuto nel campo note dell'istanza dell'oggetto scheda e convertirlo in unformato XHTML compatibile: ad esempio i caratteri non consentiti da XML e quelli di formattazione come le andate a capo vengono trasformati in XML entities e tag `<br/>`[6] rispettivamente.

Il metodo `img` ritorna una stringa che contiene (con la sintassi dell'attributo `src` per il tag `img`[7] in XHTML) l'indirizzo dell'immagine associata alla scheda, se già importata da uno Scansionatore. Si tratta di una funzione contenente una chiamata alla funzione Python `os.path.join`, [8]`alla quale vengono passati` come argomenti: la cartella delle risorse immagini-importate (`++resource++`[9]`importate`), il contenuto del campo `supporto` della scheda adattata e infine quello del campo `FTAN`.

`VisualizzaScheda` adatta dunque l'interfaccia `IScheda` a `IBrowserPage`, consentendo così una visualizzazione delle istanze dell'oggetto scheda.

Il nome della vista per l'interfaccia `IScheda` è `index.html` ed è specificato nel file di configurazione configure.zcml: il publisher sarà ora in grado di risolvere il percorso `«istanzaScheda»/index.html`.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.7.

6.1.4.4 La classe `ModuloModificaScheda`

Questa classe deriva da `EditFormBase` la quale implementa – tra le altre – le interfacce `IForm`, `IBrowserPage` e `IBrowserView` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le con-

[6] http://www.w3schools.com/TAGS/tag_br.asp
[7] http://www.w3schools.com/TAGS/tag_img.asp
[8] La funzione join collega varie cartelle tra loro in base a una realazione di contenimento, inserendo il corretto separatore in funzione del sistema operativo (ad esempio / per sistemi Unix, Linux e Mac OS X, \ per sistemi Microsoft).
`join( 'a', 'b' )` genera `'a/b'` in FreeBSD e `'a\b'` in Windows.
[9] http://mail.zope.org/pipermail/zope3-users/2006-June/003679.html

sente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente.

La sua funzione è quella, partendo da una istanza dell'oggetto scheda, di visualizzare una pagina XHTML contenente i widget[10] necessari alla modifica dell'istanza, ricevere i dati inviati dall'utente e applicarli all'istanza, se il valore dell'attributo stato della scheda è diverso da "documento"; in caso contrario la pagina conterrà un messaggio informativo che avverte circa l'impossibilità di modificare un documento.

L'attributo `template`, tramite l'invocazione del metodo `__call__` della classe `NamedTemplate` che ricerca e utilizza l'adattatore "mabon.moduloscheda", contiene un'intero file XML con tags TAL che, una volta renderizzato con il contenuto dei campi `form_fields`, genererà la pagina XHTML che verrà mostrata all'utente.

L'attributo `form_fields` consente di estrarre dall'interfaccia `IScheda` gli attributi ivi definiti. Con tali informazioni la classe è in grado di renderizzare il codice TAL presente nel modello di pagina associato tramite l'attributo `template` per generare i campi per il <form> di input e renderizzare i widget XHTML più consoni ai tipi e alle limitazioni specificate nell'interfaccia passata come parametro alla classe `Fields`.

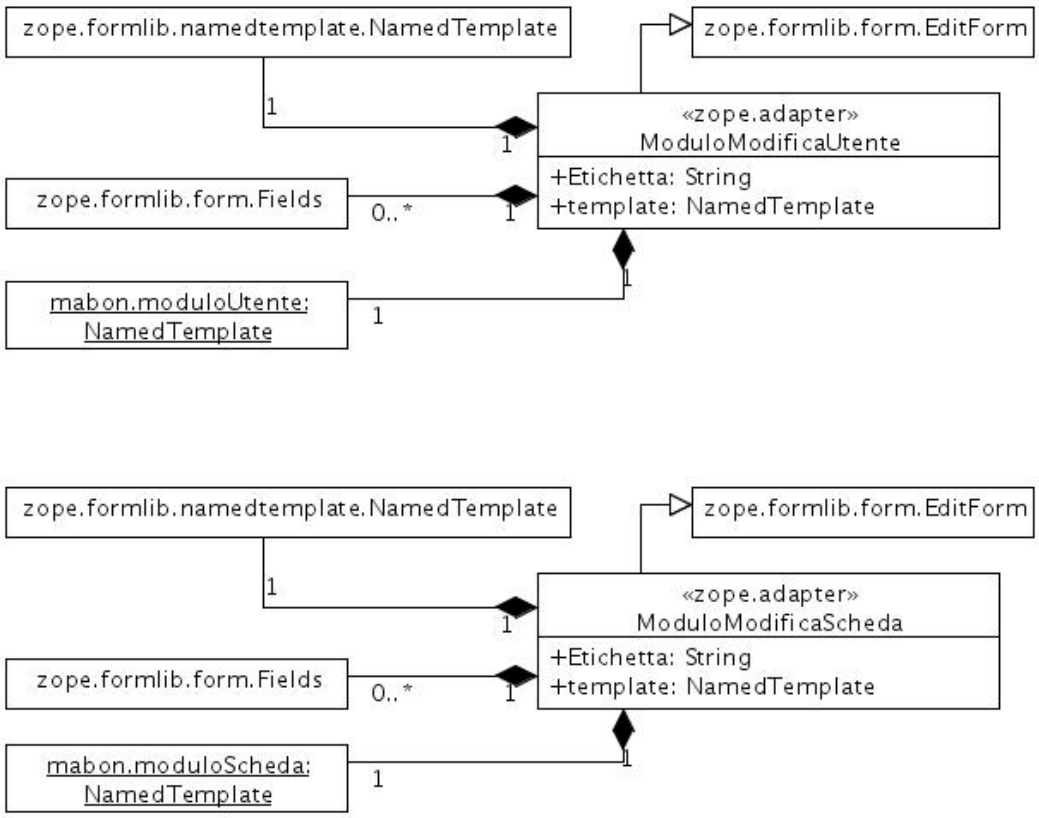
L'attributo `etichetta` contiene una stringa descrittiva per informare l'utente che sta modificando una scheda.

Vengono omessi i campi `__parent__` e `__name__` perché funzionali solo al contenimento, `titolo` perché – come abbiamo visto prima – viene aggiornato da alcuni gestori di eventi, e il campo `stato` che viene aggiornato in maniera particolare da Schedatori e Validatori.

L'attributo action del tag form nel codice XHTML generato contiene le istruzioni che consentono al componente Publisher di richiamare l'adattatore Mo-

[10] http://it.wikipedia.org/wiki/Widget

Figura 6.5: Diagrammi della classi `ModuloModificaScheda` e `ModuloModificaUtente`



duloModificaScheda per applicare – eseguendo il metodo applyChanges(istanza–context, schema_della_interfaccia, dati) – i valori digitati dall'utente all'istanza dell'oggetto scheda che è in corso di modifica.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.5.

6.1.4.5 La classe `ModuloAggiungiScheda`

Questa classe deriva da AddFormBase la quale implementa – tra le altre – le interfacce `IForm`, `IBrowserPage` e `IBrowserView` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente.

La sua funzione è quella di visualizzare una pagina XHTML contenente i

widget necessari alla modifica di una istanza dell'oggetto scheda, ricevere i dati inviati dall'utente, creare ex–novo un'istanza e copiarvi i dati.

L'attributo template, tramite l'invocazione del metodo `__call__` della classe `NamedTemplate` che ricerca e utilizza l'adattatore "mabon.moduloscheda", contiene un'intero file XML con tags TAL che, una volta renderizzato con il contenuto dei campi `form_fields`, genererà la pagina XHTML che verrà mostrata all'utente.

L'attributo `form_fields` consente di estrarre dall'interfaccia `IScheda` gli attributi ivi definiti. Con tali informazioni la classe è in grado di renderizzare il codice TAL presente nel modello di pagina associato tramite l'attributo `template` per generare i campi per il <form> di input e renderizzare i widget XHTML più consoni ai tipi e alle limitazioni specificate nell'interfaccia passata come parametro alla classe `Fields`.

L'attributo `etichetta` contiene una stringa descrittiva per informare l'utente che sta modificando una scheda.

Vengono omessi i campi `__parent__` e `__name__` perché funzionali solo al contenimento, `titolo` perché – come abbiamo visto prima – viene aggiornato da alcuni gestori di eventi, e il campo `stato` che viene aggiornato in maniera particolare da Schedatori e Validatori.

L'attributo action del tag form nel codice XHTML generato contiene le istruzioni che consentono al componente `Publisher` di richiamare l'adattatore `ModuloModificaScheda` per applicare [11] i valori digitati dall'utente all'istanza dell'oggetto scheda che è in corso di modifica.

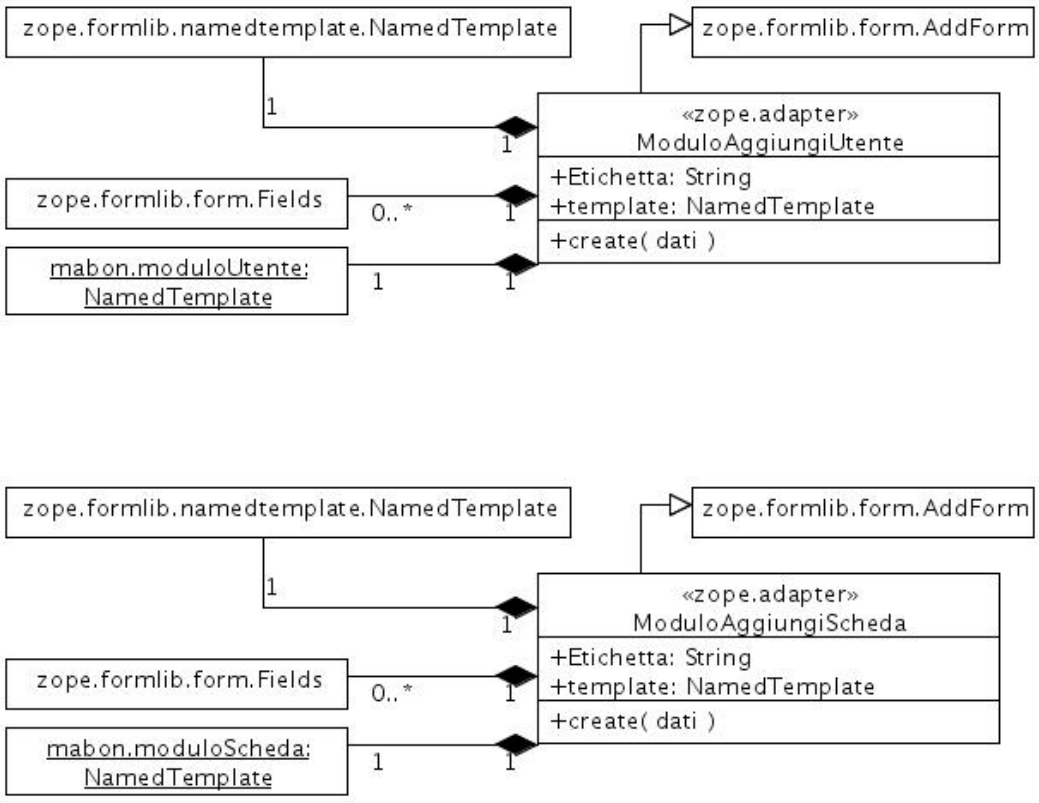
Il metodo `create( dati )` crea una nuova istanza della classe scheda, vi applica i valori digitati dall'utente e ritorna l'istanza. La classe `createObject`

[11] eseguendo il metodo `applyChanges( istanza-context, schema_della_intefaccia, dati )`

Algorithm 6.3 Il corpo del metodo `create( dati )`

```
def create(self, dati):
        scheda = createObject( u'Mabon.Scheda' )
        applyChanges( scheda, self.form_fields, dati )
        return scheda
```

Figura 6.6: Diagramma della classe `ModuloAggiungiUtente`



ritorna l'istanza creata dal generatore con nome "Mabon.Scheda" presente nel file `utente.py` e registrato nel file `configure.zcml`. Il corpo di create è mostrato in Algoritmo 6.3.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.6.

6.1.4.6 La classe `VisualizzaUtente`

La classe `VisualizzaUtente` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher`

indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente. L'unica condizione necessaria è che la classe derivata da `BrowserPage` implementi il metodo `__call__`. Quest'ultimo chiama il metodo `__call__` della classe `ViewPageTemplateFile` passando come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `vistautenti.pt`. ViewPageTemplateFile imposterà il Content–Type per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

Il metodo `preparaNote` è necessario per elaborare il testo multilinea contenuto nel campo note dell'istanza dell'oggetto scheda e convertirlo in unformato XHTML compatibile: ad esempio i caratteri non consentiti da XML e quelli di formattazione come le andate a capo vengono trasformati in XML entities e tag `<br/>` rispettivamente.

`VisualizzaUtente` adatta dunque l'interfaccia `IUtente` a `IBrowserPage`, consentendo così una visualizzazione delle istanze dell'oggetto scheda.

Il nome della vista per l'interfaccia `IUtente` è `index.html` ed è specificato nel file di configurazione `configure.zcml`: il publisher sarà ora in grado di risolvere il percorso `«istanzaUtente»/index.html`.

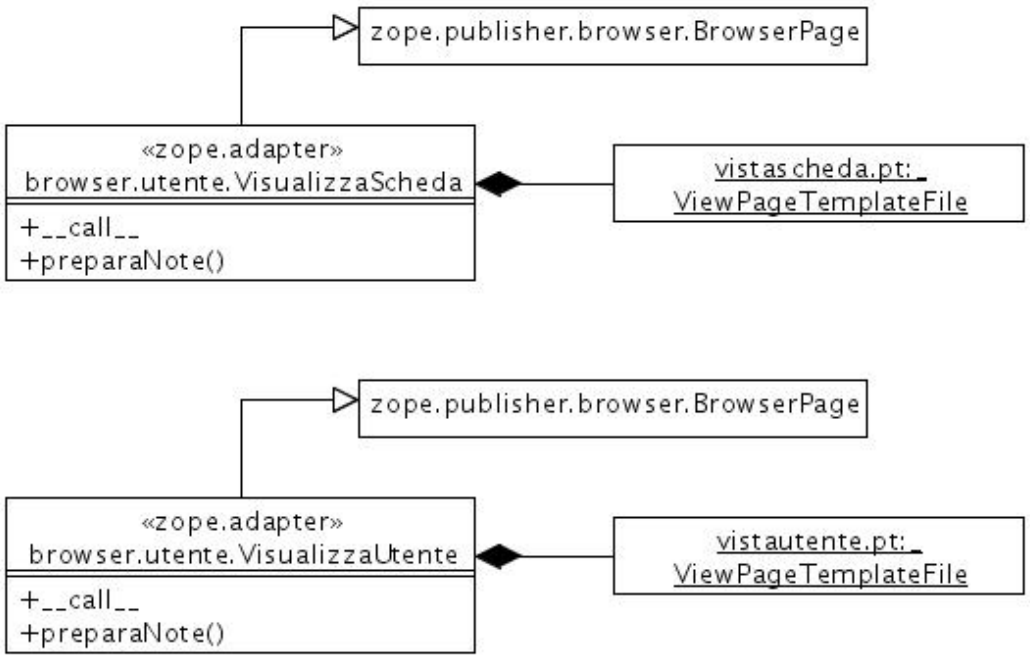
Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.7.

6.1.4.7 La classe `ModuloModificaUtente`

Questa classe deriva da EditFormBase la quale implementa – tra le altre – le interfacce `IForm`, `IBrowserPage` e `IBrowserView` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente.

La sua funzione è quella, partendo da una istanza dell'oggetto scheda, di

Figura 6.7: Diagramma della classi `VisualizzaScheda` e `VisualizzaUtente`



visualizzare una pagina XHTML contenente i widget necessari alla modifica dell'istanza, ricevere i dati inviati dall'utente e applicarli all'istanza.

L'attributo `template`, tramite l'invocazione del metodo `__call__` della classe `NamedTemplate` che ricerca e utilizza l'adattatore "Mabon.moduloutenti", contiene un'intero file XML con tags TAL che, una volta renderizzato con il contenuto dei campi `form_fields`, genererà la pagina XHTML che verrà mostrata all'utente.

L'attributo `form_fields` consente di estrarre dall'interfaccia IUtente gli attributi ivi definiti. Con tali informazioni la classe è in grado di renderizzare il codice TAL presente nel modello di pagina associato tramite l'attributo `template` per generare i campi per il <form> di input e renderizzare i widget XHTML più consoni ai tipi e alle limitazioni specificate nell'interfaccia passata come parametro alla classe `Fields`.

L'attributo `etichetta` contiene una stringa descrittiva per informare l'utente che sta modificando una utenti.

Vengono omessi i campi `__parent__` e `__name__` perché funzionali solo al contenimento, `titolo` perché – come abbiamo visto prima – viene aggiornato da alcuni gestori di eventi.

L'attributo `action` del tag form nel codice XHTML generato contiene le istruzioni che consentono al componente `Publisher` di richiamare l'adattatore `ModuloModificaUtente` per applicare – eseguendo il metodo `applyChanges( istanza-context, schema_della_interfaccia, dati )` – i valori digitati dall'utente all'istanza dell'oggetto scheda che è in corso di modifica.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.5.

6.1.4.8 La classe `ModuloAggiungiUtente`

Questa classe deriva da `AddFormBase` la quale implementa – tra le altre – le interfacce `IForm`, `IBrowserPage` e `IBrowserView` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente.

La sua funzione è quella di visualizzare una pagina XHTML contenente i widget necessari alla modifica di una istanza dell'oggetto scheda, ricevere i dati inviati dall'utente, creare ex–novo un'istanza e copiarvi i dati.

L'attributo `template`, tramite l'invocazione del metodo `__call__` della classe `NamedTemplate` che ricerca e utilizza l'adattatore "mabon.moduloscheda", contiene un'intero file XML con tags TAL che, una volta renderizzato con il contenuto dei campi `form_fields`, genererà la pagina XHTML che verrà mostrata all'utente.

L'attributo `form_fields` consente di estrarre dall'interfaccia `IUtente` gli attributi ivi definiti. Con tali informazioni la classe è in grado di renderizzare il codice TAL presente nel modello di pagina associato tramite l'attributo `template` per generare i campi per il <form> di input e renderizzare i widget XHTML più

Algorithm 6.4 Il metodo `create` della classe `ModuloAggiungiUtente`

```
def create(self, dati):
        utenti = createObject( u'Mabon.Utente' )
        applyChanges( utenti, self.form_fields, dati )
        return utenti
```

consoni ai tipi e alle limitazioni specificate nell'interfaccia passata come parametro alla classe `Fields`.

L'attributo `etichetta` contiene una stringa descrittiva per informare l'utente che sta modificando una scheda.

Vengono omessi i campi `__parent__` e `__name__` perché funzionali solo al contenimento, `titolo` perché – come abbiamo visto prima – viene aggiornato da alcuni gestori di eventi.

L'attributo action del tag form nel codice XHTML generato contiene le istruzioni che consentono al componente Publisher di richiamare l'adattatore `ModuloModificaUtente` per applicare – eseguendo il metodo `applyChanges( istanza-context, schema_della_intefacc dati )` – i valori digitati dall'utente all'istanza dell'oggetto scheda che è in corso di modifica.

Il metodo `create( dati )` crea una nuova istanza della classe scheda, vi applica i valori digitati dall'utente e ritorna l'istanza. La classe `createObject` ritorna l'istanza creata dal generatore con nome "mabon.utente" presente nel file `utente.py` e registrato nel file `configure.zcml`; l'implementazione del metodo create è disponibile nell'Algoritmo 6.1.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.6.

6.1.4.9 La classe `PaginaDiRicerca`

La classe `PaginaDiRicerca` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritor-

nare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente. L'unica condizione necessaria è che la classe derivata da `BrowserPage` implementi il metodo `__call__`. L'attributo `render` contiene un'istanza della classe `ViewPageTemplateFile`, la quale riceve come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `ricerca.pt`. Il metodo `__call__` della classe `ViewPageTemplateFile` imposterà il Content–Type per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

Il metodo `update( query )` viene eseguito dal metodo `__call__` e consente di preparare dei dati che serviranno agli algoritmi del metodo `__call__`. Il parametro `query` contiene la stringa di ricerca digitata dall'utente nel form di ricerca. Il corpo di `update` consiste in una ricerca del componente catalogo nel registro dei componenti e nella successiva ricerca dei dati inviati dall'utente tramite l'indice testuale.

Il metodo `__call__( query )`, oltre a chiamare il metodo `update( query )`, ritorna la pagina XHTML renderizzata dall'istanza della classe contenuta in render sulla base del modello XML con codice TAL; il contenuto della pagina è la lista delle schede che soddisfano i criteri di ricerca. Per ciascun risultato viene mostrata un piccola anteprima limitata, un link cliccabile e anche la percentuale di aderenza ai termini presenti nella stringa di ricerca.

Il metodo `getResultsInfo` è necessario per elaborare i risultati della ricerca – una lista di istanze – e ottenere tutta una serie di informazioni aggiuntive utili per una più gradevole visualizzazione. Tali informazioni non vengono tornate tutte insieme in una tupla con una istruzione return, ma tramite un generatore Python, quindi con l'istruzione `yield`[12], per non creare un picco di utilizzo al server. Per ciascun risultato della ricerca:

[12] `yield`: http://docs.python.org/reference/simple_stmts.html#the-yield-statement

- viene cercato un multi–adattatore che adatti l'interfaccia dell'istanza del risultato (`IScheda`) e la richiesta (`IBrowserRequest`) e implementi `IIconDirective` di nome "zmi_icon";
- viene estratto il titolo dell'istanza;
- viene estratta una preview limitata della stringa identificativa dell'oggetto nell'indice;
- e viene ritornata un vocabolario Python composto dai succitati campi e l'url dell'istanza.

`VisualizzaUtente` adatta dunque l'interfaccia `IUtente` a `IBrowserPage`, consentendo così una visualizzazione delle istanze dell'oggetto Scheda.

Il nome della vista per l'interfaccia `IUtente` è `index.html` ed è specificato nel file di configurazione configure.zcml: il publisher sarà ora in grado di risolvere il percorso `«istanzaUtente»/index.html`.

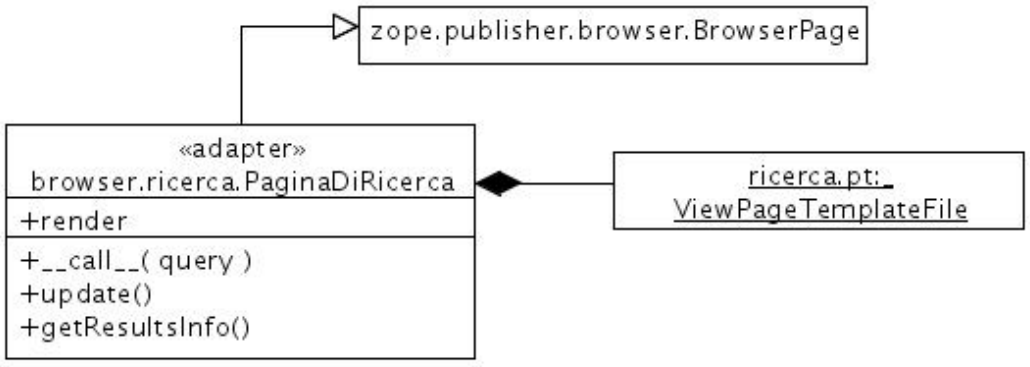
La vista `ricerca.html`, della quale `PaginaDiRicerca` è la classe associata nel tag ZCML `browser:page`, viene richiesta dal visualizzatore "ricerca".

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.8.

6.1.4.10 La classe `ModuloRicercaAvanzata`

La classe `ModuloRicercaAvanzata` è una classe derivante da `FormBase`. Come suggerisce il nome, `FormBase` è una classe utile nella creazione di viste che debbano generare una pagina XHTML contente un form con campi estratti da un'interfaccia e implementa – tra le altre – le interfacce `IForm`, `IBrowserPage` e `IBrowserView` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`.

Figura 6.8: Diagramma della classe PaginaDiRicerca



Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente.

La sua funzione è quindi quella, partendo dallo schema di attributi definiti da un'interfaccia, di visualizzare una pagina XHTML contenente i widget necessari alla modifica di una eventuale istanza di un oggetto che implementi tale interfaccia. I dati inseriti nei campi del form verranno inviati alla classe `PaginaDiRicercaAvanzata`.

L'attributo `template`, tramite l'invocazione del metodo `__call__` della classe `ViewPageTemplateFile`, contiene un'intero file XML con tags TAL che, una volta renderizzato con il contenuto dei campi `form_fields`, genererà la pagina XHTML che verrà mostrata all'utente. Il file contenente il modello di pagina usato è `moduloricerca.pt`.

L'attributo `form_fields` consente di estrarre dall'interfaccia IScheda gli attributi ivi definiti. Con tali informazioni la classe è in grado di renderizzare il codice TAL presente nel modello di pagina associato tramite l'attributo `template` per generare i campi per il <form> di input e renderizzare i widget XHTML più consoni ai tipi e alle limitazioni specificate nell'interfaccia passata come parametro alla classe `Fields`.

Vengono omessi i campi `__parent__` e `__name__` perché funzionali solo al

contenimento, `titolo` perché – come abbiamo visto prima – viene aggiornato da alcuni gestori di eventi.

L'attributo `etichetta` contiene una stringa descrittiva per informare l'utente che sta per effettuare una ricerca avanzata.

Poiché stiamo utilizzando un classe per generare un form "base", le spcifiche di BaseForm ci obbligano ad implementare almeno due ulteriori metodi: uno che provveda alla validazione dei dati immessi (o non immessi) dall'utente, il metodo `validate( self, action, data )`, e uno che permette all'utente di manipolare i dati immessi prima della validazione, il metodo `aggancio_azione_ricerca( self, action, data)`[13]. Poiché implementiamo i due metodi per adempiere alle richieste della classe base, il metodo `validate` torna una tupla[14] vuota, mentre il corpo del metodo `aggancio_azione_ricerca` è vuoto[15].

6.1.4.11 La classe `PaginaDiRicercaAvanzata`

La classe `PaginaDiRicercaAvanzata` è nelle funzionalità identica alla classe `PaginaDiRicerca`, ma variano i parametri dei metodi `__call__( self )` e `update( self, query )`, dove query è in questo caso un dizionario Python[16].

Il metodo `__call__` estrae da `self.request.form` la lista dei parametri di ricerca ricevuti dalla vista (il lettore mi consenta l'omissione delle interazioni utente-server-publisher-vista e viceversa) per la ricerca avanzata. Tali parametri vengono inseriti nel dizionario `self.query` nella forma `'nomeattributo=valore da cercare'` e il dizionario viene passato al metodo update che esegue la ricerca nel catalogo. Ciascun `nomeattributo` specifica anche l'indice omonimo nel quale cercare il corrispondente valore. Automaticamente l'utilità catalogo compone i

[13] Quest'ultimo verrà identificato dal decoratore Python @action(_(u'Ricerca')) definito in zope.formlib.form. Da notale che il parametro della action comparirà come testo del pulsante submit del form. Per una descrizione dettagliata dei decoratori si veda: http://www.python.org/dev/peps/pep-0318/

[14] Tipo di dato in Python, che consiste in una serie di valori non modificabili.

[15] O meglio contiene solo un'istruzione Python `pass`: http://www.network-theory.co.uk/docs/pytut/passStatements.html.

[16] Ossia un tipo dati contentente dei valori indicizzati.

risultati restituiti da ciascun indice quando si riferiscono allo stesso oggetto[17].

6.1.4.12 La classe `ModificaStatoScheda`

La classe `ModificaStatoScheda` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente. L'unica condizione necessaria è che la classe derivata da `BrowserPage` implementi il metodo `__call__`. L'attributo `render` contiene un'istanza della classe `ViewPageTemplateFile`, la quale riceve come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `statoscheda.pt`. Il metodo `__call__` della classe `ViewPageTemplateFile` imposterà il Content–Type per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

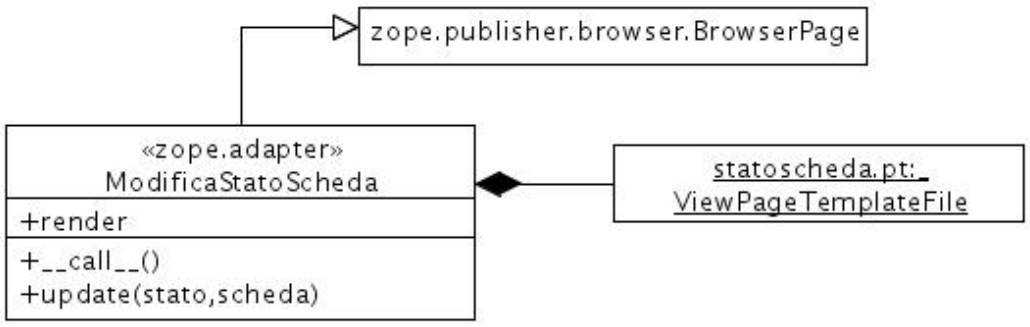
Il metodo `__call__()` estrae da `self.context.request` i paramentri ricevuti dal form dal visualizzatore `OpzioniModificaStato` e, dopo aver chiamato il metodo `update( stato, scheda )`, ritorna la pagina XHTML renderizzata dall'istanza della classe contenuta in render sulla base del modello XML con codice TAL; il contenuto della pagina è un breve testo che informa l'utente circa l'avvenuta modifica dello stato.

Il metodo `update( stato, scheda )` assegna il valore stato all'attributo stato dell'istanza scheda e, nei casi in cui si abbia una transizione dallo stato "proposta" a "documento" o da "documento" a "bozza", sposta la scheda nell'opportuno `ContenitoreSchede`, che ricordo è `/Mabon/schede/` per i documenti e `/Mabon/aree utenti/utente.login/schede/` per le bozze.

La vista `statoscheda.html`, della quale `PaginaDiRicerca` è la classe associata nel tag ZCML `browser:page`, viene richiesta dal visualizzatore "modificastato".

[17] Ricordo che ciascun oggetto presente nello ZODB è identificato da un Identificatore numerico, *IntID* o *IntegerID*.

Figura 6.9: Diagramma della classe `ModificaStatoScheda`



Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.9.

6.1.4.13 La classe `ElencaImmagini`

La classe `ElencaImmagini` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente. L'unica condizione necessaria è che la classe derivata da `BrowserPage` implementi il metodo `__call__`.

L'attributo `render` contiene un'istanza della classe `ViewPageTemplateFile`, la quale riceve come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `importaimmagini.pt`. Il metodo `__call__` della classe `ViewPageTemplateFile` imposterà il Content–Type per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

Il metodo `__call__()` si limita a chiamare il metodo `update()` e a restituire il codice ritornato dal metodo `__call__()` dell'istanza contenuta nell'attributo `render`.

Il metodo `update()` ricava dal vocabolario `form` della variabile globale `request`

il nome della cartella (è uno dei valori contenuti nel vocabolario "Supporti") nella quale si trovano i files `.jpg` contententi le immagini da importare; tale dato è contenuto nell'indice `cartella` (`request.form`['cartella']). Viene quindi implementata una ricerca di tutti i files `*.jpg` nella cartella `immagini/nuove/valore_form/`. I files così trovati vengono inseriti in nella variabile `self.immagini` di tipo `list` che viene utilizzata dal codice TAL presente nel file `importaimmagini.pt`.

Il codice TAL presente nel file `importaimmagini.pt` contiene, tra l'altro, un form che consente la selezione di ciascun file da importare. Il campo action del form punta alla vista `importaimmagini.html`. Il codice TAL accede alla lista `view/immagini` (la lista `self.immagini` esportata al codice TAL dalla classe `BrowserPage`) e per ciascun elemento genera una checkbox[18] con il nome del file e una piccola anteprima dell'immagine. Nel caso in cui sia già stata importata un'immagine con medesimo nome nella cartella passata alla classe, viene visualizzata anche l'anteprima di quest'ultima immagine, per consentire allo Scansionatore di valutare se importare o meno il file. Il campo form invia come campo nascosto anche il parametro `cartella`.

La vista `elencaimmagini.html`, della quale `ElencaImmagini` è la classe associata nel tag ZCML `browser:page`, viene richiesta da un form presente nella vista "azioni", il quale contiene un menù per la selezione della cartella di importazione.

6.1.4.14 La classe `ImportaImmagini`

La classe `ImportaImmagini` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato al-

[18] <input type="checkbox"...: http://www.w3schools.com/tags/tag_input.asp

l'utente. L'unica condizione necessaria è che la classe derivata da `BrowserPage` implementi il metodo `__call__`.

L'attributo `render` contiene un'istanza della classe `ViewPageTemplateFile`, la quale riceve come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `risultatoimportazione.pt`. Il metodo `__call__` della classe `ViewPageTemplateFile` imposterà il Content–Type per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

Il metodo `__call__()` si limita a chiamare il metodo `update()` e a restituire il codice ritornato dal metodo `__call__()` dell'istanza contenuta nell'attributo `render`.

Il metodo `update()` ricava dal vocabolario `form` della variabile globale `request` il nome della cartella (è uno dei valori contenuti nel vocabolario "Supporti") nella quale si trovano i files `.jpg` da importare e la lista degli stessi; tali dati sono contenuto nell'indice `cartella` e nei vari indici `importafile` (`request.form`['cartella'] e `request.form['importafile']`). Viene controllata la lunghezza del nome del file calcolando il numero di cifre (0-9) presenti da sinistra verso destra fermandosi al primo carattere differente (ad esempio '15r2.jpg' ha lunghezza due: '15'); se quest'ultima risulta inferiore a 3 caratteri, vengono aggiunti in testa degli '0'. Queste tre cifre (xyz) formano il prefisso di archiviazione[19]. Questo artificio viene utilizzato per evitare che una cartella rischi di contente più files di quel-

[19]Differenti filesystems hanno differenti limiti. Il più stringente tra quelli testati (reiserfs, ext3, ntfs, ufs è quello di FAT32, che consente al massimo 65.535 entrate in una cartella. L'effettivo consumo di queste entrate e quindi l'assegnazione a ciascun file o cartella contenuta varia in base alla lunghezza del nome.

Ntfs e ReiserFS impongono 4.294.967.295 (2^{32} - 1) files in ciascun drive.

Ext3 e UFS impongono dei limiti in funzione del numero di inodes ma meno stringenti di FAT32.

FAT32: http://www.microsoft.com/whdc/system/platform/firmware/fatgendown.mspx e http://en.wikipedia.org/wiki/File_Allocation_Table

NTFS: http://en.wikipedia.org/wiki/Ntfs

ReiserFS: http://en.wikipedia.org/wiki/ReiserFS

Ext3: http://en.wikipedia.org/wiki/Ext3

UFS: http://en.wikipedia.org/wiki/Unix_File_System

li consentiti dal filesystem. Ciascun file indicizzato viene copiato dalla cartella `immagini/nuove/cartella/` alla cartella `immagini/nuove/cartella/xyz/` con l'eventuale nuovo nome ottenuto aggiungendo degli zeri.

Il codice TAL presente nel file `importaimmagini.pt` visualizza l'esito dell'importazione e contiene un link alla vista `azioni.html`.

La vista `importaimmagini.html è associata alla classe ImportaImmagini` tramite un tag ZCML `browser:page`.

6.1.4.15 La classe Azioni

La classe `Azioni` deriva dalla classe `BrowserPage`, che implementa – tra le altre – le interfacce `IBrowserPage` direttamente e `IBrowserPublisher` indirettamente tramite `IBrowserPage`. Questa sua genealogia le consente di ritornare al componente `Publisher` l'output dell'applicazione che poi verrà mostrato all'utente. L'unica condizione necessaria è che la classe derivata da `BrowserPage` implementi il metodo `__call__`.

L'attributo `render` contiene un'istanza della classe `ViewPageTemplateFile`, la quale riceve come parametro il nome del file che contiene un modello della pagina in XHTML, in questo caso `azioni.pt`. Il metodo `__call__` della classe `ViewPageTemplateFile` imposterà il Content–Type per l'intestazione HTTP e nel corpo della risposta porrà il contenuto del file dopo averlo renderizzato.

Il metodo `__call__()` si limita chiamare il metodo `update()` e a restituire il codice ritornato dal metodo `__call__()` dell'istanza contenuta nell'attributo `render`.

Il metodo `update()` utilizzando l'attributo `principal` della variabile globale `request` e l'utilità locale `PrincipalRoleManager`[20], pone nella variabile `self.ruolo` il Ruolo del principale autenticato.

[20]La funzione di questa utilità è quella di consentire di ottenere (tra l'altro) una lista dei Ruoli associati ad un Principale e viceversa.

Il file `azioni.pt` contiene vari forms che consentono le azioni disponibili ai vari ruoli, quali creare un utente, una scheda, apporre delle modifiche importare delle immagini, ecc. Ciascun form è "inscatolato" da dei blocchi TAL condizionali (`tal:condition`), che ne consentono la visualizzazione solo se il contenuto di `view/ruolo` (la variabile `self.ruolo` esportata al codice TAL) combacia con quello specificato blocco TAL.

La vista `azioni.html è associata alla classe Azioni` tramite un tag ZCML `browser:page`. La classe `Azioni` è anche una vista per l'interfaccia `ISitoMabon`.

6.1.5 I visualizzatori

6.1.5.1 Generale

I visulizzatori sono dei gestori di contenuto nella forma di multi–adattatori tra la richesta effettuata dall'utente, una vista, un'interfaccia (contenuto) e la skin (rappresentanta dal gestore di visualizzatori).

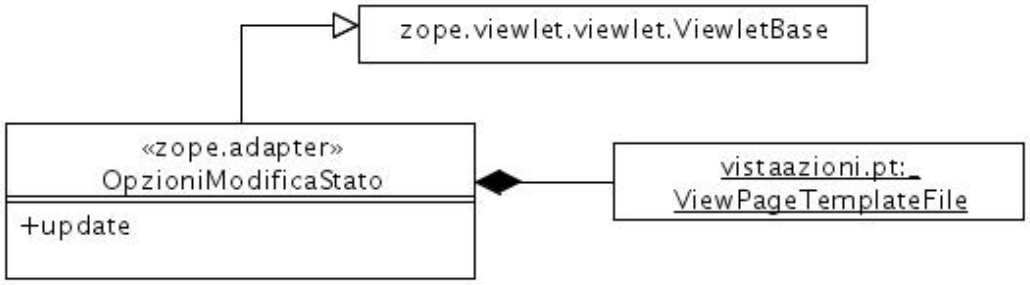
In Mabon vi sono due visualizzatori particolarmente significativi: quello per immettere i termini di ricerca su una scheda e quello per la modifica dello stato (`Scheda.stato`) di un'istanza dell'oggetto Scheda.

6.1.5.2 Il visualizzatore di ricerca

La funzione del visualizzatore "ricerca" è quello di fornire uno strumento per cercare delle istanze dell'oggetto scheda in base al testo contenuto negli attributi.

L'implementazione di questo visualizzatore è realizzata tramite un blocco di codice XHTML contenuto nel file `skins/ricerca.pt` contenente un tag form con attributo action contenente la vista `ricerca.html` e il campo per la digitazione del testo da cercare.

Figura 6.10: Diagramma della classe `OpzioniModificaStato`



6.1.5.3 La classe `OpzioniModificaStato` e il relativo visualizzatore

La funzione del visualizzatore "modificastato" è quello di consentire la modifica del valore dell'attributo stato dell'istanza della scheda visualizzata.

Una rappresentazione delle relazioni tra l'interfaccia e la implementazione si trova in Figura 6.10.

La classe `OpzioniModificaStato` deriva dalla classe `ViewletBase` e tramite il metodo `update`, chiamato come per le viste in automatico prima della renderizzazione del codice nell'istanza dell'oggetto contenuta nell'attributo `render`, riempie l'attributo `self.menu` in base al valore dell'istanza dell'oggetto Scheda e al Ruolo del principale autenticato, secondo la Tabella 6.1.

Nel caso in cui il codice XHTML restituito non sia una stringa vuota, il form contiene una combobox[21] con i nuovi possibili stati (campo `stato`) e un campo nascosto con l'IntId dell'istanza (campo `scheda`). Il campo action del tag form contiene "`@@statoscheda.html`", il nome della vista delegata alla modifica dello stato.

I valori ammissibili per il campo menu sono gli stessi presenti nel diagramma di stato per il campo stato di un'istanza dell'oggetto Scheda.

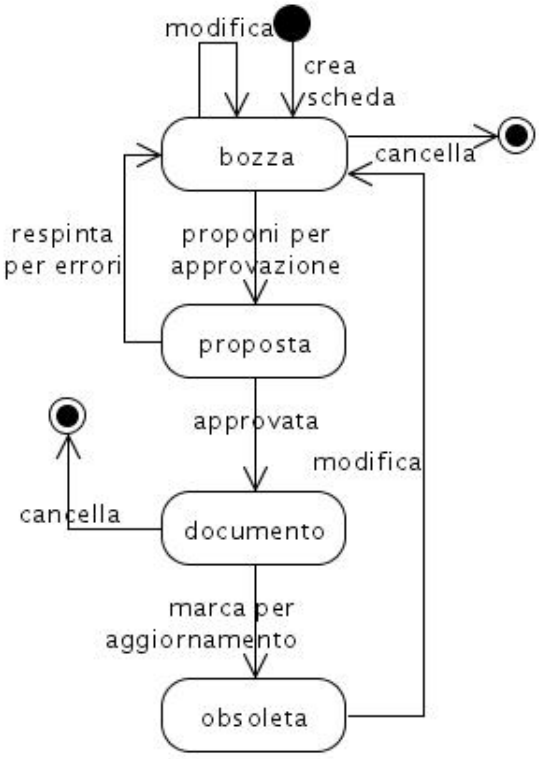
Un riferimento all'istanza dell'oggetto Scheda è recuperabile tramite `self.context`, mentre l'identificativo del principale autenticato da `self.request.principal`.

[21] http://it.wikipedia.org/wiki/Combo_box

Tabella 6.1: Tabella delle azioni che modificano lo stato di una scheda

stato	ruolo	menu	codice XHTML ritornato
bozza	Schedatore	proponi	form con pulsante “proponi per approvazione”
bozza	Schedatore	cancella	form con pulsante “cancella”
proposta	Validatore	valida	form con radiobutton “approva / rigetta”
obsoleta	Schedatore	importa	form con pulsante “modifica”
documento	Validatore	modifica	form con pulsante “importa”
documento	Validatore	cancella	form con pulsante “cancella”
qualsiasi altro	qualsiasi altro	stringa vuota	stringa vuota

Figura 6.11: Stati di una scheda



Tramite l'utilità locale `PrincipalRoleManager(principal)` è possibile ottenere il Ruolo del `Principal` autenticato.

6.1.6 I generatori

6.1.6.1 Generale

I generatori in Mabon sono delle utilità, cioè dei componenti che ritornano un'istanza di un oggetto e che possono opzionalmente avere un nome.

Possono verificarsi due possibilità:

- se si dispone già di una istanza, si deve registrarla come componente (attributo `component` al tag `utility` nei files `configure.zcml`) – è il caso dei generatori dei vocabolari;
- se si dispone di una classe, funzione o di qualsiasi altra cosa che deve essere chiamata per creare l'utilità, si deve registrarla come `Factory` (attributo `factory` al tag `utility` nei files `configure.zcml`) o generatore propriamente detto.

6.1.6.2 La classe `generatoreScheda`

La classe `generatoreScheda` è una `Factory` (generatore), che ritorna un'istanza di un oggetto Scheda ed è caratterizzata dai due attributi titolo e descrizione che forniscono indicazioni sull'attività di questo componente.

6.1.6.3 La classe `generatoreUtente`

La classe `generatoreUtente` è una `Factory` (generatore), che ritorna un'istanza di un oggetto Utente ed è caratterizzata dai due attributi titolo e descrizione che forniscono indicazioni sull'attività di questo componente.

6.1.6.4 La funzione `vocabolarioRuoli`

La funzione vocabolarioRuoli fornisce l'interfaccia `IVocabularyFactory` (generatore di vocabolario) e non utilizza il parametro `context`.

L'algoritmo utilizzato per la generazione del vocabolario richiede al registro dei componenti la lista di tutte le utilità che offrono l'interfaccia `IRole` e il cui attributo identificativo inizi con la stringa "mabon.". Tali utilità vengono collezionati i una lista che viene utilizzata per costruire il vocabolario "Ruoli".

6.1.6.5 La funzione `vocabolarioFondi`

La funzione vocabolarioFondi fornisce l'interfaccia `IVocabularyFactory` (generatore di vocabolario) e non utilizza il parametro `context`.

Nel corpo della funzione è presente una lista con gli otto possibili fondi archivistici. Tale lista viene utilizzata per costruire il vocabolario "Fondi".

6.1.6.6 La funzione `vocabolarioImmaginiNuove`

La funzione `vocabolarioImmaginiNuove` fornisce l'interfaccia `IVocabulary-Factory` (generatore di vocabolario) e non utilizza il parametro `context`.

La funzione elenca tutti i files `.jpg` presenti nella cartella `immagini/nuove`; da tale elenco viene costituito il vocabolario "Immagini Nuove".

6.1.6.7 La funzione `vocabolarioImmaginiImportate`

La funzione vocabolarioImmaginiImportate fornisce l'interfaccia `IVocabulary-Factory` (generatore di vocabolario) e non utilizza il parametro `context`.

La funzione elenca tutti i files `.jpg` presenti nella cartella `immagini/importate`; da tale elenco viene costituito il vocabolario "Immagini Importate".

6.1.6.8 La funzione `vocabolarioSupporti`

La funzione `vocabolarioSupporti` fornisce l'interfaccia `IVocabularyFactory` (generatore di vocabolario) e non utilizza il parametro `context`.

Nel corpo della funzione è presente una lista con i nove possibili supporti per

il materiale fotografico. Tale lista viene utilizzata per costruire il vocabolario "Supporti".

6.1.6.9 La funzione `vocabolarioEtichette`

La funzione vocabolarioEtichette fornisce l'interfaccia `IVocabularyFactory` (generatore di vocabolario) e, a differenza dei generatori precedenti utilizza il parametro `context` per ottenere dal campo `principal`, l'identificatore del pricipale autenticato.

Partendo dall'identificatore, la funzione estrae il `login` corrispondete e crea una lista con tutti i nomi delle istanze della classe `Etichetta` presenti nel contenitore `gestoredelsito/aree utenti/utente.login/etichette/`. Tale lista viene utilizzata per costruire il vocabolario "Supporti".

6.2 Codice e componenti client-side

6.2.1 La visualizzazione dell'immagine associata alla scheda

Vi sono tre componenti che necessitano di accedere e visualizzare le immagini contenute in Mabon ma non operano attivamente su di esse: `ModuloAggiungiScheda`, `ModuloModificaScheda`, `VisualizzaScheda`.

Nel caso di `VisualizzaScheda`, data la natura statica della vista, è sufficiente che il metodo `img` generi una stringa contenente un riferimento all'immagine, la quale viene poi incorporata in un tag `img/src` nel codice XHTML visualizzato dall'utente.

Nel caso di `ModuloModificaScheda` e `ModuloAggiungiScheda` la situazione è più complessa. La natura delle pagine generate dai due `Modulo*Scheda` è dinamica e quindi il codice che consente la visualizzazione deve agire a tempo di esecuzione sul browser dell'utente. La soluzione in questo caso è del codice ECMAscript[22]

[22]http://it.wikipedia.org/wiki/ECMAScript

inserito o collegato al file `moduloscheda.pt`. Nella pagina è inserito un pulsante che, quando premuto, attiva una funzione ECMAscript `visualizza_immagine` (tag `<button type=''button'' onclick=''visualizza_immagine()''>`[23]), la quale visualizza l'immagine in un'apposita area della pagina (un tag `img` con `name` e `id` di valore "anteprima" e con visibilità iniziale `visibility=''hidden`"[24]). L'accesso all'area avviene tramite un riferimento DOM[25].

La visualizzazione dell'immagine non è obbligatoria, poiché, una volta compilati i campi `ftan` e `supporto_fisico` della scheda, l'immagine associata è automaticamente determinata:
il file `immagini/importate/«supporto_fisico»/ftan.jpg`[26].

Però l'esperienza con Mabon I ha dimostrato che lo Schedatore gradisce poter visualizzare l'immagine associata alla scheda. Vi è anche un ulteriore vantaggio nel consentire la visualizzazione: un eventuale errore di attribuzione del nome al file `.jpg` dell'immagine o di compilazione dei campi `ftan` o `supporto_fisico`, può essere scoperto dallo Schedatore direttamente in fase di compilazione della scheda e non di successiva visualizzazione.

[23] Per una descrizione del tag `button`: http://www.w3schools.com/tags/tag_button.asp
Per una descrizione dell'evento `onclick`: http://www.w3schools.com/tags/ref_eventattributes.asp

[24] `visibility` è un tag CSS. Visibilità di un elemento: http://www.w3schools.com/css/pr_class_visibility.asp

[25] Posto che l'area apposita sia un tag `<img id=''anteprima''...`, il riferimento DOM sarà del tipo `document.getElementbyId("anteprima")`. HTML DOM: http://www.w3schools.com/js/js_obj_htmldom.asp

[26] Il valore `ftan` deve essere trattato per soddisfare i vincoli esposti in 6.1.4.14.

Capitolo 7

Localizzazione dei componenti nei files

7.1 Generale

Dopo aver descritto le classi e le funzioni che implementano i vari componenti dell'applicazione Mabon, segue una lista visuale costituita da tutte le figure del capitolo 7 per localizzare i componenti nei files che li contengono.

Figura 7.1: Diagramma di struttura per i componenti `AreaUtente`, `ContenitoreUtenti`, `ContenitoreScheda`, `ContenitoreEtichetta`, `SincronizzaNomeUtente`, `SincronizzaNomeScheda` e `SincronizzaNomeEtichetta`.

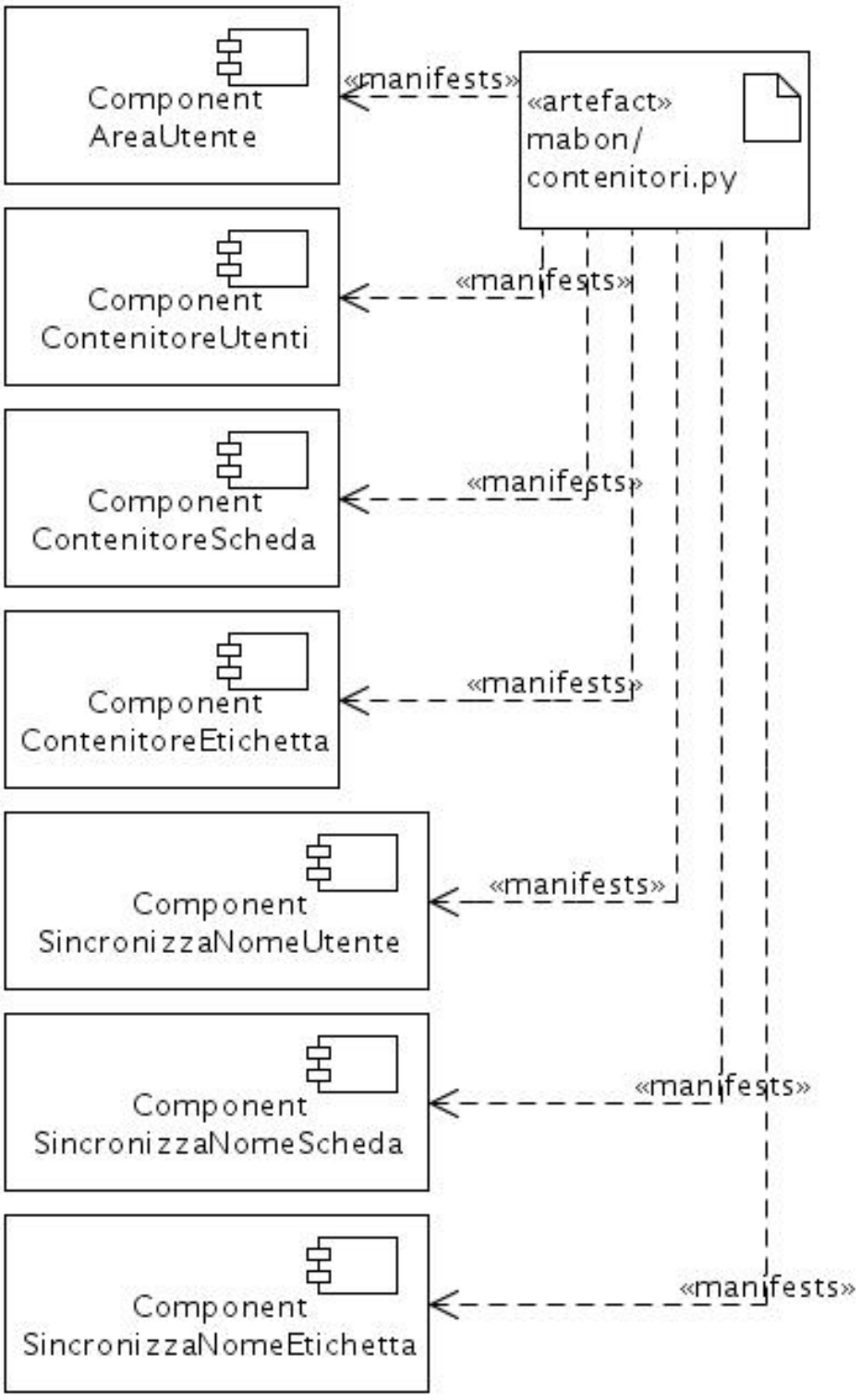


Figura 7.2: Diagramma di struttura per i componenti `Dimensione Principale`, `DimensioneScheda` e `DimensioneUtente`.

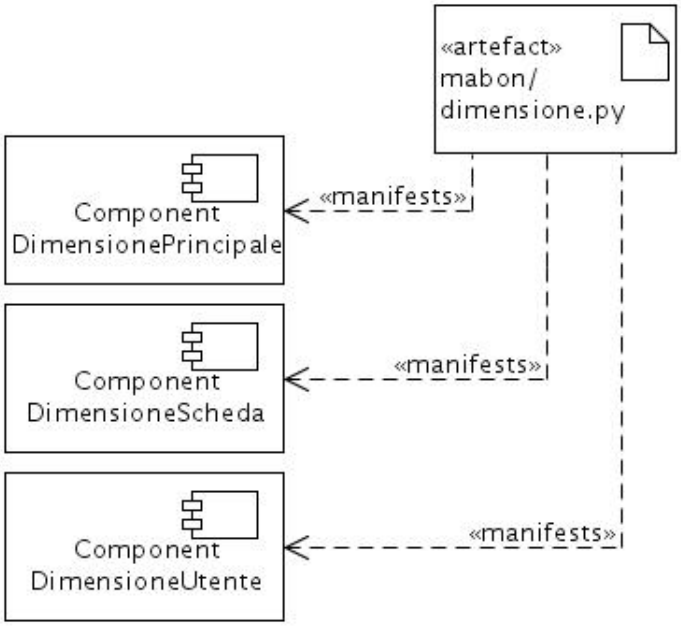


Figura 7.3: Diagramma di struttura per i componenti `ElencaImmagini` e `ImportaImmagini`.

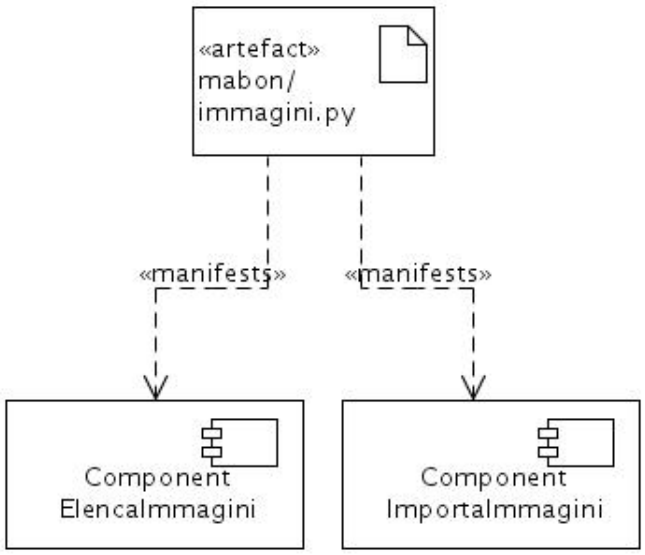


Figura 7.4: Diagramma di struttura per i componenti `aggiornaDCdaEtichetta`, `aggiornaDCdaScheda` e `aggiornaDCdaUtente`.

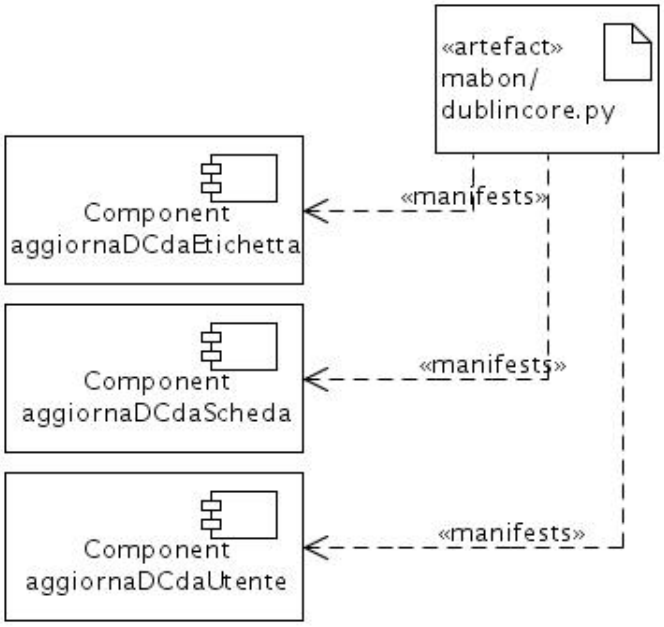


Figura 7.5: Diagramma di struttura per i componenti `vocabolarioEtichette` e `vocabolarioFondi`.

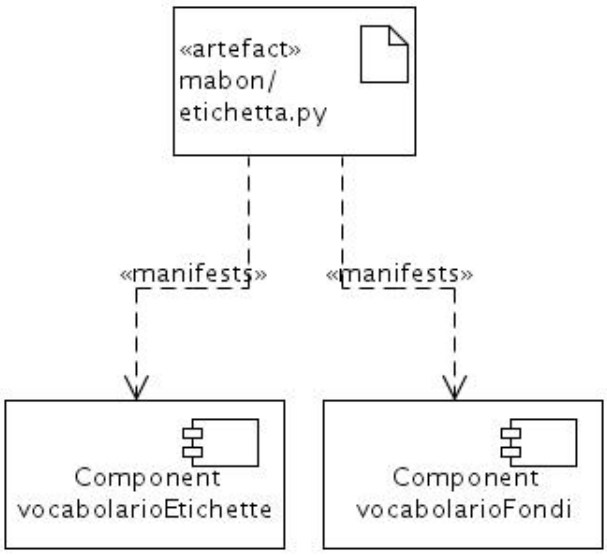


Figura 7.6: Diagramma di struttura per i componenti `generaTitoloScheda`, `aggiornaTitoloScheda`, `generatoreScheda`, `Scheda`, `VisualizzaScheda`, `ModuloModificaScheda` e `ModuloAggiornaScheda`.

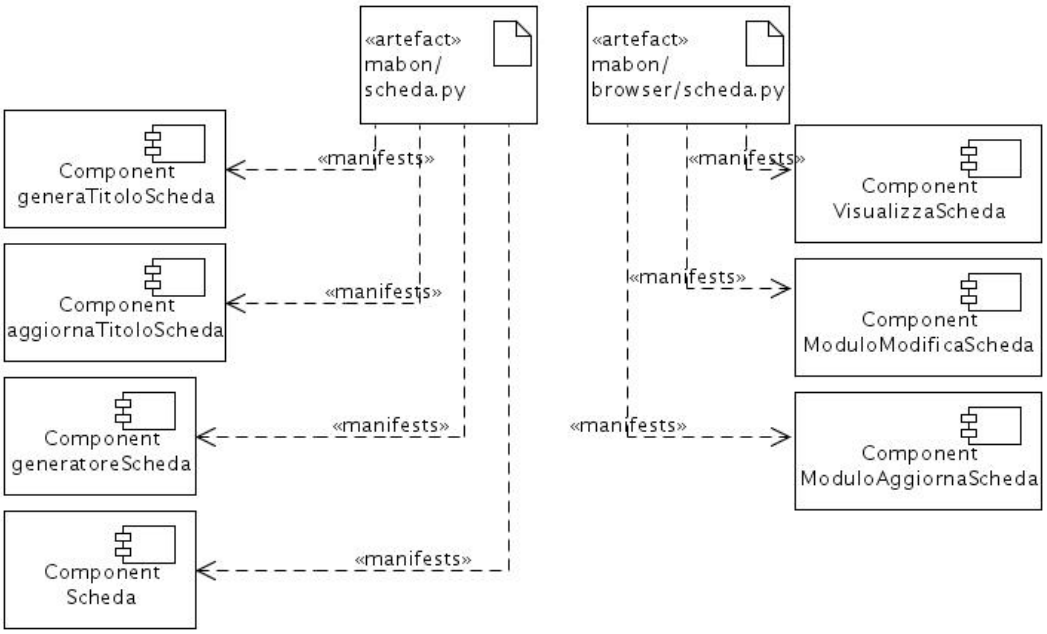


Figura 7.7: Diagramma di struttura per i componenti `generaTitoloUtente`, `aggiornaTitoloUtente`, `generatoreUtente`, `Utente`, `VisualizzaUtente`, `ModuloModificaUtente` e `ModuloAggiornaUtente`.

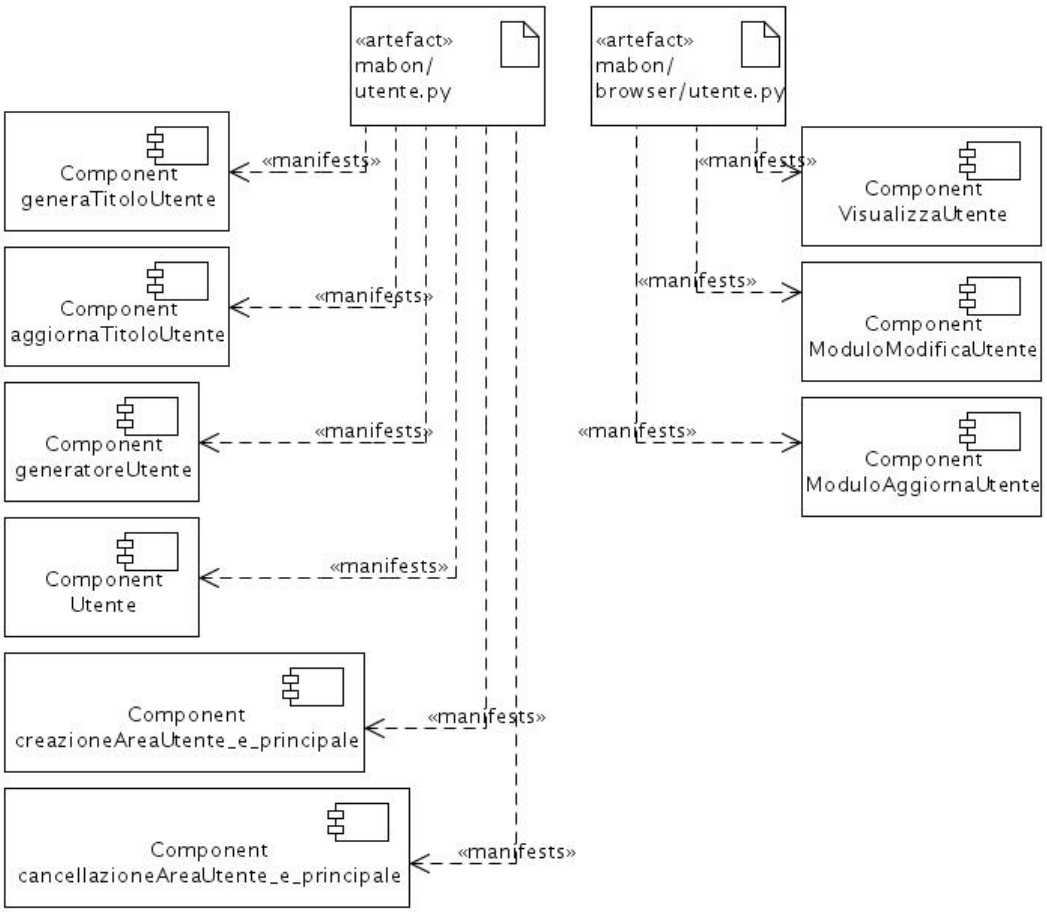


Figura 7.8: Diagramma di struttura per i componenti `SitoMabon`, `EventoNuovoSitoMabon`, `autenticazione_e_principali` e `cookie_e_sessioni`.

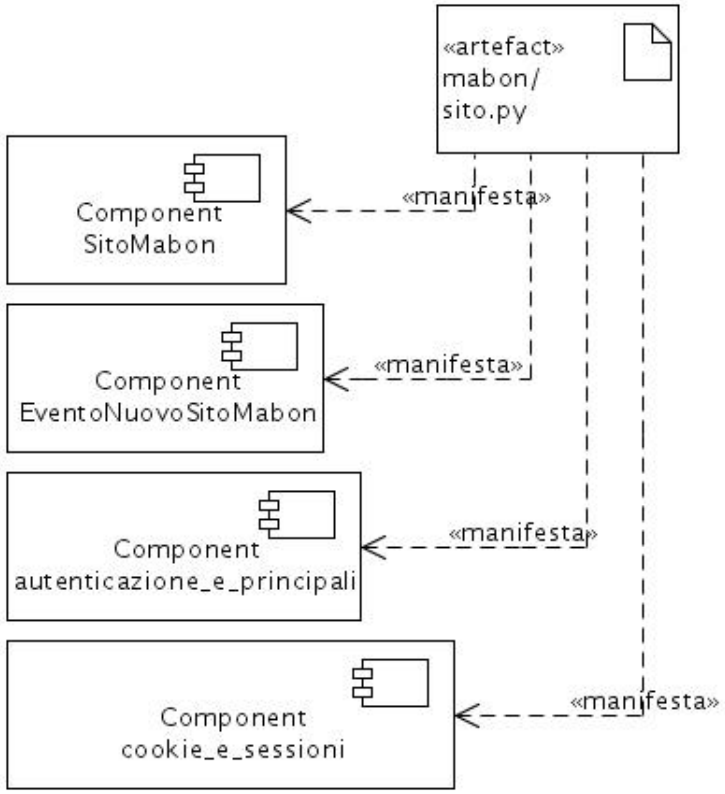
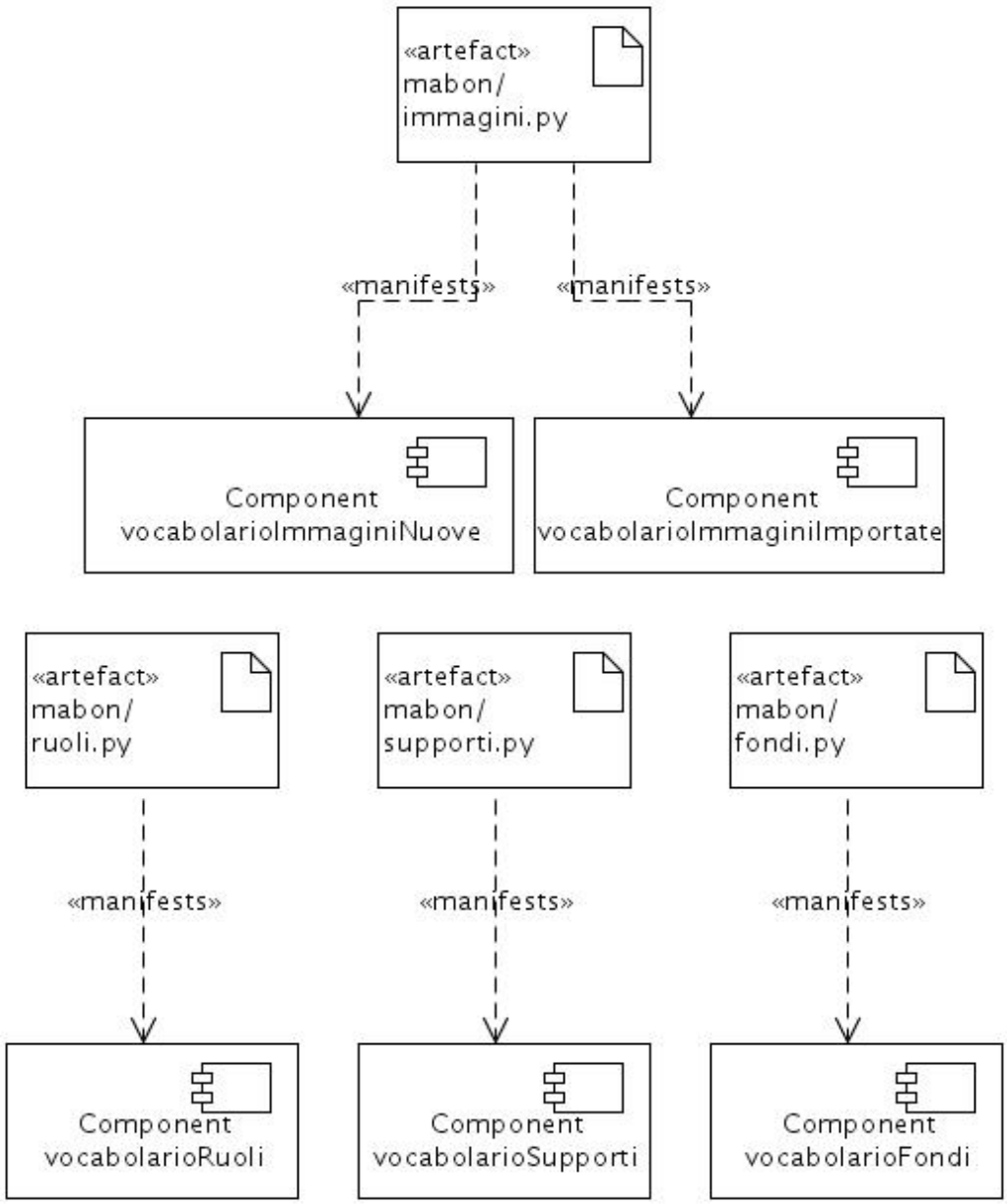


Figura 7.9: Diagramma di struttura per i componenti `vocabolarioImmaginiNuove`, `vocabolarioImmaginiImportate`, `vocabolarioSupporti`, `vocabolarioFondi` e `vocabolarioRuoli`.



Capitolo 8

Relazioni e vincoli fra le classi e le istanze

8.1 Generale

Prima di un'analisi dettagliata delle interazioni che avvengono a runtime, è necessario illustrare le relazioni che intercorrono fra le classi, gli oggetti e le istanze, mostrate nella successiva immagine.

8.2 Le relazioni imperniate su `Utente`

Nella figura 8.1 sono evidenziate in verde le classi che sono riconducibili alla Vista, ovvero i multi–adattatori che implementano direttamente o meno l'interfaccia `IBrowserPublisherRequest` e gli oggetti da mostrare. Tali classi si trovano in files presenti nella pacchetto browser. Sono quindi il tramite tra il Modello, o comunque le classi che lo adattano, e l'interazione con l'utente che avviene tramite il Controllore rappresentato qui in blu.

Per motivi di spazio non sono stai mostrati alcuni componenti del Controllore responsabili della generazione degli eventi per effetto delle azioni svolte

dalle classi `ModuloModificaUtente` e `ModuloAggiungiUtente`, dal componente `generaUtente`; è comunque presente un riquadro in grigio scuro che rappresenta gli eventi che attivano i gestori evidenziati in viola. L'adattatore `IntIds` presente nelle librerie di Zope, ha il compito di creare, aggiornare e rimuovere un indentificatore intero per le istanze dell'oggetto Utente presenti nello ZODB.

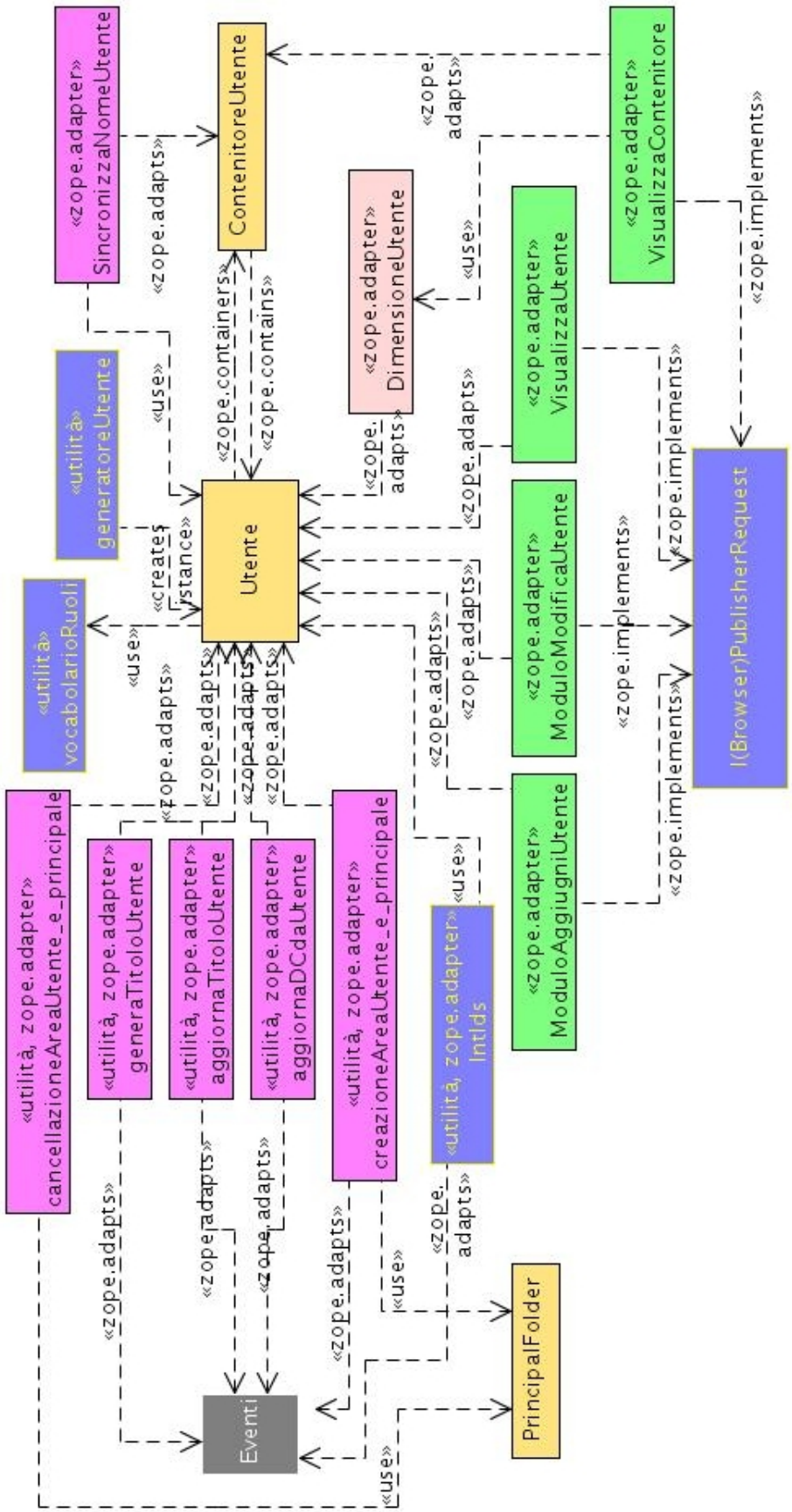
Il componenti del Modello sono rappresentati in arancione, ovvero il componente Utente e il suo contenitore. A metà tra il Modello e il Controllore vi è anche l'adattatore rosa `DimensioneUtente` la cui funzione è di adattare alle richieste di `VisualizzaContenitore`, vista di un contentitore fornito da Zope, l'oggetto Utente.

8.3 Le relazioni imperniate su `Scheda`

Nella figura 8.2 sono evidenziate in verde le classi che sono riconducibili alla Vista, ovvero i multi–adattatori che implementano direttamente o meno l'interfaccia `IBrowserPublisherRequest` e gli oggetti da mostrare. Tali classi si trovano in files presenti nella pacchetto browser. Sono quindi il tramite tra il Modello, o comunque le classi che lo adattano, e l'interazione con l'utente che avviene tramite il Controllore rappresentato qui in blu.

Per motivi di spazio non sono stai mostrati alcuni componenti del Controllore responsabili della generazione degli eventi per effetto delle azioni svolte dalle classi `ModuloModificaScheda` e `ModuloAggiungiScheda`, dal componente `generaScheda`; è comunque presente un riquadro in grigio scuro che rappresenta gli eventi che attivano i gestori evidenziati in viola. L'adattatore `IntIds` presente nelle librerie di Zope, ha il compito di creare, aggiornare e rimuovere un indentificatore intero per le istanze dell'oggetto Scheda presenti nello ZODB. L'adattatore `TestoSchedaPerRicerca`, necessario al catalogo per l'estrazione dei dati relativi

Figura 8.1: Diagramma delle relazioni relativo al componente Utente

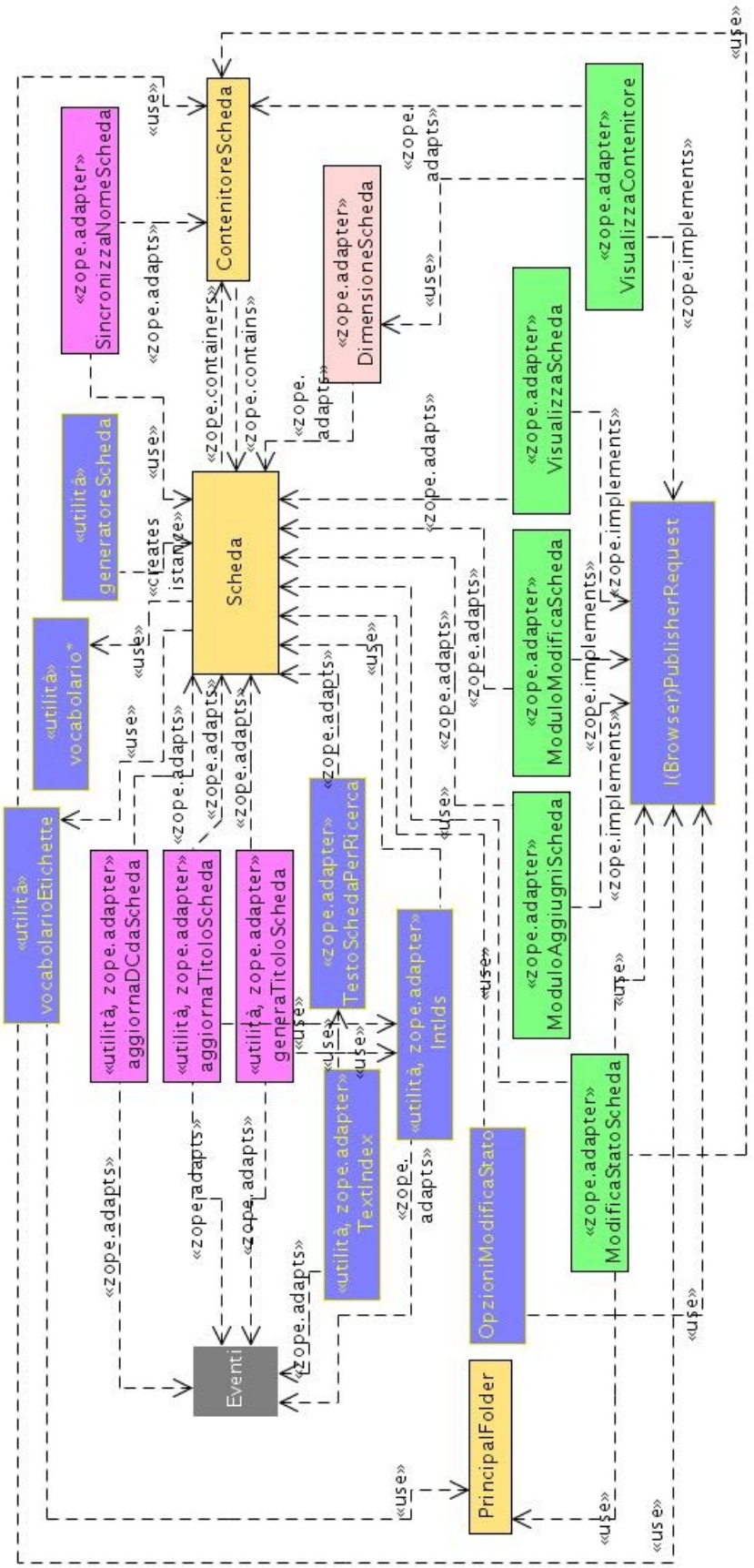


alla compilazione dell'indice, viene utilizzato da un gestore di eventi automaticamente creato da Zope come effetto dell'istanziazione della classe `TextIndex` nel gestore di eventi `catalogo_e_indice`. L'adattatore `ModificaStatoScheda` accede agli attributi stato e `__parent__` dell'istanza dell'oggetto Scheda da modificare e al campo principal dell'istanza dell'oggetto `Request` e, se necessario, anche all'istanza del `ContenitoreScheda` che conterrà e a quella che contiene l'istanza della scheda da spostare.

La classe `OpzioniModificaStato` è il visualizzatore che accede all'istanza della classe `Request` e ad una istanza dell'oggetto scheda visualizzata per ottenere il Principale autenticato e il valore del campo `stato` dell'istanza della `Scheda`.

Il componenti del Modello sono rappresentati in arancione, ovvero il componente Scheda e il suo contenitore. A metà tra il Modello e il Controllore vi è anche l'adattatore rosa `DimensioneScheda` la cui funzione è di adattare alle richieste di `VisualizzaContenitore`, vista di un contentitore fornito da Zope, l'oggetto Scheda.

Figura 8.2: Diagramma delle relazioni relativo al componente Scheda



Capitolo 9

Interazioni fra i componenti

9.1 Generale

Per una più agevole comprensione delle complesse interazioni fra i componenti, quest'ultime sono state suddivise in varie categorie in base ai componenti coinvolti (esclusivamente di Mabon o alcuni di Mabon e altri forniti dal framework di Zope) e alla caratteristica di essere sincrone o meno.

9.2 Le interazioni asincrone

Le interazioni asincrone sono quelle che coinvolgono componenti che vengono richiamati o utilizzati indirettamente o per effetto di altri componenti oppure quelle che necessitano di più di una interazione con l'utente per completarsi.

Nel primo caso si tratta di gestori di eventi la cui attivazione avviene a causa di alcuni eventi generati da altri componenti e di viste di anteprima o comunque di adattatori predisposti alla visualizzazione di un'istanza, che vengono attivati in base alle scelte dell'utente; nel secondo caso le viste di aggiunta e modifica di un'istanza.

9.2.1 I gestori di eventi: SitoMabon

Due degli eventi generati dall'istanziazione della classe SitoMabon vengono

intercettati dai componenti di Mabon.

L'aggiunta del gestore locale del sito a quello globale dell'istanza di Mabon, è seguito dalla generazione dell'evento `EventoNuovoSitoMabon`. Due diversi gestori `catalogo_e_indice` e `cookie_e_sessioni` vengono attivati.

- Il primo collega al gestore del sito Mabon l'utilità di generazione dell'IntegerID (che genera un identificatore numerico univoco per ciascuna istanza di un'oggetto che verrà aggiunta alle istanze dei contenitori presenti in Mabon necessario alle utilità di ricerca e indicizzazione), l'utilità di ricerca e infine il plugin `TextIndex` dell'utilità di Ricerca per la generazione di un'indice delle istanze degli oggetti che verranno inseriti in Mabon. Il plugin di Indicizzazione necessita della specifica di un'interfaccia (`ISearchableText` in Mabon) e un metodo o un attributo (nel caso di Mabon il metodo `estraiTesto`), presente nel componente implementante l'interfaccia, che permette l'estrazione del testo da indicizzare dall'istanza di un oggetto. La compilazione dell'indice viene quindi delegata in maniera asincrona ai componenti elencati nel registro dei componenti che offrono l'interfaccia `ISearchableText`. Quest'ultimo a sua volta istanzierà automaticamente tre gestori di eventi (uno per la creazione, uno per la modifica e uno per la cancellazione delle istanze degli oggetti) adibiti alla ricerca di adattatori che implementano l'interfaccia `ISearchableText` e adattano almeno una tra quelle offerte dall'istanza che ha generato l'evento. Se tale adattatore esiste (la classe `TestoSchedaPerRicerca`), viene infine inserita nell'indice tupla con l'`IntegerID` dell'istanza e il testo estratto da essa.

- Il secondo configura i contenitori (persistenti, residenti cioè nel database e non in RAM) per i dati di Sessione e i parametri per i cookies di sessioni, entrambi necessari alla plugin per le Credenziali `SessionCredential`.

L'aggiunta dell'istanza della classe SitoMabon al contenitore principale di Zope genera un evento `IObjectAddedEvent`. Quest'ultimo viene intercettato dal gestore `autenticazione_e_principali` che collega:

- l'istanza `SitoMabon` al registro gestore del sito (a quello locale, non quello globale dell'istanza di Zope);
- all'istanza del contenitore `SitoMabon` le istanze dei contenitori per le schede con stato di documento, per gli utenti e per le aree utenti;
- al gestore del sito un'istanza dell'utilità di Autenticazione;
- a quest'ultima un'istanza del contenitore `PrincipalFolder`, plugin di Autenticazione, dove risiederanno le istanze degli oggetti `InternalPrincipal` contenenti ciascuno una tripletta (`livello di privilegio`[1], `login`, `password`) associata a ciascuna istanza della classe `Utente` creata;
- un `InternalPrincipal` definito da (`er_capo`, `123`, `Amministratore`) per consentire l'accesso all'applicazione subito dopo l'installazione.

  Infine vengono attivate in seno all'utilità di Autenticazione le plugins per le Credenziali (`NoChallengeIfAuthenticated` e `SessionCredential`) e la plugin di Autenticazione.

9.2.2 I gestori di eventi: Schede

La creazione, modifica e cancellazione di un'istanza dell'oggetto scheda provoca la generazione dei relativi eventi che vengono intercettati sia da componenti di Mabon sia da componenti di Zope che vengono utilizzati da Mabon.

[1] il Ruolo.

L'evento generato dalla creazione di un'istanza scheda per effetto dell'azione del componente generatoreScheda viene intercettato dall'utilità di assegnazione degli IntIds e dall'estrattore di informazioni per la costruzione dell'indice per quel che concerne gestori forniti da Zope; mentre il componente `generaTitoloScheda` genera il valore per l'attributo `IScheda.titolo`.

L'evento generato dalla modifica di un'istanza scheda per effetto delle viste `ModuloModificaScheda` e `ModuloAggiungiScheda`, che assegna i valori immessi dall'utente ad un'istanza appena creata da `generatoreScheda`, viene intercettato dall'utilità di assegnazione degli `IntIds` e dall'estrattore di informazioni per la costruzione dell'indice per quel che concerne gestori forniti da Zope e dai componenti `aggiornaTitoloScheda` e `aggiornaDCdaScheda` di Mabon.

L'evento generato dalla cancellazione di un'istanza scheda viene intercettato dall'utilità di assegnazione degli IntIds e dall'estrattore di informazioni per la costruzione dell'indice per quel che concerne gestori forniti da Zope.

La creazione, modifica e cancellazione di schede non riguarda solo le istanze stesse ma anche le istanze dei `ContenitoreSchede`; infatti aggiunta, rimozione e spostamento degli oggetti contenuti genera degli eventi, ma la gestione di tali eventi in relazione agli oggetti scheda non è utile a Mabon.

9.2.3 I gestori di eventi: Utenti

La creazione, modifica e cancellazione di un'istanza dell'oggetto utente provoca la generazione dei relativi eventi che vengono intercettati sia da componenti di Mabon sia da componenti di Zope che vengono utilizzati da Mabon.

L'evento generato dalla creazione di un'istanza scheda per effetto dell'azione del componente `generatoreUtente` viene intercettato dall'utilità di assegnazione degli IntIds per quel che concerne gestori forniti da Zope; mentre il componente `generaTitoloUtente` di Mabon genera il valore per l'attributo `IUtente.titolo`.

L'evento generato dalla modifica di un'istanza scheda per effetto delle viste `ModuloModificaUtente` e `ModuloAggiungiUtente`, che assegna i valori immessi

dall'utente ad un'istanza appena creata da `generatoreUtenti`, viene intercettato dall'utilità di assegnazione degli `IntIds` e dall'estrattore di informazioni per la costruzione dell'indice per quel che concerne gestori forniti da Zope e dai componenti `aggiornaTitoloUtente` e `aggiornaDCdaUtente` di Mabon.

L'evento generato dalla cancellazione di un'istanza scheda viene intercettato dall'utilità di assegnazione degli `IntIds` e dall'estrattore di informazioni per la costruzione dell'indice per quel che concerne gestori forniti da Zope.

La creazione, modifica e cancellazione di schede non riguarda solo le istanze stesse ma anche le istanze dei `ContenitoreUtenti`; infatti aggiunta, rimozione e spostamento degli oggetti contenuti genera degli eventi che vengono intercettati dai due gestori di creazione e rimozione dell'area utente. Quello attivato dall'aggiunta dell'istanza di `Utente` all'istanza di `ContenitoreUtenti` interagisce con la plugin di autenticazione, dove viene creato un nuovo `Principal`, con una parte dei dati dell'utente appena creato, con l'utilità locale offerta da Zope per la gestione dei Ruoli, per assegnare il Ruolo al nuovo `Principal`, e con il contenitore per le aree utente, dove creano un'istanza di `ContenitoreEtichette` per le etichette dell'utente appena creato e un'istanza di `ContenitoreSchede` per le bozze. L'altro, attivato dalla rimozione dell'istanza di `Utente` dall'istanza di `ContenitoreUtenti` rimuove le due istanze di `ContenitoreEtichette` e di `ContenitoreSchede` e il Principale dalla plugin di Autenticazione.

9.2.4 La generazione dell'identificatore intero per un istanza contenuta

Ogni qual volta un'istanza di un oggetto viene creata o cancellata, viene creato o rimosso un riferimento `IntegerId` tramite un gestore di eventi istanziato dall'utilità di `IntIds`.

9.2.5 La generazione dell'indice per gli oggetti scheda

Ogni qual volta un'istanza di un oggetto scheda viene creata o modifica-

ta, vengono aggiornati gli Indici del Catalogo: l'indice per la ricerca semplice tramite l'adattatore `TestoPerRicerca`, gli altri (per la ricerca avanzata) tramite un adattatore istanziato da Zope che implementa le specifiche di estrazione (`ISearchableText`) e adatta l'istanza. Tali adattatori sono conseguenza degli effetti del codice implementato nel gestore di eventi `catalogo_e_indici`.

9.2.6 I componenti di aggiunta e modifica

I moduli `ModuloAggiungiScheda`, `ModuloAggiungiUtente`, `ModuloModificaScheda` e `ModuloModificaUtente` necessitano di più interazioni con l'utente per completare le proprie operazioni.

9.2.6.1 Il componente di aggiunta

L'azione di aggiunta di un'istanza di una classe `Utente` o `Scheda`, dal punto di vista della renderizzazione del pagine XHTML restituita all'utente, è caratterizzata da un form con attributo action contenente il valore "`@@+/action.html`" e un'opzione contenente il valore "`Mabon.Utente`" o "`Mabon.Scheda`".

Il diagramma di interazione per la creazione di un oggetto Utente è raffigurato in Figura 9.1.

La sequenza inizia quando l'utente seleziona l'opzione "aggiungi utente" (o "aggiungi scheda") e preme il pulsante submit del form. Al server arriva una richiesta POST contenente i dati `@@+/action.html` e `Mabon.Utente` o `Mabon.Scheda` che vengono estratti e inoltrati al `Publisher`.

Quest'ultimo nella *fase di traversing* li interpreta nella vista (`@@`[2]) (che compie un'operazione) di aggiunta o *adding view* (`+`[3]) con nome "`action.html`" alla quale passa come dato "`Mabon.Utente`" o "`Mabon.Scheda`". La vista `action.html` cerca nel registro dei componenti un adattatore che implementi l'interfaccia `IAdding`

[2]Il Publisher identifica una vista preponendo i caratteri @@ ad un identificatore. Se immaginiamo che @@ sia la stilizzazione di due occhi che guardano, il simbolo molto espressivo per questa funzione.

[3]Il Publisher identifica una vista di aggiunta tramite il +, simbolo molto espressivo per questa funzione.

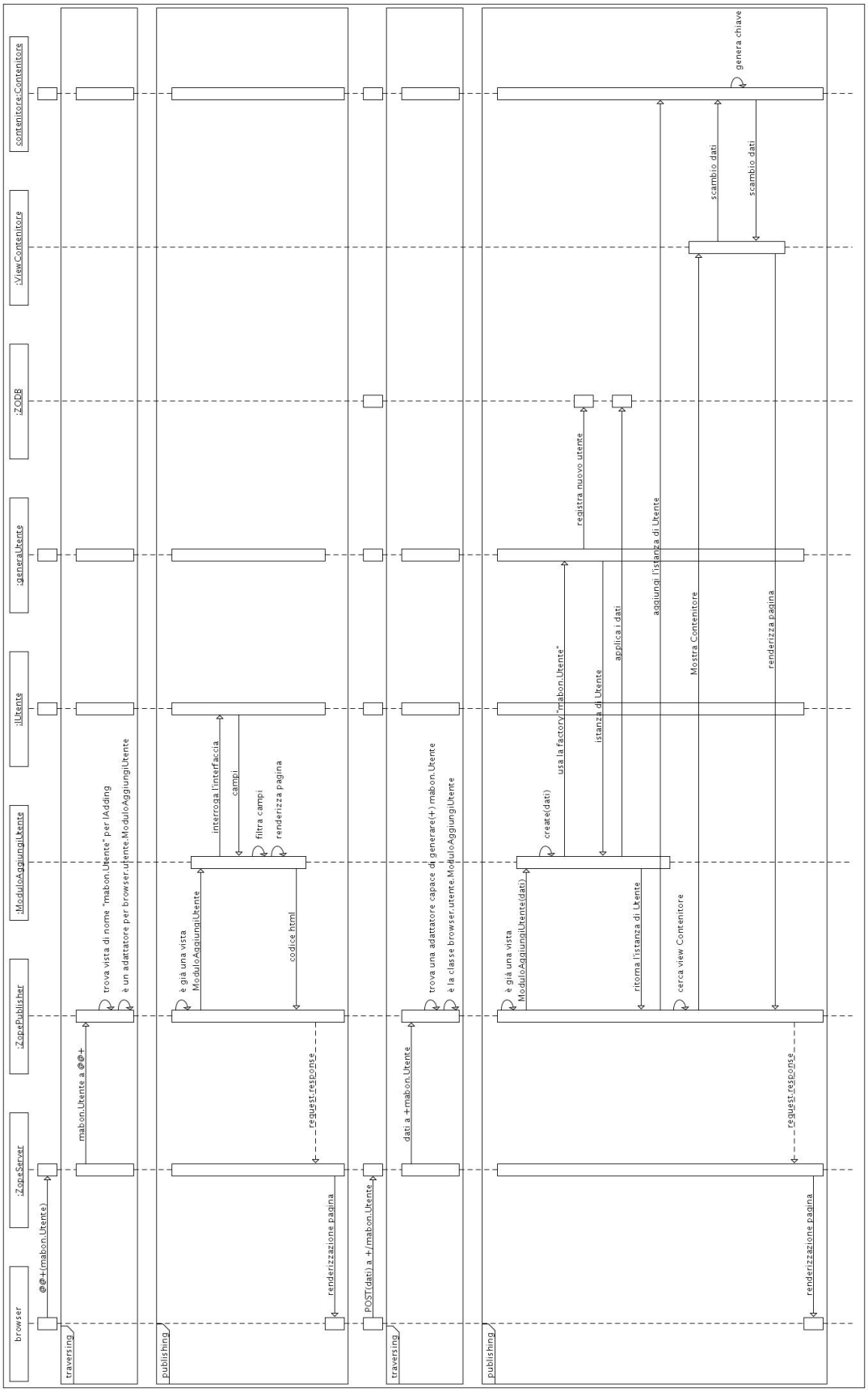
e che abbia nome a seconda dei casi "`Mabon.Utente`" o "`Mabon.Scheda`": tali adattotori sono rispettivamente implementati dai componenti `ModuloAggiungiUtente` e `ModuloAggiungiScheda`.

Viene quindi evocata dal `Publisher` nella *fase di publishing* la vista implementata dalla classe `ModuloAggiungiScheda` o `ModuloAggiungiUtente` che estrae la lista degli attributi dall'interfaccia `IScheda` o `IUtente` renderizza il codice XHTML per l'utente che viene restituito dalla vista al Publisher, e poi al server e infine visualizzato sul browser dell'utente.

Dopo la compilazione da parte dell'utente del form e segue l'invio dei dati al server; nel form della pagina rederizzata il campo action contiene "`+/Mabon.Utente`" o "`+/Mabon.Scheda`". Il Publisher ricerca un generatore (`+`) col nome suddetto come oggetto a cui inviare i dati digitati dall'utente; tali generatore sono i componenti `ModuloAggiungiUtente` / `ModuloAggiungiScheda`. Il metodo `create()` di tali componenti crea[4] infine, un'istanza della classe `Scheda` / `Utente` e vi assegna i valori immessi dall'utente. L'evento `IObjectCreatedEvent`/`IUtente` o `IScheda` attiva il gestore `generaTitolo`.

L'istanza viene restituita al `Publisher` che provvede ad inserirla nell'istanza del `ContenitoreUtente` / `ContenitoreScheda`. L'adattatore `SincronizzaNomeUtente` / `SincronizzaNomeScheda` provvede ad indicizzare l'istanza appena creata all'interno del contenitore col valore restituito dal componente `Sincronizza*`. Viene infine generato un evento del tipo `IObjectAddedEvent`/`IUtente` o `IScheda`, che nel caso di un'istanza di `Utente` attiva il gestore `creazioneAreaUtente_e_principale` che, come abbiamo visto nella descrizione del componente interagisce con la plugin di Autenticazione e aggiunge un'istanza di `ContenitoreSchede` e una di `ContenitoreEtichette` nel contenitore "`Aree Utente`".

Figura 9.1: Diagramma di interazione per la creazione di un oggetto Utente.



9.2.6.2 Il componente di modifica

La modifica di un'istanza `Scheda` o `Utente` avviene tramite un form con campo action contenente la vista `@@edit.html` dell'oggetto e più precisamente la richiesta sarà del tipo "`.../XYZ/@@edit.html`", con al posto di XYZ la stringa restituita dall'adattatore `SincronizzaNome*`.

Il Publisher evoca il componente che fornisce la vista per l'istanza, implementato dalla classe `ModuloModificaUtente` o `ModuloModificaScheda`. Quest'ultimo estrae la lista dei campi per la corrispondente interfaccia `IUtente` o `IScheda`, e restituisce tramite la catena Publisher–server all'utente la pagina contenente il form per modificare l'istanza. Il campo action di questo form conterrà lo stesso valore di quello utilizzato per evocare la vista nel quale è anche contenuta l'indicazione dell'istanza da modificare; il nome e l'id del pulsante Submit (`form.actions.apply`) indicano che i dati digitati dall'utente che verranno trasmessi in POST dovranno essere processati dal metodo apply della classe `EditForm` da cui derivano i due componenti `ModuloModifica*`. Il metodo apply copierà tali dati nei rispettivi attributi dell'istanza.

La modifica di un'istanza genera un segnare `IObjectModifiedEvent`/`IScheda` o `IUtente`, il quale attiva il gestore `aggiornaTitoloScheda`, `aggiornaDCdaScheda` e `aggiornaDCdaUtente`.

9.2.7 La ricerca delle schede

9.2.7.1 La ricerca semplice

Il diagramma delle interazioni per l'operazione di ricerca semplice è raffigurato in Figura 9.2.

L'operazione di ricerca semplice è offerta dal visualizzatore "ricerca" il cui contenuto renderizzato in una pagina XHTML è una porzione di codice contenente

[4] tramite il componente `generatoreScheda` / `generatoreUtente`.

il form di ricerca (con nome e id "`query`") e l'attributo action del form con valore "`@@ricerca.html`".

Il Publisher, ricevuti i dati dal server, trova la vista col nome "`ricerca.html`" la cui implementazione è realizzata tramite la classe `PaginaDiRicerca`, la quale cerca il componente di Catalogo nel registro dei componenti e lo utilizza passando all'indice la stringa di ricerca "`query`". Tramite il modello associato e la sinergia col codice TAL presente nello stesso, viene renderizzata la pagina XHTML contenente i risultati della ricerca.

9.2.7.2 La ricerca avanzata

L'operazione di ricerca avanzata è offerta dalla vista `parametriricercaavanzata.html`, implementata dalla classe `ModuloRicercaAvanzata`. La vista restituisce all'utente una pagina contenente un form di ricerca che punta (tramite il l'attributo action) alla vista `ricercaavanzata.html`, implementata dalla classe `PaginaDiRicercaAvanzata`, e che contiene i widget di ricerca, ciascuno con un identificativo proprio.

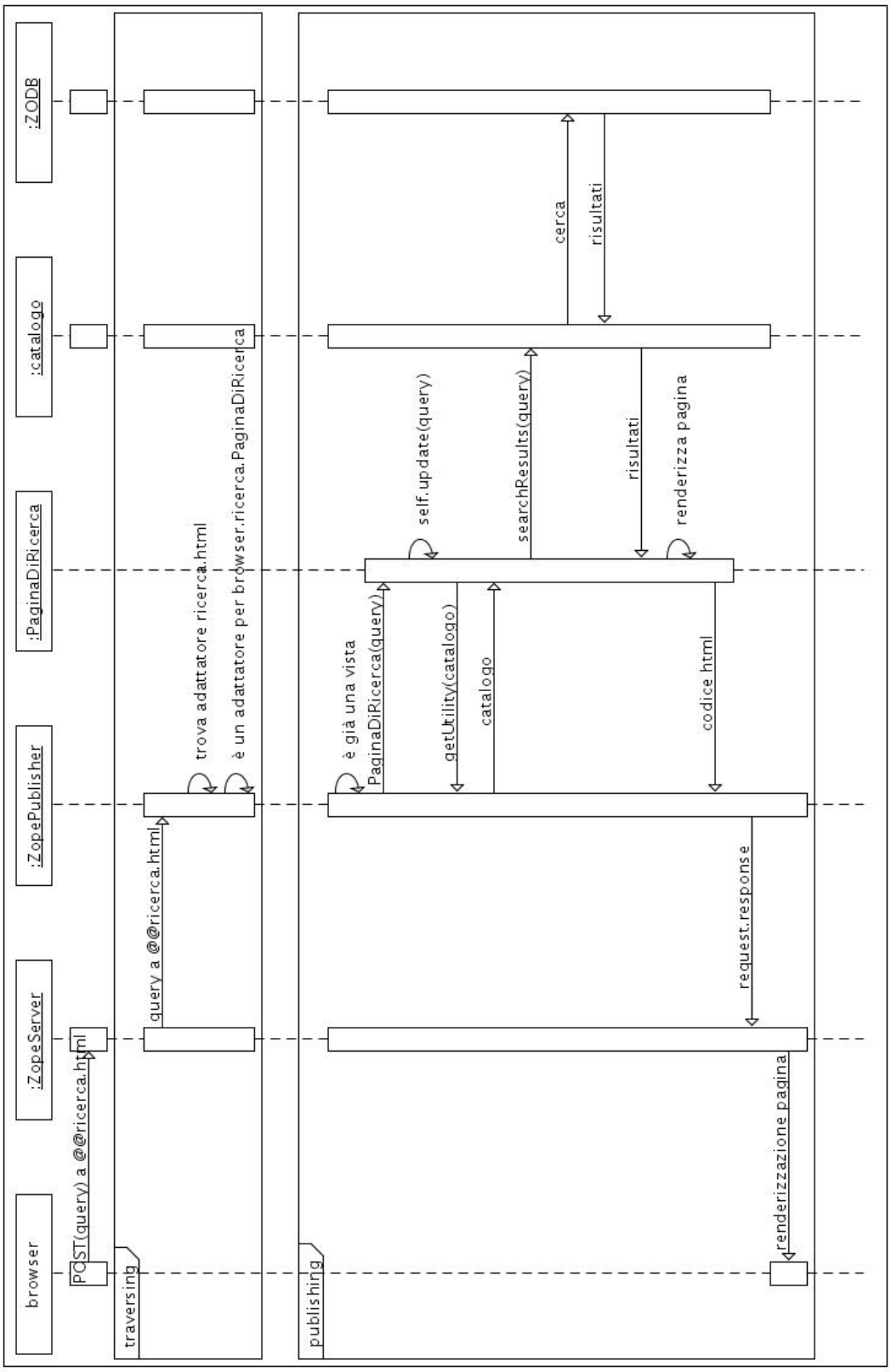
Il Publisher, ricevuti i dati dal server immessi tramite la vista `parametriricercaavanzata.html`, trova la vista col nome `ricercaavanzata.html` e ne invoca il metodo `__call__()`. Ciascun parametro, se non nullo, viene utilizzato per cercare nell'Indice (del componente di Catalogo) avente indice[5] omonimo del nome del campo stesso nel form XHTML.

Viene così creato un dizionario con nome `query` contenente come chiavi (ovvero indici) gli identificatori dei campi del form (che corrispondono per omonimia agli Indici del Catalogo) e come valore associato a ciascuna chiave il testo digitato dall'utente.

Il dizionario viene passato come parametro alla funzione di ricerca del Ca-

[5]Il componente Catalogo è anche un oggetto contenitore. Ciascun Indice del Catalogo viene dunque indicizzato a sua volta all'interno del contenitore. Gli l'aggiunta degli Indici al catalogo viene effettuata dal gestore di eventi `catalogo_e_indici`.

Figura 9.2: Diagramma di interazione per una ricerca semplice



talogo. Infine la pagina XHTML, restituita dalla vista `ricercaavanzata.html`, contiene i risultati della ricerca.

9.2.8 L'importazione delle immagini

La sequenza dell'importazione delle immagini inizia dalla vista `azioni.html`.

Quando lo Scansionatore seleziona nella vista l'opzione di importazione, in realtà preme il pulsante submit di un form che contiene anche il menù popolato con gli elementi presenti nel vocabolario "Supporti". Ciascuna voce del menu è collegata all'omonima cartella in `immagini/nuove/`. La voce del menu scelta e inviata alla vista `listaimmagini.html` indica la cartella nella quale cercare le immagini da importare.

La vista `listaimmagini.html` produce una pagina XHTML contenente un form con la lista delle immagini da importare presenti in `immagini/nuove/voci_del_menù_scelte` ed eventualmente quelle già importate e che hanno lo stesso nomefile. Ciascun file è selezionabile tramite una checkbox. I files selezionati dall'utente e la il `SupportoFisico` scelti dallo Schedatore vengono poi inoltrati dal form alla vista `importaimmagini.html` che infine effettua l'importazione.

9.3 Le interazioni sincrone

9.3.1 Generale

Quasi tutte le interazioni sincrone presenti in Mabon avvengono tra componenti forniti dal sistema e componenti implementati per l'applicazione.

9.3.2 L'autenticazione dell'utente

La sequenza che porta all'autenticazione dell'utente può avere inizio in due modi:

- l'utente (e quindi l'oggetto che rappresenta la sua interazione derivato da `IPublisherRequest`) interagisce con un componente che richiede dei permessi che l'utente non possiede e quindi viene sollevata un'eccezione `Unauthorized` che viene intercettata dal `PAU`;
- l'utente naviga esplicitamente alla pagina di login.

A ciascuna plugin per le Credenziali, tra quelle registrate per la `PAU`, viene offerta la possibilità di richiedere all'utente di autenticarsi tramite la visualizzazione di un form di login; se nessuna lo fa viene mostrato un messaggio di errore di tipo "Utente non autorizzato".

Altrimenti l'utente, che viene rappresentato tramite un oggetto `Principal`, interagisce con la vista di login (è una vista perché adatta il componente di autenticazione `PAU` alla interfaccia `IPublisherRequest` per mezzo di una plugin per le Credenziali) e immette i propri dati identificativi (login e password) che vengono estratti dalla plugin e inviati alle varie plugin di Autenticazioni registrate per la `PAU`.

Queste plugin vengono attivate in sequenza finché una di esse non ritorni un oggetto che offre un'interfaccia `IPrincipalInfo`; in caso contrario viene mostrato un messaggio di errore di tipo "Utente non autorizzato".

L'oggetto `PrincipalInfo` restituito assume il Ruolo associatogli dal gestore di `eventi creazioneAreaUtente_e_Principale` che viene ereditato dagli oggetti `Request` associati al `Principal`. La plugin `SessionCredentials` aggiunge l'oggetto Principal corretto a tutte le istanze di Request in ciascuna interazione tra l'utente corrispondete e l'applicazione.

9.3.3 La modifica dello stato della scheda

La modifica dello stato della scheda coinvolge l'apposito il visualizzatore e quindi la classe `OpzioniModificaStato` che lo implementa. In funzione dello stato della scheda visualizzata, la cui interfaccia IScheda è adattata dalla classe `OpzioniModificaStato`, e del ruolo del `Principal` attualmente autenticato, viene generato o meno un form contenente un campo nascosto con l'IntId della scheda, eventualmente l'indice identificativo dell'utente nell'`Area Utente` e il nuovo stato per l'oggetto Scheda.

Questi valori vengono trasmessi alla vista `statoscheda.html` implementata dalla classe `ModificaStatoScheda` che effettua la modifica dello stato e l'eventuale spostamento in un nuovo contenitore (nel caso di cambio dello stato da "proposta" a "bozza" o da "documento" a "bozza".

9.3.4 Il calcolo della dimensione di un oggetto

Le viste di un'istanza di un oggetto contenitore mostrano vari dettagli degli oggetti contenuti, tra i quali vi è la dimensione che viene restituita da un'adattatore che implementi `ISized` e adatti l'interfaccia dell'oggetto contenuto. Poiché è previsto che questi adattatori restituiscano una stringa, in Mabon sono stati utilizzati per fornire un'importante informazioni sui Principali e gli Utenti (tramite rispettivamente gli adattatori DimensionePrincipale e DimensioneUtente): il Ruolo associato. Similmente l'adattatore DimensioneScheda restiuisce per una scheda in stato di "bozza" il numero di etichette associate.

9.3.5 La visualizzazione di un oggetto

Le viste degli oggetti Scheda e Utente vengono realizzate per mezzo degli adattatori implementati dalle classe `VisualizzaUtente` e `VisualizzaScheda`. Come tutti gli adattatori ricevono l'istanza da adattare come parametro del costruttore e come viste, tramite l'utilizzo dei dati restituiti da un oggetto `View-`

`PageTemplateFile`, restituiscono la pagina XHTML per l'utente contenente una rappresentanzione intelligibile per l'utente del contenuto dell'istanza.

Parte III

Installazione ed uso del programma

Capitolo 10

Installazione

10.1 Requisiti

Per l'installazione dell'applicazione Mabon è necessario:

- un sistema operativo che consenta l'esportazione e l'accesso in scrittura di porzioni di disco in rete tramite password (opzionale, ma garantisce una migliore sicurezza);
- un'istanza di Zope 3.

Per l'utilizzo è necessario:

- almeno uno scanner per il Ruolo di scansionatore;
- o alternativamente le immagini che si assoceranno alle schede già scansionate in formato .jpg.

10.2 Installazione

Creare una cartella `mabon` (minuscolo) all'interno della cartella Products dell'istanza di Zope 3 e scompattarvi i files contenuti nell'archivio di installazione in formato `.tar.bz2`.

Nella cartella `etc/package-includes/` vanno copiati i due files `mabon-configure.zcml` e `mabon-overrides.zcml` presenti nella cartella `mabon/etc/` e va aggiunto al file `etc/principals.zcml` un `grant`[1] del `Role` "`mabon.Visitatore`" al `Principal` "`zope.anybody`" per consentire ad un utente non utenticato l'accesso alla vista di login.

La cartella `immagini/nuove` e tutte le sottocartelle presenti dovrnno essere condivisa in maniera tale che sia gli utenti remoti sia l'applcazione possano copiarvi e cancellarvi dei files senza che però le cartelle siano cancellabili.

10.3 Aggiunta di una nuova traduzione dell'interfaccia

L'aggiunta di una nuova traduzione necessita di un file in formato `.mo` (ovvero `.pot` / `.po` compilato).

La compilazione può essere effettuata tramite il programma `msgfmt` facente parte delle utilità GNU *gettext* oppure il file può già essere salvato in formato `.mo` dal programma utilizzato per effettuare la traduzione (vedi pagina 36).

Il file `.mo` generato deve chiamarsi `mabon.mo` e va inserito nella cartella `mabon/locales/`*`linguaggio`*`/LC_MESSAGES/`, dove *`linguaggio`* deve essere nella forma `de`, `en`, `it`, ecc.

[1] Tramite il tag ZCML `grant` si può attribuire un Ruolo a un Principal.

Capitolo 11

Uso

11.1 Generale

L'utente che vuol utilizzare l'applicazione deve necessariamente disporre di un account.

Il primo passo è l'autenticazione tramite il form di login, al quale segue nel caso di successo, la visualizzazione di una pagina con le varie azioni disponibili in funzione del Ruolo associato all'account.

Non è consentito creare due accounts con stesso login.

Terminata la sessione di lavoro è necessario effettuare il logout.

11.2 Opzioni per l'Amministratore

11.2.1 Generale

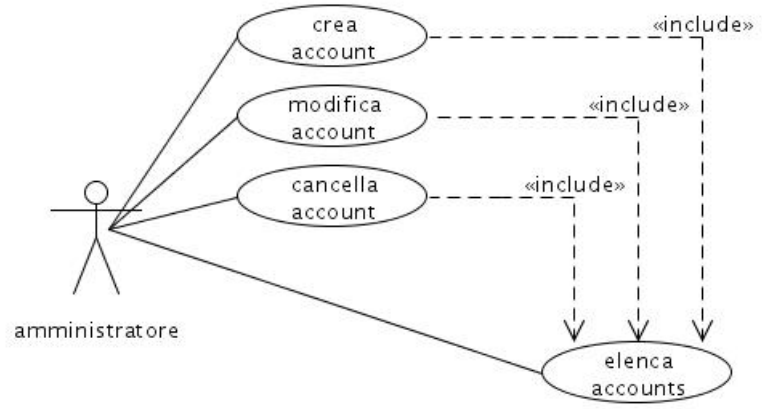
L'Amministratore è la figura chiave per l'utilizzo dell'applicazione, poiché è l'unico Ruolo in grado di creare gli accounts utente.

Il diagramma dei casi d'uso si trova in Figura 11.1.

11.2.2 Opzioni e casi d'uso

L'Amministratore dispone delle seguenti opzioni dopo il login:

Figura 11.1: Diagramma dei casi d'uso per l'Amministratore.



- visualizzazione degli accounts creati nell'applicazione;
- creazione nuovo account;
- cancellazione account.

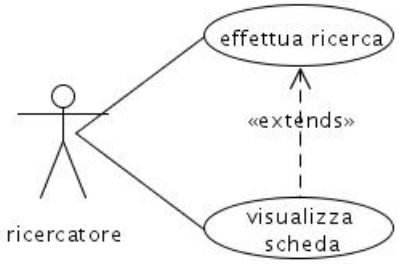
Queste opzioni vengono realizzate tramite la vista fornita da Zope per l'oggetto `ContenitoreUtenti`, che consente di selezionare le voci da cancellare e poi premere il pulsante "Cancella" o quella da rinominare e poi premere il pulsante "Rinomina". La creazione di un'account avviene tramite il menu a tendina in alto a sinistra.

Durante la creazione dell'account, l'Amministratore deve inserire `nome`, `cognome`, `password`, `email` e `ruolo` dell'utente. I primi quattro campi sono liberi mentre l'ultimo è gestito da un vocabolario chiuso (il cui contenuto è quello del vocabolario "Ruoli").

11.3 Effettuare una ricerca nel database

Mabono offre due tipi di ricerca nel database: una semplice e l'altra avanzata.

Figura 11.2: Diagramma dei casi d'uso per il Ricercatore



La ricerca semplice, effettuabile tramite un apposita area nella parte sinistra dello schermo, consiste nella ricerca di una porzione di testo in tutte le schede presenti nel database, anche quelle che non hanno stato di *documento*. L'utente deve compilare l'apposito campo e premere il pulsante "Cerca".

La ricerca avanzata consente all'utente di specificare una stringa di ricerca per ciascun campo presente in una scheda. I campi compilativi – quelli che non contengono valori provenienti da un vocabolario predefinito – vengono inclusi nella ricerca quando nel riquadro relativo al campo viene immesso del testo; i campi chiusi – quelli il cui valore viene estratto da un vocabolario – dispongono di un'ulterire opzione "Non usare" per non includerli nella ricerca.

Dopo aver premuto il tasto "Cerca", viene mostrata una pagina contenente le schede trovate.

11.4 Opzioni per il Ricercatore

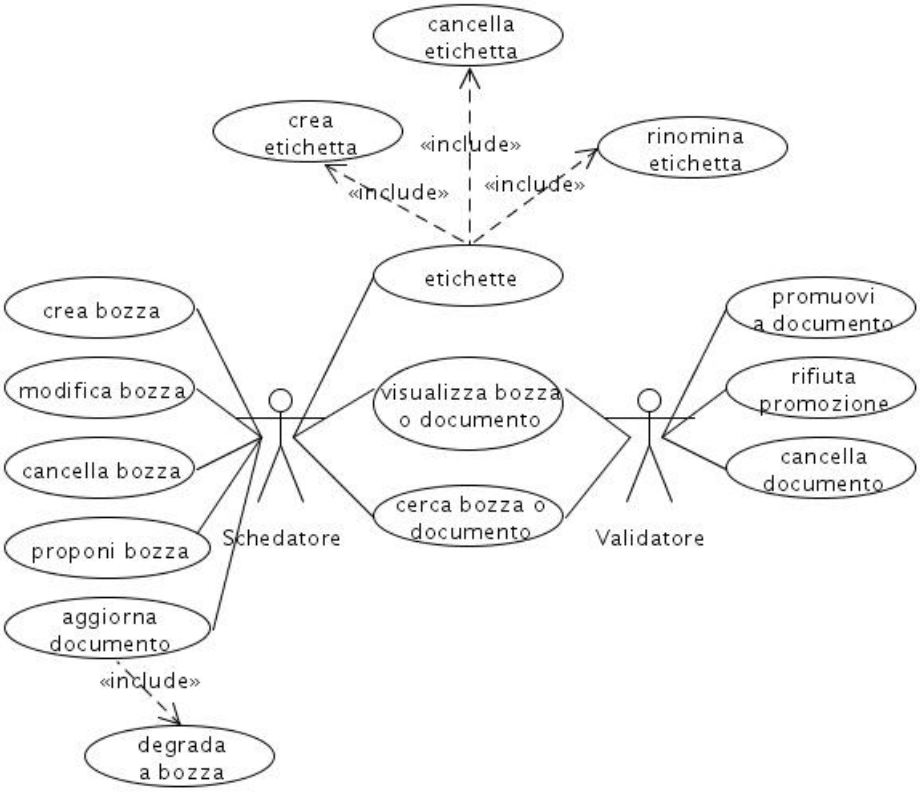
11.4.1 Generale

Un Ricercatore, come suggerisce il titolo, può solo ricercare e visualizzare schede presenti nell'applicazione.

11.4.2 Opzioni e casi d'uso

Opzione offerta:

Figura 11.3: Diagramma dei casi d'uso per lo Schedatore e il Validatore.



- ricerca nel database.

Il diagramma dei casi d'uso si trova in Figura 11.2.

11.5 Opzioni per lo Schedatore

11.5.1 Generale

Lo Schedatore è colui il quale popola l'applicazione con le schede. È anche il responsabile della proposta per la trasformazione da bozza a documento e per l'aggiornamento di un documento e la successiva trasformazione in bozza.

11.5.2 Opzioni e casi d'uso

Uno Schedatore dispone delle seguenti opzioni:

- creazione di una bozza;
- compilazione e aggiornamento di una bozza (comprende associare etichette e un'immagine alla scheda);
- cancellazione di una bozza;
- creazione di un'etichetta
- modifica di un'etichetta
- cancellazione di un'etichetta
- proposta di promozione di una *bozza* in *documento*;
- modifica di una scheda in stato *obsoleta*;
- ricerca nel database.

Il diagramma dei casi d'uso si trova in Figura 11.2.

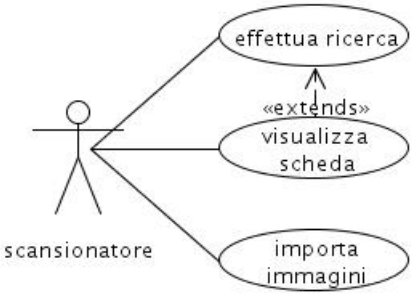
Se durante la compilazione o successiva modifica di una scheda viene lasciato vuoto un campo obbligatorio, l'utente verrà avvertito – dopo l'invio della scheda – dall'evidenziazione in rosso dei campi obbligatori.

Il collegamento fra la scheda e l'immagine scansionata è automatico e avviene tramite la ricerca di un'immagine importata da uno Scansionatore nell'area dedicata al `SupportoFisico` della scheda e con nome del file pari al campo `FTAN(.jpg)`.

Le operazioni di creazione e di eventuali successive modifiche di una bozza offrono anche la possibilità di applicare o rimuovere delle etichette definite dallo Schedatore proprietario della bozza stessa.

La rinominazione e cancellazione delle etichette viene effettuata tramite la vista fornita da Zope per l'oggetto `ContenitoreEtichette`, che consente di selezionare le voci da cancellare e poi premere il pulsante "Cancella" o quella da

Figura 11.4: Diagramma dei casi d'uso per lo Scansionatore.



rinominare e poi premere il pulsante "Rinomina". La creazione di un'etichetta avviene tramite il menu a tendina in alto a sinistra nella pagina .

La proposta di promozione di una scheda da *bozza* a *documento* avviene tramite l'apposita opzione nel menù sulla sinistra – che appare solo quando viene visualizzata una scheda in stato *bozza*; similmente la modifica (e quindi l'importazione nella propria area di lavoro) di un *documento*, ovvero da aggiornare, è resa possibile tramite l'apposita opzione della stessa area della precedente quando viene visualizzata una scheda marcata come *obsoleta*. Nel caso in cui la promozione venga rifiutata da un Validatore, nel campo `Note` della scheda verrà riportato il commento inserito dal Validatore; sempre nel campo `Note` sono presenti le indicazioni inserite dal Validatore riguardo l'aggiornamento della scheda.

11.6 Opzioni per lo Scansionatore

11.6.1 Generale

Lo Scansionatore è il responsabile per la scansione e l'importazione delle immagini nell'applicazione.

Le immagini *devono* essere scansionate e importate in formato JPEG.

11.6.2 Opzioni e casi d'uso

Le opzioni disponibili per lo scansionatore sono:

- importazione delle immagini;
- ricerca nel database.

Il diagramma dei casi d'uso si trova in Figura 11.4.

Concerne allo Scansionatore anche un'altra azione non mostrata in Mabon: la copia delle immagini scansionate nella cartella di rete – o comunque condivisa – delle immagini scansionate.

Per importare un'immagine, occore copiare il file desiderato in un'apposita cartella condivisa per il / dal server Mabon. Nella cartella condivisa principale `immagini` dell'applicazione, si trovano varie sottocartelle (`Lastra A`, `Pellicola 6x6`, `Digitale`) e il nome di ciascuna è associata a un supporto fisico. Ciascun file `.jpg`, contenente un'immagine da importare, va copiato nella sottocartella corrispondente. Il nome del file deve essere identico al campo `FTAN` della scheda alla quale si vuole associare l'immagine; infatti il collegamento tra la scheda e l'immagine viene effettuato dall'applicazione tramite la coppia <`FTAN`, `SupportoFisico`> della scheda e <`nomefile`, `cartellaDiImportazione`> del file `.jpg`.

Ad esempio, per importare un'immagine JPEG in maniera che venga associata alla scheda con campi <`543`, `Digitale`>, il file dovrà essere localizzato e chiamarsi come segue: `/immagini/Digitale/543.jpg`.

L'intero processo di importazione è quindi riassumibile come: copia delle immagini da importare nelle apposite cartelle, autenticazione dello Scansionatore in Mabon, selezione l'opzione di importazione immagini e click su "Importa". Viene mostrata una piccola anteprima di ciascuna immagine da importare.

Nel caso in cui nell'applicazione sia già presente un'immagine con stesso `FTAN` e `SupportoFisico`, tale immagine verrà visualizzata per aiutare lo Scansionatore a decidere se sovrascriverla o se rinominare il file in fase di importazione (da sistema operativo, non da Mabon).

Tutte le immagini da importare, che non presentino un'omonima già importata, sono già preselezionate.

Le immagini importate vengono automaticamente cancellate dalla cartella condivisa.

11.7 Opzioni per il Validatore

11.7.1 Generale

Il Validatore ha due differenti aree di azione: il vaglio (con conseguente approvazione o rifiuto) delle bozze proposte dagli Schedatori e gli oneri dell'eventuali cancellazione e segnalazione della necessità di aggiornamento di documenti.

11.7.2 Opzioni e casi d'uso

Le opzioni disponibili per un validatore sono:

- approvazione o rigetto di una *bozza*;
- proposta di modifica di un *documento*;
- cancellazione di un *documento*;
- ricerca nel database.

Il diagramma dei casi d'uso si trova in Figura 11.2.

Quando il Validatore visualizza una scheda, come risultato di una ricerca, nella barra laterale della pagina appaiono alcune azioni in funzione dello stato della scheda.

Una scheda in stato *documento* offrirà la possibilità di essere marcata per l'aggiornamento oppure per la cancellazione. La prima opzione offre un piccolo riquadro il cui contenuto verrà copiato nel campo `Note` della scheda e contiene le indicazioni per lo Schedatore che effettuerà la modifica.

Una scheda in stato *proposta* offrirà la possibilità di essere approvata o rifiutata. La seconda opzione offre un piccolo riquadro il cui contenuto verrà copiato nel campo `Note` della scheda e contiene le motivazioni della bocciatura, utili allo Schedatore.

Parte IV

Considerazioni finali e Ringraziamenti

Capitolo 12

La documentazione di Zope

All'inizio della tesi ho subito riscontrato una notevole difficoltà a trovare della documentazione che permettesse di imparare Zope 3 dalle basi e ancora meglio dei tutorials aggiornati e approfonditi.

Vi sono tre libri su Zope 3, [Ric05], [Wei08] e [Bai08], ma nessuno dei essi svolge una vera e propria funzione didattica che consenta al programmatore di affrontare agevolmente la ripida curva di apprendimento di Zope 3, spiegando chiaramente non solo la filosofia della programmazione in Zope, ma anche tutte le interazioni dietro le quinte che avvengono fra i componenti. In particolare, manca un'approfondita e dettagliata presentazione delle API, una Wiki ed esempi su come interconnettere tra loro le componenti, con casi pratici e magari una descrizione di lunghezza superiore ad un rigo; in questo ambito la capillare ed esaustiva documentazione di PHP dovrebbe essere presa come esempio.

Molti componenti di Zope interagiscono tra loro e presuppongono che determinati metodi o attributi delle classi abbiano un identificativo obbligatorio o predeterminato, ovviamente in inglese. Questa informazione spesso non è riportata sui libri [Wei08] e [Ric05], dal momento che tutta la documentazione e il codice di Zope è in inglese; al contrario, la lingua utilizzata nel mio codice è l'italiano e questo è stato frequentemente causa di malfunzionamenti e oscuri messaggi di

errore.

Questi problemi mi hanno rallentato sin dall'inizio, dato che l'unico modo che esiste per esplorare e conoscere nel dettaglio le interfacce – e quindi funzionalità offerte da Zope – è quello di esaminare il contenuto dei vari file con un editor di testo usando come scarno indice l'esigua documentazione offerta da `http://apidoc.zope.org/++apidoc++/` e di utilizzare il debugger `pdb` a linea di comando.

L'unica oasi in questo deserto è stata la mailing list zope3–users@zope.org, dove gli sviluppatori di Zope3 e i programmatori più esperti sono sempre stati gentili e solerti nel rispondere alle domande che vengono poste.

D'altra parte la struttura ad oggetti di Zope 3 e le ricche librerie fornite, velocizzano enormemente il lavoro di progettazione e implementazione di un sito web (e in particolare di Mabon II).

Capitolo 13

Ringraziamenti

Desidero ringraziare per la cortese disponibilità:

- Andrea Domenici, professore della Facoltà di Ingegneria dell'Università di Pisa e mio primo Relatore Interno;
- Giovanni Stea, professore della Facoltà di Ingegneria dell'Università di Pisa e mio secondo Relatore Interno;
- Clara Baracchini, Vice Soprintendente della Soprintendenza BAPPSAE e mia prima Relatrice Esterna;
- Umberto Parrini, Responsabile Sviluppo Software per la Scuola Normale Superiore e mio secondo Relatore Esterno;
- Christian Lück, Shailesh Kumar, Jim Fulton, Stephan Richter, Tim Cook, Christophe Combelles per l'aiuto offertomi sulla mailing list zope3-users;
- Severina Russo: Responsabile Fototeca e Catalogo Gallerie della Soprintendenza BAPPSAE;
- MAriateresa BONanotte (si è questa l'origine del nome dell'applicazione), per tutto l'aiuto offertomi;
- la mia famiglia;

- i miei amici della mailing list gdtapisa@yahoogroups.com per lo svago;
- i miei compagni di danza del Circolo Pisano Società di Danza per gli incoraggiamenti.

Appendice A

Elenchi

Elenco degli algoritmi

5.1 da `Mabon.sito` 54

6.1 Indice per la ricerca semplice 65

6.2 Indice per la ricerca avanzata 65

6.3 Il corpo del metodo `create( dati )` 78

6.4 Il metodo `create` della classe `ModuloAggiungiUtente` 82

Elenco delle figure

2.1 Rappresentazione macroscopica del framework Zope 3 e dell'applicazione Mabon. 13

2.2 Diagramma di attività con le fasi di traversing e publishing. . . . 16

2.3 Diagramma di classe di un adattatore. 19

3.1 Stati di una scheda 33

3.2 Attività effettuabili su una scheda 34

4.1 Diagramma dei pacchetti 40

4.2 Pacchetto `mabon` 41

4.3 Pacchetto `browser` 42

4.4 Pacchetto `skin` 42

4.5 Pacchetto `risorse` 43

4.6 Pacchetto `etc` 43

4.7 Pacchetto `locales` 44

5.1 Diagramma delle classi `Contenitore*` 47

5.2 Diagramma della classe `Etichetta` 50

5.3 Diagramma della classe `SitoMabon` 53

5.4 Diagramma della classe `SitoMabon` 55

5.5 Diagramma della classe `Etichetta` 56

5.6 Diagramma della classe `Utente` 59

6.1 Diagramma di attività della funzione `autenticazione_e_principali` 63
6.2 Diagramma di attività della funzione `cookie_e_sessioni` 64
6.3 Diagramma di attività della funzione `catalogo_e_indici` 66
6.4 Diagramma di attività della funzione `catalogo_e_indici` 69
6.5 Diagrammi della classi `ModuloModificaScheda` e `ModuloModificaUtente` 76
6.6 Diagramma della classe `ModuloAggiungiUtente` 78
6.7 Diagramma della classi `VisualizzaScheda` e `VisualizzaUtente` . 80
6.8 Diagramma della classe `PaginaDiRicerca` 85
6.9 Diagramma della classe `ModificaStatoScheda` 88
6.10 Diagramma della classe `OpzioniModificaStato` 93
6.11 Stati di una scheda 95

7.1 Diagramma di struttura per i componenti `AreaUtente`, `ContenitoreUtenti`, `ContenitoreScheda`, `ContenitoreEtichetta`, `SincronizzaNomeUtente`, `SincronizzaNomeScheda` e `SincronizzaNomeEtichetta`. 102
7.2 Diagramma di struttura per i componenti `Dimensione Principale`, `DimensioneScheda` e `DimensioneUtente`. 103
7.3 Diagramma di struttura per i componenti `ElencaImmagini` e `ImportaImmagini`.103
7.4 Diagramma di struttura per i componenti `aggiornaDCdaEtichetta`, `aggiornaDCdaScheda` e `aggiornaDCdaUtente`. 104
7.5 Diagramma di struttura per i componenti `vocabolarioEtichette` e `vocabolarioFondi`. 104
7.6 Diagramma di struttura per i componenti `generaTitoloScheda`, `aggiornaTitoloScheda`, `generatoreScheda`, `Scheda`, `VisualizzaScheda`, `ModuloModificaScheda` e `ModuloAggiornaScheda`. 105
7.7 Diagramma di struttura per i componenti `generaTitoloUtente`, `aggiornaTitoloUtente`, `generatoreUtente`, `Utente`, `VisualizzaUtente`, `ModuloModificaUtente` e `ModuloAggiornaUtente`. 106

7.8 Diagramma di struttura per i componenti `SitoMabon`, `EventoNuovoSitoMabon`, `autenticazione_e_principali` e `cookie_e_sessioni`. 107

7.9 Diagramma di struttura per i componenti `vocabolarioImmaginiNuove`, `vocabolarioImmaginiImportate`, `vocabolarioSupporti`, `vocabolarioFondi` e `vocabolarioRuoli`. 108

8.1 Diagramma delle relazioni relativo al componente Utente 111

8.2 Diagramma delle relazioni relativo al componente Scheda 113

9.1 Diagramma di interazione per la creazione di un oggetto Utente. . 122

9.2 Diagramma di interazione per una ricerca semplice 125

11.1 Diagramma dei casi d'uso per l'Amministratore. 136

11.2 Diagramma dei casi d'uso per il Ricercatore 137

11.3 Diagramma dei casi d'uso per lo Schedatore e il Validatore. 138

11.4 Diagramma dei casi d'uso per lo Scansionatore. 140

Elenco delle tabelle

5.1 Corrrispondenze tra contenitori, contenuti e adattatori. 48

6.1 Tabella delle azioni che modificano lo stato di una scheda 94

Appendice B

Bibliografia

[Bai08] Muthukadan Baiju. A Comprehensive Guide to Zope Component Architecture. -, 2008. http://www.muthukadan.net/docs/zca.html. [.html]

[BarLavPia06] Luciano Baresi, Luigi Lavazza, and Massimiliano Pianciamore. Dall'idea al codice con UML 2. Pearson Education. Pearson, 2006.

[GamHelJohVli08] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison Wesley Prefessional Computing Series. Addison Wesley, 2008.

[KopDal07] Helmut Kopka and Patrick W. Daly. Guide to Latex, Fourth Edition. Addison Weslry, 2007.

[NeuArl07] Ila Neustadt and Jim Arlow. UML 2 e Unified Process 2a edizione. Workbooks. McGraw-Hill, 2007.

[Ric05] Stephan Richter. Zope 3 Developer's Handbook. Developer's Library. Sams Publishing, 2005.

[Sta03-01] Richard Matthew Stallman. Software Libero Pensiero Libero. Volume Primo, volume 1 of Eretica. Stampa Alternativa, 2003.

[Sta03-02] Richard Matthew Stallman. Software Libero Pensiero Libero. Volume Secondo, volume 2 of Eretica. Stampa Alternativa, 2003.

[W3S] Online web tutorials: http://www.w3schools.com/

[Wei08] Philipp von Weitershausen. Web Component Developing with Zope 3, Third Edition. Springer, 2008.