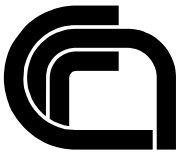

Consiglio Nazionale delle Ricerche

Archele, a web based acquisition, search and retrieval system

M. Andreini, C.Lucchesi, M. Martinelli, G. Vasarelli

IIT TR-09/2007

Technical report

Luglio 2007

Istituto di Informatica e Telematica

ArchEle, a web based acquisition, search and retrieval system

Marco Andreini, Cristian Lucchesi,
Maurizio Martinelli, Giuseppe Vasarelli

Istituto di Informatica e Telematica (IIT)
Italian National Research Council (CNR)
Via G. Moruzzi, 1 - 56124 PISA, Italy
{marco.andreni, cristian.lucchesi, maurizio.martinelli, giuseppe.vasarelli}@iit.cnr.it

Abstract. *The problem of filing and storing paper and electronic documents is of great concern for public administration as well as private companies. We introduce a system that allows acquisition of images by means of one or more computer equipped with scanners, and associated with metadata for the subsequent search for and storage of data in a Document Repository protected by a system of access control lists (ACL).The system we describe here allows filing, search and retrieval of millions of documents by means of web interfaces. The architecture is fully compliant with web standards and with its design principles. The system we describe is called ArchEle (in Italian, Archiviazione Elettronica).*

Keywords: data storage, CMS, information retrieval, web standards

ACM Classification: H.3.2 Information Storage

Indice generale

1 Introduzione....3

2 Le tecnologie dell'applicazione....4
2.1 Una panoramica....4
2.2 L'interfaccia Web: XHTML, CSS e PNG....5
2.3 L'application server Zope....6
2.3.1 Cosa fornisce Zope? 6
2.4 Il content management system Plone....8
2.4.1 Gli skin....9
2.5 Archetypes....10
2.5.1 Field....10
2.5.2 Widget 12
2.5.3 Archetypes come base di ArchEle....13
2.6 Lo storage dei dati: PostgreSQL e il Filesystem....13
2.7 Il client di acquisizione....14

3 ArchEle....15
3.1 Funzionalità....15
3.2 Limiti da superare....16
3.3 Ottenere la scalabilità richiesta....18
3.3.1 ArcheleBaseType....18
3.3.2 SQLStorageOnly....20
3.3.3 ExtFolder....21
3.4 Trasformazioni aggiuntive....22
3.5 Skin....22
3.5.1 Creazione degli oggetti....22
3.5.2 Upload delle immagini 22
3.5.3 Lancio dell'acquisizione....23
3.6 Il client: Isaac....23
3.7 Case Study....25
3.7.1 I Cambi MNT e le LAR....26
3.8 Integrazione con i dati del Registro....29
3.8.1 Interfacce per l'integrazione....29
3.8.2 Gli aggiornamenti dei dati....31
3.8.3 I cambi Multipli....32
3.9 Verifiche....33
3.10 Integrazione con gli altri dati del Registro....33

4 La scalabilità dell'applicazione....34

5 Sviluppi futuri....36
5.1 Personalizzazioni via Web e UML....36
5.2 Full Text Search....36
5.3 Workflow documentale....37
5.4 Firma elettronica dei documenti....37

1 Introduzione

Sono ancora molte le realtà che non hanno ancora soluzioni facilmente usabili, configurabili e personalizzabili per archiviare grosse quantità di dati cartacei e non.

L'obbiettivo del software sviluppato è quello di creare una piattaforma che permetta da una parte di acquisire i documenti cartacei del Registro del ccTLD ".it" (*Registro* nel resto del documento) tramite alcune postazioni dotate di scanner, dall'altra di rendere facilmente consultabile l'archivio dei documenti acquisiti.

Il software genera le interfacce di consultazione e inserimento in modo automatico ed è stato progettato per rendere molto veloce la definizione di nuovi tipi di documento da acquisire.

L'applicazione è anche in grado di supportare con buone prestazione l'acquisizione e la consultazione di milioni di documenti.

L'applicazione nasce inoltre con il supporto per l'internazionalizzazione, quindi è possibile aggiungere le traduzioni e renderla multilingua; già nella prima fase l'applicazione è fruibile sia in italiano che in inglese.

Al momento della scrittura di questo documento il software è utilizzato per l'archiviazione delle lettere di assunzione responsabilità (LAR), i cambi Maintainer, i cambi Maintainer e Registrante e le lettere di "No Provider" del Registro dei nomi a dominio.

Sono archiviate nell'applicazione 213.500 LAR e 513.500 cambi Maintainer, 105.000 cambi Maintainer e registrante e 118.500 lettere di "No Provider", per un totale di oltre 950.000 documenti.

2 Le tecnologie dell'applicazione

2.1 Una panoramica

L'applicazione è stata sviluppata seguendo un approccio di tipo client – server basato su tecnologie Web.

Per permettere l'inserimento, l'aggiornamento e la consultazione dei dati associati ai documenti sì è sviluppata l'interfaccia XHTML [XHTML] con la quale diversi utenti, con i diritti necessari, operano attraverso i client. È stato inoltre sviluppato un client grafico che permette, in accordo con i dati forniti dal server, di acquisire pagine dallo scanner e di inviarli al server.

Sul lato server è stata invece predisposta la parte dell'applicazione che si occupa di archiviare i documenti trasmessi dalle varie postazioni client, di indicizzare i campi inseriti dai vari operatori e di creare le interfacce per l'aggiornamento e la consultazione dei dati presenti.

Per quanto riguarda l'interfaccia di inserimento e di interrogazione la scelta è stata quella di utilizzare le tecnologie web di presentazione dei contenuti: l'XHTML ed i CSS.

Questa scelta porta con sé il vantaggio di essere multipiattaforma, consultabile da qualunque postazione abilitata, senza l'ausilio di software aggiuntivi rispetto ad un browser qualunque compatibile con le attuali specifiche riguardanti XHTML1.0, CSS2 (Cascading Style Sheet level 2) [CSS2] e PNG (Portable Network Graphics) [PNG].

Il client grafico che si occupa del trasferimento delle immagini acquisite con lo scanner sul server è stato sviluppato in Python, con il supporto del modulo wxPython [WXPYTHON], del modulo TWAIN [TWAINPY] e della PIL (Python Imaging Library) [PIL].

Il client ad oggi funziona solo su piattaforma Windows in quanto il modulo TWAIN è presente solo per questo sistema operativo, ma potrebbe facilmente essere portato su altri sistemi operativi per esempio con l'utilizzo del modulo SANE (per Linux). Il resto dei moduli utilizzati e il Python sono multipiattaforma quindi funzionano sulla maggioranza delle piattaforme odierne.

Dal lato server è stato scelto di usare Linux per la sua robustezza, la sua affidabilità a livello server e molte altre caratteristiche. In particolare è stata scelta la distribuzione Debian GNU/Linux [DEBIAN]. I punti fondamentali per questa scelta sono stati: la larga disponibilità di pacchetti software sempre aggiornati, la facilità di manutenibilità di larghe collezioni di software e l'accurata politica di gestione delle configurazioni che semplifica gli aggiornamenti.

La parte applicativa è stata implementata con l'application server Zope (Z Object Publishing Environment) [ZOPE], che mette a disposizione un framework ad oggetti per la progettazione e la realizzazione di web application.

Zope inoltre mette a disposizione una serie di prodotti che forniscono funzionalità che erano necessarie per la buona riuscita del prodotto.

Alla base dell'applicazione appoggiata su Zope è stato posto il Content Management System Plone [PLONE] con una serie di prodotti che forniscono un sistema granulare di gestione degli utenti; un efficiente sistema di validazione dei contenuti inseriti, la creazione di codice riutilizzabile e molto altro. Tutto questo tramite la consolidata interfaccia ad oggetti di Zope.

È stato scritto un apposito prodotto – ArchEle – che fornisce le funzionalità di base necessarie alla gestione degli oggetti da salvare sul server.

Per descrivere l'architettura e l'interazione dei vari componenti si descriveranno i seguenti argomenti:

- scelte sui formati usati per l'interfaccia utente;
- descrizione dell'application server Zope e delle sue funzionalità;
- descrizione del content management system Plone;
- descrizione dei tipi di documenti dinamici su Plone con Archetypes, con dettaglio su:
  - i campi dei documenti con i Field;
  - i campi delle interfacce con i Widget;
- introduzione ad ArchEle;
- l'uso di PostgreSQL come database e i Filesystem per milioni di file;
- il client per la digitalizzazione e la memorizzazione.

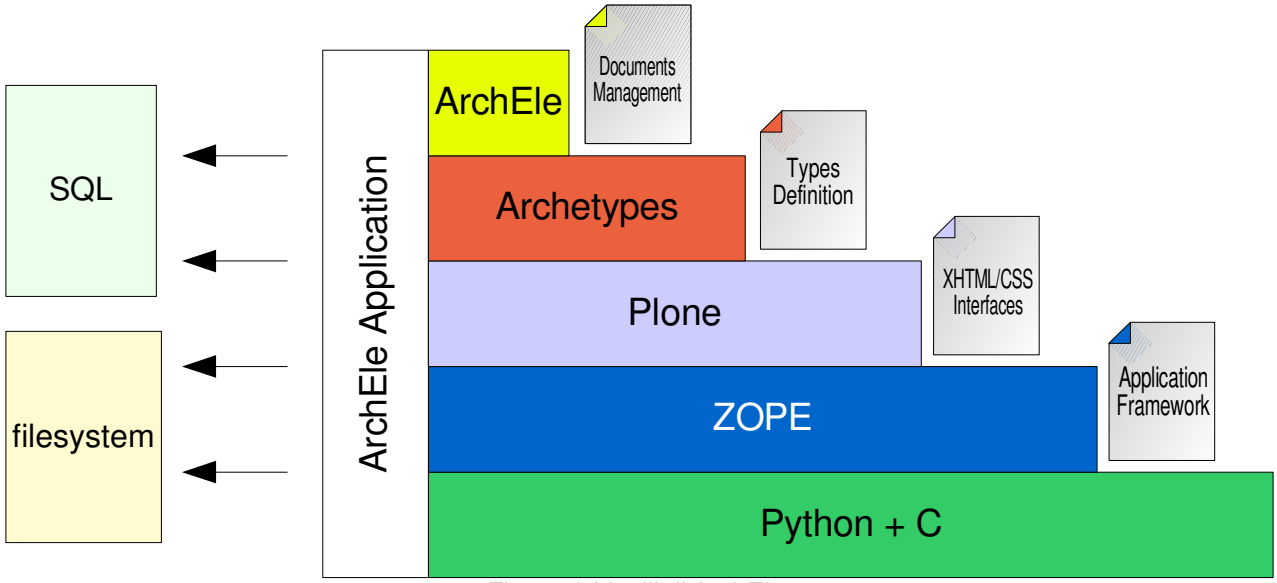

Figura 1-Livelli di ArchEle

La Figura 1 mostra i livelli della business logic di ArchEle.

2.2 L'interfaccia Web: XHTML, CSS e PNG

Il web e le sue tecnologie consentono un accesso di tipo universale [W3C-7P] alle informazioni, fruibili attraverso dispositivi compatibili con gli standard del web. Scrivere una applicazione tramite un'interfaccia web consente inoltre bassi costi di manutenzione lato client che non ha bisogno di essere modificato (ed eventualmente reinstallato ogni qual volta sia necessario modificare il software).

Per tale motivo l'interfaccia utente è stata realizzata utilizzando il linguaggio XHTML 1.0 + CSS2.

Per la visualizzazione delle immagini acquisite è stato scelto di utilizzare il formato PNG in quanto:

- è integrato in tutti i browser odierni;
- ha un ottimo algoritmo di compressione (usato anche nel gzip e nello zip)
- ha una buona resa grafica.

Altri formati, come ad esempio il TIFF G3, avrebbero permesso di risparmiare spazio disco, ma non avrebbero permesso la loro visualizzazione diretta nei browser, visto che questi formati non sono supportati dai browser stessi.

2.3 L'application server Zope

Come base per la nostra applicazione è stato scelto l'application server Zope che mette a disposizione un ambiente ad oggetti per la creazione di robuste applicazioni, facilmente integrabili ed è scritto in Python [PYTHON] e ciò permette di interfacciarsi a tutte le librerie disponibili per questo linguaggio. È inoltre modulare e quindi estremamente estendibile.

Zope è un *framework* per costruire *applicazioni web,* cioè applicazioni che usano il web per presentare la loro interfaccia utente, permettendo di utilizzarle con la stessa facilità del web stesso, perché in effetti sono quasi indistinguibili da esso. In termini più pratici è un prodotto che aiuta nella creazione, nella gestione e nel servizio di complessi siti dinamici.

Cominciando a progettare una applicazione web usando una piattaforma che fornisce già gli strumenti per la costruzione di applicazioni web (invece delle sole funzionalità di un linguaggio di programmazione come Perl, Python, TCL, Java o C), è possibile appoggiarsi ai servizi che la piattaforma mette a disposizione, evitando così di scrivere da zero alcune funzionalità come con un tradizionale linguaggio di programmazione.

Inoltre Zope può essere utilizzato sulla maggior parte delle piattaforme esistenti: Unix, Linux, Mac OS, e le varianti di Windows (98, NT, 2000, XP).

2.3.1 Cosa fornisce Zope?

Zope è costituito da più moduli che interagendo tra di loro permettono di offrire vari servizi. I più rilevanti che Zope mette a disposizione sono:

- **Un server web**

Zope contiene già un server web in grado di supportare tutte le funzionalità necessarie al corretto funzionamento delle applicazioni sviluppate. Zope comunque mette a disposizione anche dei meccanismi per utilizzare un server web esterno nel caso in cui si desideri ad esempio aumentare le prestazioni, affidarsi a software specializzati, o nel caso in cui si desiderino fornire anche altri servizi. Per questo scopo Zope supporta l'interfaccia PCGI, FastCGI ed in più ha un meccanismo per il Virtual Hosting che rende possibile il suo uso attraverso un Server Web che svolga le funzioni di proxy http.

- **Un database ad oggetti**

La grande differenza tra Zope e prodotti analoghi è il fatto che Zope ha trasformato il Web in una struttura ad oggetti (Zope: Z Objects Publishing Environment) e quindi contiene al suo interno un database ad oggetti nel quale sono immagazzinati gli oggetti che compongono le applicazioni che devono essere servite.

- **Un'interfaccia di amministrazione**

Zope fornisce una completa interfaccia utente che può essere utilizzata per la gestione e lo sviluppo delle applicazioni. L'interfaccia può essere usata da qualsiasi browser web moderno e si presenta in modo molto familiare agli utenti, visualizzando la gerarchia degli oggetti tramite una struttura ad albero, in modo simile a come i più diffusi file manager visualizzano i filesystem.

• Integrazione con database relazionali

I dati necessari alle applicazioni possono essere memorizzati direttamente nel database ad oggetti di Zope - ZODB[1], ma ciò non è strettamente necessario. È infatti possibile sostituire il database ad oggetti con un qualsiasi DBMS relazionale per il quale sia disponibile un driver ODBC o uno Zope Adapter[2].

• Linguaggi di scripting

Zope mette a disposizione alcuni linguaggi per sviluppare le diverse parti delle applicazioni: per lo sviluppo dei modelli di presentazione dei documenti è possibile scegliere se utilizzare DTML(Document Template Markup Language) o ZPT (Zope Page Template). Il DTML è un linguaggio di markup integrato con HTML, che ne espande le possibilità permettendo di accedere agli oggetti di Zope. Gli ZPT (Zope Page Template) utilizzano i *namespace* XML per ottenere un ambiente separato, ma integrato nel codice XHTML: così si ha un linguaggio di templating che sfrutta le caratteristiche del Python per lo stretto indispensabile nella presentazione.

Per lo sviluppo della logica applicativa si utilizza il Python, che è il linguaggio nativo di Zope ed in cui è anche scritto. È però possibile l'integrazione con altri linguaggi: ad esempio esiste una possibile sinergia con Perl. La Figura 2 mostra l'archiettura di Zope.

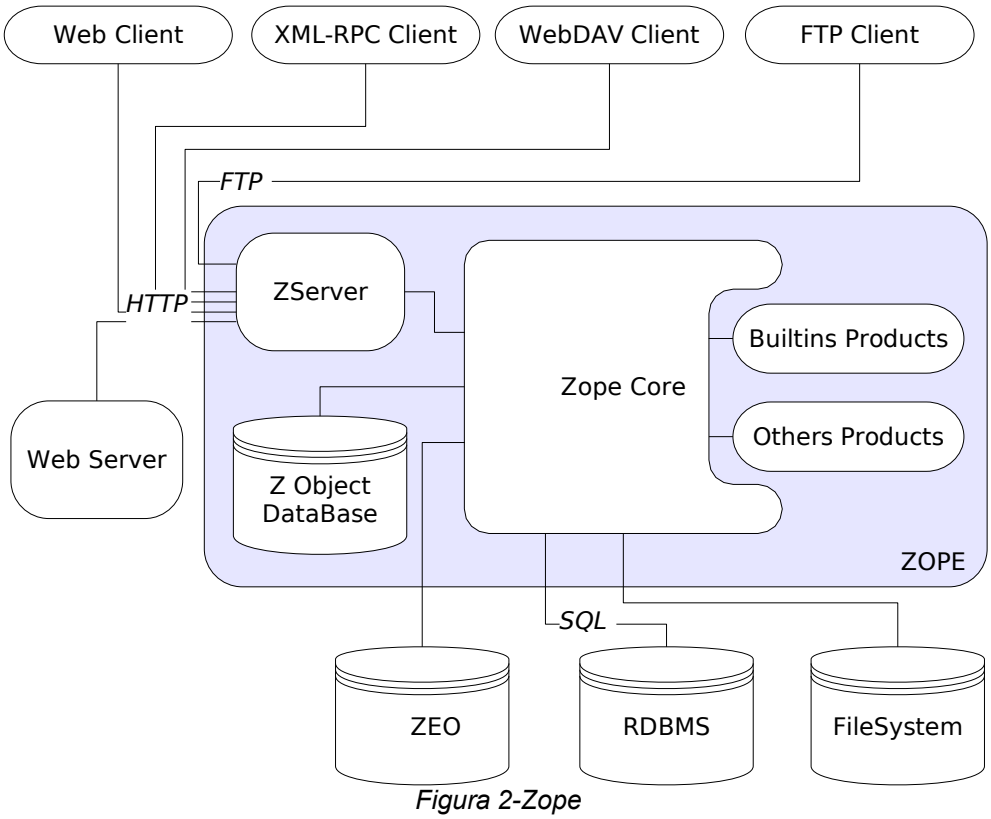

Figura 2-Zope

1 Lo ZODB - Zope Object DataBase - è un database ad oggetti che memorizza i risultati di tutte le transazioni.

2 Gli Zope Adapter sono il meccanismo con il quale si connettono i Database esterni (SQL, LDAP, ...) all'applicativo.

2.4 Il content management system Plone

Plone è un sistema di gestione dei contenuti (Content Management System) libero (free) ed a sorgente aperto (Open Source).

Plone aggiunge a Zope alcune funzionalità particolarmente importanti:

- un efficace ed elegante framework per la navigazione, basato su cartelle e sul loro contenuto piuttosto che su collegamenti tra documenti html;
- uno strumento semplice per la creazione di documenti strutturati anche complessi;
- un sistema di gestione di utenti e gruppi di utenti;
- grafica e template per la presentazione dei contenuti facilmente personalizzabili basate su tecnologie XHTML e CSS, con particolare cura dell'accessibilità e dell'usabilità degli strumenti;
- interfacce web per l'amministrazione dei siti (realizzati con Plone);
- un sistema di indicizzazione (portal_catalog) integrato che permette facili e potenti ricerche su documenti strutturati diversamente;
- indicizzazione di file in vari formati: .doc, .xls, .sxw, .ppt, .pdf, etc;
- strumenti per il controllo di flusso documentale (Workflow);
- strumenti per il controllo degli accessi e la configurazione di “ruoli”;
- una serie di tipi di contenuti strutturato (News, Documenti, Cartelle, File, ...);
- una interfaccia multilingua;
- una larga comunità di utilizzatori, sviluppatori e autori in tutto il mondo che contribuiscono al suo sviluppo e miglioramento.

In definitiva uno strumento che introduce una elevata astrazione e molte funzionalità e che può costituire la base per un ampio spettro di applicazioni web, partendo dal Content Management, ma andando anche in molte altre direzioni.

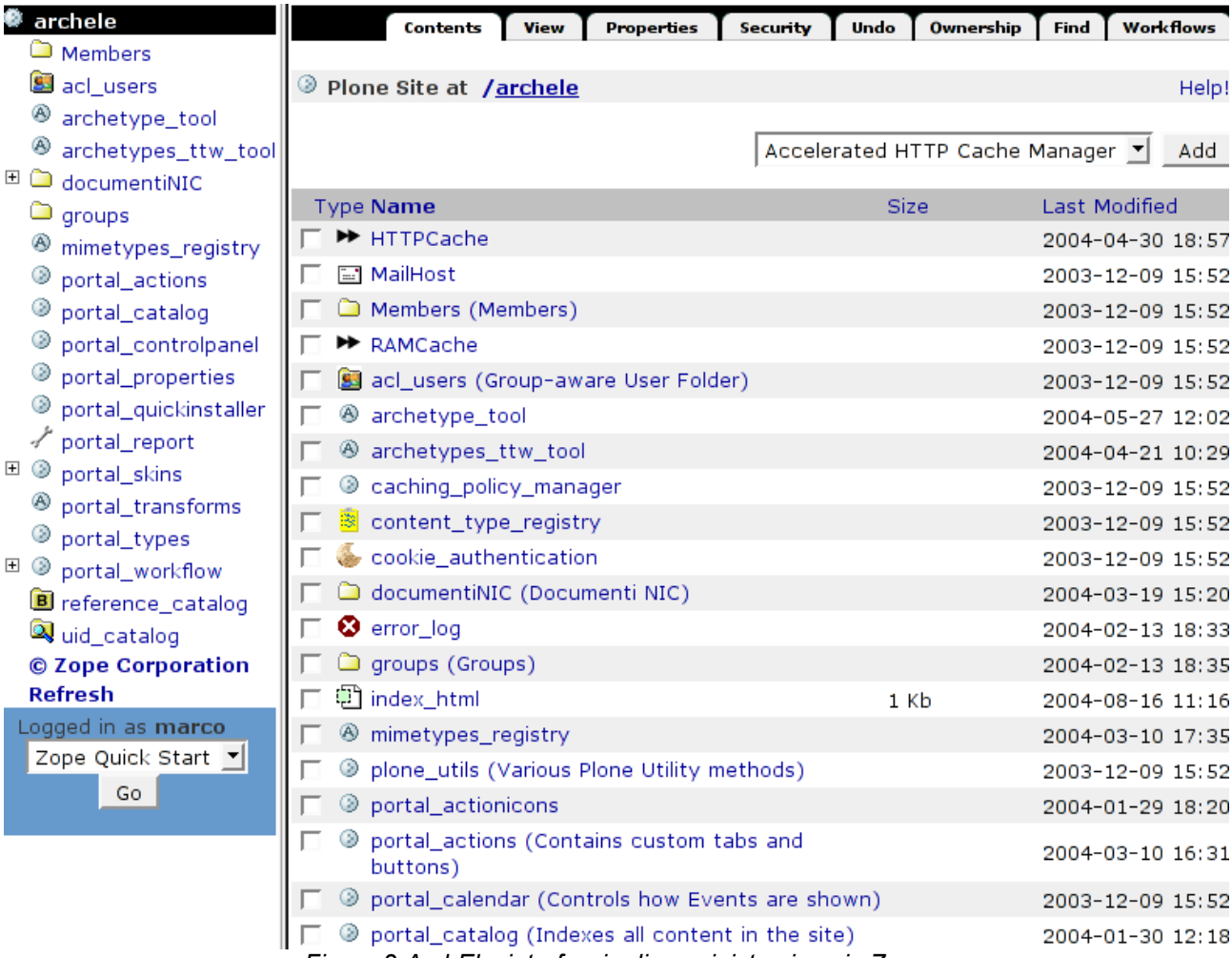

Figura 3-ArchEle: interfaccia di amministrazione in Zope

2.4.1 Gli skin

Uno *skin* è un *guscio* che racchiude di solito piccole porzioni della logica relativa alla presentazione. L'aspetto grafico di Plone è personalizzabile tramite skin (in effetti anche il layout predefinito, usato da plone.org, è contenuto all'interno di skin). Definendo un nuovo skin si riesce a modificare radicalmente la presentazione dei contenuti, pur mantenendo le funzionalità del Content Management.

Nel caso si utilizzino più skin si può dare la possibilità all'utente di scegliere la grafica che preferisce, lasciando inalterati il comportamento del software e i contenuti del sito. In taluni casi è anche preferibile utilizzare la flessibilità e la completezza dell'interfaccia standard di Plone per la modifica e l'inserimento dei contenuti del sito, fornendo una grafica per la visualizzazione al pubblico esterno, completamente diversa e personalizzata.

Gli skin sono utilizzabili tramite lo strumento *portal_skins (CMF)*.

Utilizzando gli skin sono state scritte tutte le nuove interfacce e le modifiche a quelle esistenti necessarie ad ArchEle, utilizzando gli ZPT e laddove fosse necessario, gli script python. Tutto ciò ha reso possibile lavorare con XHTML fornendo template che fossero dinamici, adattabili ai vari tipi di dato, e facilmente mantenibili.

2.5 Archetypes

La gestione documentale fornita con Plone [PLONEBK] è strutturata su pochi tipi di documenti (News, File, Eventi, Immagini, ...) ed è insufficiente per poter gestire oggetti strutturati in modo diverso da quello dei tipi predefiniti. La scelta fatta allo scopo di affrontare questa problematica è quella di utilizzare Archetypes.

Archetypes è modulo di software progettato per la costruzione di applicazioni basate su Plone/CMF (Content Management Framework). La sua caratteristica principale è di fornire un architettura comune per la costruzione di oggetti strutturati. Questa costruzione è basata sulla definizione dello schema del tipo.

I punti di forza di Archetypes sono:

- generazione automatica di form;
- librerie di tipi di campo già pronte (field);
- librerie di widget, cioè elementi per la generazione di template XHTML;
- librerie di validatori dei campi;
- facile integrazione e personalizzazione degli elementi dello schema: field, widget e validatori.

Il cuore di un oggetto creato con Archetypes è il suo “schema” cioè la sequenza dei campi del tipo.

Archetypes fornisce alcuni schemi di base tra cui: BaseSchema, BaseFolderSchema, BaseBTreeFolderSchema. Ciascuno di questi schema include i due campi 'id' (identificativo) e 'title' (titolo), così come anche i metadati standard, definiti dal Dublin Core [DC].

Lo Schema rappresenta la definizione di cosa conterranno gli oggetti e come presentare le informazioni contenute.

Archetypes prende in consegna gli schemi delle oggetti registrati e automaticamente genera i metodi per accedere e modificare ognuno dei field definiti negli Schema.

2.5.1 Field

Il Field è il componente base degli Schema e permette di definire le proprietà dei vari campi di cui l'oggetto sarà composto. Queste proprietà vengono utilizzate per la generazione delle interfacce (di inserimento, di modifica, di cancellazione e di ricerca) e per la validazione dei valori da immettere.

Si può aggiungere un campo allo schema tramite uno dei tipi di field disponibili. In Archetypes mette a disposizione alcuni field:

'Field', 'ObjectField', 'StringField',

'FileField', 'TextField', 'DateTimeField', 'LinesField',

'IntegerField', 'FloatField', 'FixedPointField',

'ReferenceField', 'ComputedField', 'BooleanField',

'CMFObjectField', 'ImageField', 'PhotoField'.

Archetypes permette comunque lo sviluppo di nuovi field estendendo quelli predefiniti. [ARCHDG].

I field predefiniti condividono alcune proprietà che è possibile modificare al momento dell'istanziazione dell'oggetto. Suddette proprietà usate anche in ArchEle nella dichiarazione di campi sono:

- required
  Rende il campo obbligatorio durante la validazione.
- widget
  Il Widget utilizzato per la visualizzazione e la modifica del contenuto del field (si possono creare anche Widget personalizzati)
- default
  Imposta il valore di default del field nel momento della sua inizializzazione
- vocabulary
  Specifica un vocabolario da utilizzare (da specificare tramite un metodo che restituisce un DisplayList). Il contenuto del vocabolario è utilizzato come lista dei valori che possono essere scelti per riempire questo field.
- enforceVocabulary
  Quando impostato verifica, durante la validazione, se il valore da inserire è presente nel vocabolario assegnato tramite la proprietà vocabulary.
- multiValued
  Se impostato, il campo può avere valori multipli invece di un valore singolo
- isMetadata
  Se impostato il campo è considerato essere un metadato
- accessor
  Il nome del metodo utilizzato per accedere ai dati di questo field. Se il metodo non viene impostato Archetypes genererà un metodo base per l'accesso.
- edit_accessor
  Nome del metodo che verrà utilizzato per prelevare i dati per la modifica. A differenza dell'accessor che può effettuare trasformazioni prima di restituire i dati, questo metodo restituisce i dati “grezzi” senza alcuna trasformazione. Nel caso non sia indicata, Archetypes genera un proprio edit_accessor.
- mutator
  Nome del metodo utilzzato per modificare il valore del field.
- mode
  I possibili valori sono: `r`, `w` `rw`. Se il valore è `r`, sarà generato solo l'accessor. Se il valore è solo `w` verrà generato solo il mutator, altrimenti verranno generati entrambi i metodi.
- read_permission
  Permessi necessari per visualizzare il field.
- write_permission
  Permessi necessari per modificare il field.
- storage
  Uno degli storage presenti (quelli predefiniti oppure quelli creati dallo sviluppatore).
- validators
  Uno dei validatori presenti (quelli predefiniti oppure quelli creati dallo sviluppatore).
- index
  Una stringa che indica uno o più tipi di indice da utilizzare durante la fase di indicizzazione dei contenuti per mezzo del portal_catalog.
- schemata
  Gli Schemata sono utlizzati per raggruppare i field in insiemi (detti fieldset).

Questo è un esempio di schema (dai sorgenti di Archetypes: '*examples/SimpleType.py*'):

```
schema = BaseSchema + Schema((
  TextField("body",
        required=1,
        searchable=1,
        default_output_type="text/html",
        allowable_content_types=("text/plain",
```

```
                                "text/restructured",
                                "text/html",
                                "application/msword"),
        widget  = RichWidget,
        ),
  ))
```

2.5.2 Widget

I Widget sono dei metodi, sia predefiniti che parametrici, per la visualizzazione, la modifica e la ricerca dei Field.

Si possono creare nuovi Widget, personalizzare quelli esistenti oppure utilizzare quelli messi a disposizione che sono sufficienti per la maggior parte delle applicazioni.

Archetypes fornisce i Widget predefiniti per:

- Stringhe – *StringWidget*
- Numeri decimali – *DecimalWidget*
- Numeri Interi – *IntegerWigdet*
- Riferimenti – *ReferenceWigdet*
- Campi calcolati – *ComputedWidget*
- TextArea – *TextAreaWidget*
- Campi su più linee – *LinesWidget*
- Booleani – *BooleanWidget*
- Date (con calendari) – *CalendarWidget*
- Campi con selezione – *SelectionWidget*
- Campi con selezione multipla – *MultiSelectionWidget*
- Campi di Keyword – *KeywordWidget*
- Campi con rich text – *RichWidget*
- File – *FileWidget*
- Identificativi – *IdWidget*
- Immagini – *ImageWidget*
- Etichette – *LabelWidget*
- Password – *PasswordWidget*
- Campi visuali – *VisualWidget*
- Campi di testo modificabili tramite Epoz – *EpozWidget (editor HTML completamente javascript)*

L'associazione tra Field e Widget avviene come visto nei paragrafi precedenti nel momento della definizione dello “Schema”. Inoltre i nuovi Widget creati sono riutilizzabili all'interno di qualunque applicazione che faccia uso di Archetypes.

In modo coerente, la personalizzazione dei Widget e dei metodi di presentazione di Archetypes viene effettuata tramite il meccanismo degli skin già esposto.

2.5.3 Archetypes come base di ArchEle

La decisione di utilizzare Archetypes come base per la nostra applicazione, scegliendo implicitamente di utilizzare Zope e Plone nasce dall'analisi delle caratteristiche di Archetypes stesso. In particolare:

- la realizzazione automatica di interfacce;
- la validazione dei valori immessi;
- la facile strutturazione dei documenti;
- l'elevato livello di personalizzazione di tutti i metodi.

L'architettura di Archetypes non garantisce la scalabilità nell'immagazzinamento di milioni di oggetti. In particolare nell'uso di Archetypes vengono effettuate molte modifiche agli oggetti e soprattutto se questi sono di grosse dimensioni, rendono impraticabile l'uso dello ZODB [ADVZODB] per lo storage; è da notare come anche l'uso di storage “Non-ZODB” per i field (ad esempio SQLStorage; si veda [ARCHDG]) non risolva questo problema.

Per risolvere questo problema ed effettuare le altre personalizzazioni si è deciso di sviluppare un prodotto che estendesse Archetypes nelle sezioni relative allo storage; questo prodotto è stato chiamato **ArchEle**.

2.6 Lo storage dei dati: PostgreSQL e il Filesystem

Per immagazzinare i documenti si sono utilizzati un database relazionale e il filesystem: il database relazionale contiene i campi del documento usati anche per le ricerche (titolo, autore, data, ...), mentre il filesystem contiene i file (anche immagini) associati ai documenti.

Si è deciso di utilizzare il database relazionale PostgreSQL che garantisce la transazionalità delle operazioni necessaria nel caso di modifiche che non vanno a buon fine, il supporto per le statistiche sull'utilizzo del DB, i meccanismi di integrità referenziale, oltre alla sua robustezza e scalabilità.

Per lo storage delle immagini acquisite si è scelto tra i journalling filesystem per dare maggiore affidabilità al sistema; tra questi si è scelto ReiserFS [REISERFS](o in alternativa XFS [XFS]). L'architettura implementata usa una directory per ciascun documento, che contiene tutti i file ad esso associati.

La scelta di ReiserFS piuttosto che Ext3 si basa su due punti fondamentali:

- l'occupazione di spazio delle directory;
- l'ordine di grandezza dei tempi di accesso per la lettura e la scrittura degli elementi del filesystem.

Vista la necessità di creare un grosso numero di directory la scelta di ReiserFS permette di risparmiare molto spazio: le directory occupano mediamente 300 Byte mentre in Ext2/3 occupano al minimo la dimensione dell'i-node (da 1024 a 4096 byte).

Stimando un numero di directory di almeno 1.000.000 lo scenario con i due tipi di filesystem sarebbe stato:

- $10^6 \cdot 4096$ byte = $4096 \cdot 10^9$ Byte ~= 4 Gbyte *con EXT3*
- $10^6 \cdot 300$ byte ~= 300 Mbyte *con ReiserFS*

Inoltre sorgeva la necessita di effettuare una suddivisione delle directory su almeno due livelli, con più di 1000 directory su ogni livello in modo da non appesantire ciascuna

directory con un numero di elementi troppo elevato. Con l'uso di ReiserFS (e anche XFS) la struttura delle directory ha i tempi di accesso tipici di un albero binario, quindi molto minori dei tempi d'accesso agli elementi delle directory con algoritmo di ricerca sequenziale utilizzato dai filesystem di tipo Ext2/3 (si veda ad esempio l'articolo [JFL]).

Riassunto delle specifiche per ReiserFS:

versione	3.6
numero massimo di file	2^{32}
numero massimo di file per directory	2^{31}
numero massimo di sotto directory in una directory	$(2^{16}-1) \cdot 1024$
dimensione massima di un file	2^{60}-1 byte → 1 Ebyte
numero massimo di link ad un file	2^{32}
massima dimensione del filesystem	2^{32} (4096) blocchi => 17.6 Tbyte

Infine per poter garantire l'espandibilità delle dimensioni del filesystem in caso di necessità, si è scelto di utilizzare EVMS - Enterprise Volume Management System - [EVMS] per la gestione delle varie partizioni (volumi in EVMS).

2.7 Il client di acquisizione

Il client si occupa di acquisire i documenti direttamente dallo scanner e, dopo averli trasformati nel formato indicato (PNG), vengono inviati al server. Le scelta del Python e delle librerie collegate è stata effettuata sia per la rapidità di sviluppo che per coerenza con il resto dell'applicativo. Da notare come le fasi di trasferimento dati siano state effettuate ancora via HTTP/HTTPS, rendendo di fatto uniforme al resto della piattaforma il protocollo di trasmissione.

3 ArchEle

3.1 Funzionalità

ArchEle (ARCHiviazione ELEttronica) nasce come un prodotto (modulo) per Zope, installabile laddove sia possibile installare Plone >= 2.0 e Archetypes >= 1.3.

ArchEle estende le funzionalità di Archetypes e permette la gestione di grosse quantità di documenti da archiviare (anche per centinaia di Gigabyte), oltre alla scalabilità necessaria all'inserimento dei dati da parte di molti operatori, anche in maniera concorrenziale.

Funzionalità di ArchEle:

- permette la definizione dello schema (o tipo) dei documenti da acquisire con le relativi proprietà (Widget, validazione, tipo dello storage, etc.), producendo le interfacce di visualizzazione, modifica e ricerca ed i metodi di creazione, in modo automatico;
- memorizza sul server SQL i campi dei documenti; nessun dato viene scritto sullo ZODB che quindi viene usato solo per contenere alcune parti dell'applicativo;
- memorizza su filesystem i file (o le immagini) associati ai documenti;
- evita il tipico problema del ReadConflictError [MVCC] che si verifica nello ZODB per grossi carichi, tramite l'utilizzo dei cookie, lo storage non ZODB ed altre accorgimenti tecnici;
- riceve dal client appositamente sviluppato (denominato ISAAC) le immagini acquisite da più postazioni dotate di scanner e le memorizza sul filesystem predisposto sul server.

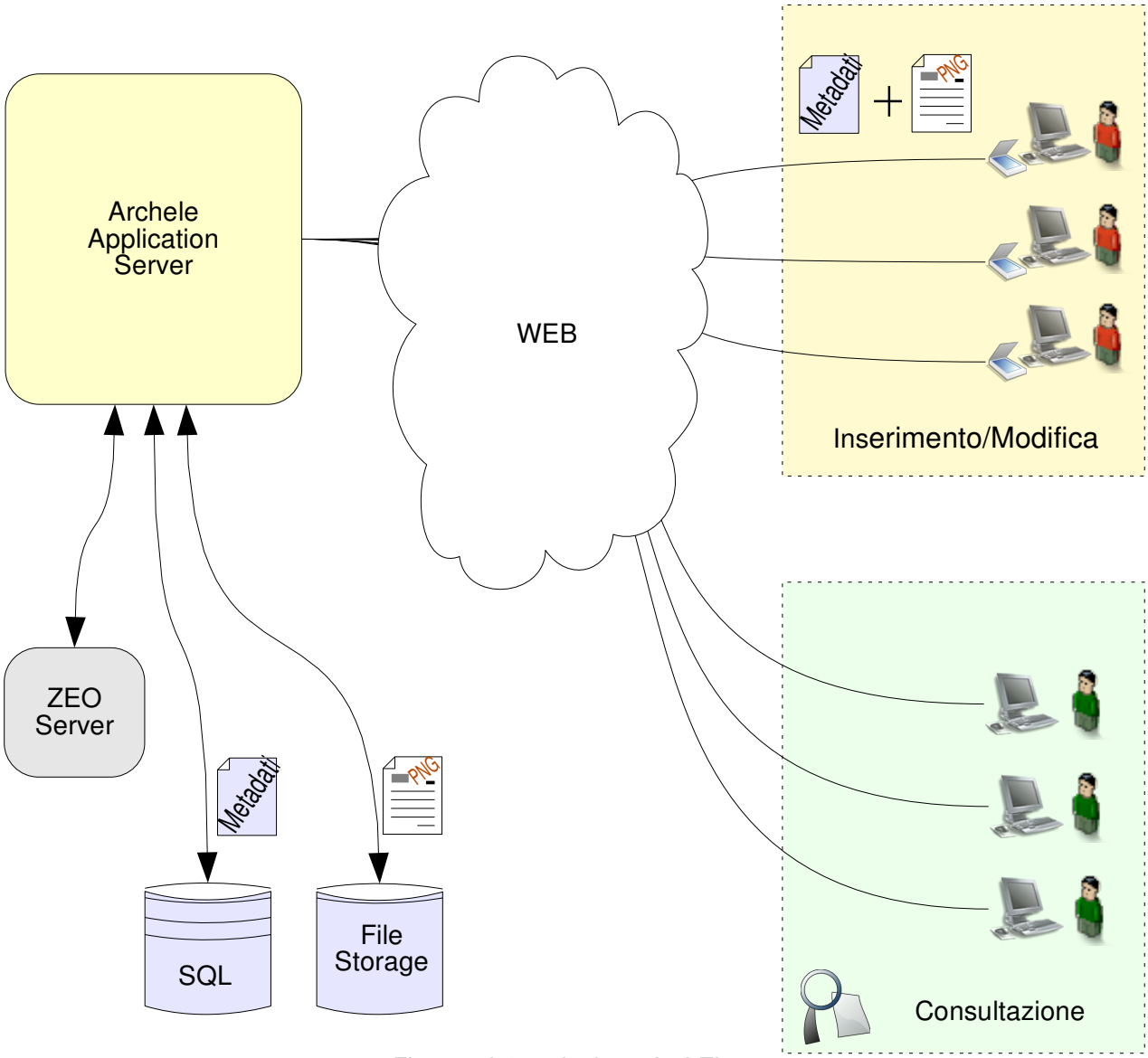

Figura 4-Interazioni con ArchEle

3.2 Limiti da superare

L'utilizzo per milioni di documenti dell'applicazione descritta ha evidenziato dei limiti nell'archiviazione degli oggetti all'interno dello ZODB. Nonostante siano state provate soluzioni che permettono di diminuire il carico di Zope, per esempio memorizzando le immagini su filesystem con ExtFile [EXTFILE], si sono riscontrati problemi di prestazioni dovuti all'inserimento di milioni di oggetti Archetypes. Si sono effettuato una serie di rilevamenti su vari indicatori, da cui si sono tratte delle analisi, alcune delle quali sono significative:

- per l'inserimento di ciascun oggetto vengono effettuate una serie di transazioni, che in totale fanno incrementare la dimensione dello ZODB di circa 160 KB; in queste transazioni sono presenti i nuovi oggetti Archetypes, le modifica della BTreeFolder che lo contiene, gli ExtFile delle immagini e le aggiunte agli indici (*portal_catalog*);
- con questo modello lo ZODB cresce anche di 250 MB al giorno, il che rende necessario

un “pack”[3] giornaliero;

- gli oggetti Archetypes – anche con pochi field – hanno una dimensione di circa 6 KB dopo il “pack”, dei quali soltanto una piccola parte (mediamente circa 300 Byte);
- il server era sottoposto ad un carico notevole dovuto all'elaborazione dell'elevato numero di oggetti coinvolti in ciascuna operazione; ciò imponeva delle lunghe attese durante il normale lavoro dei tre operatori: il load-average del server era sempre elevato (~3);
- i tempi di elaborazione dipendono in modo molto pronunciato dall'aumento del numero di oggetti contenuti e catalogati.

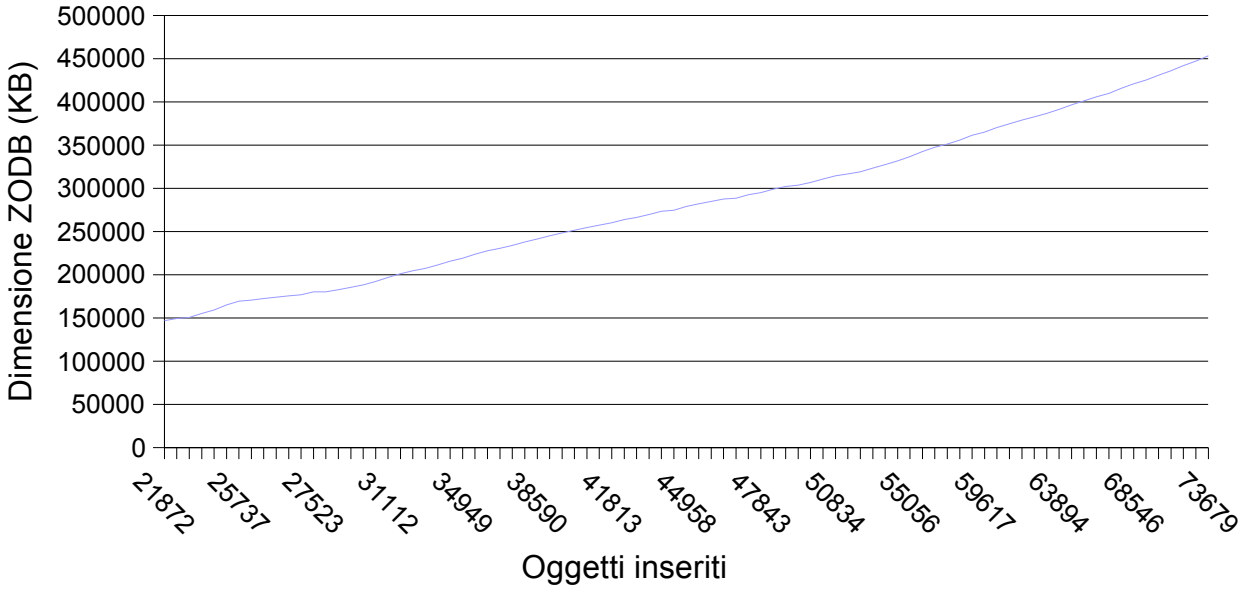

Figura 5-ZODB vs. Oggetti

I due grafici di Figura 5 e Figura 6 riportano i dati rilevati durante la fase di analisi e di primo sviluppo, nonché l'evoluzione con le modifiche riportate nei paragrafi seguenti.

3 Il pack è l'operazione con la quale si eliminano le vecchie transazioni, ormai superflue, per lo stato attuale dello ZODB.

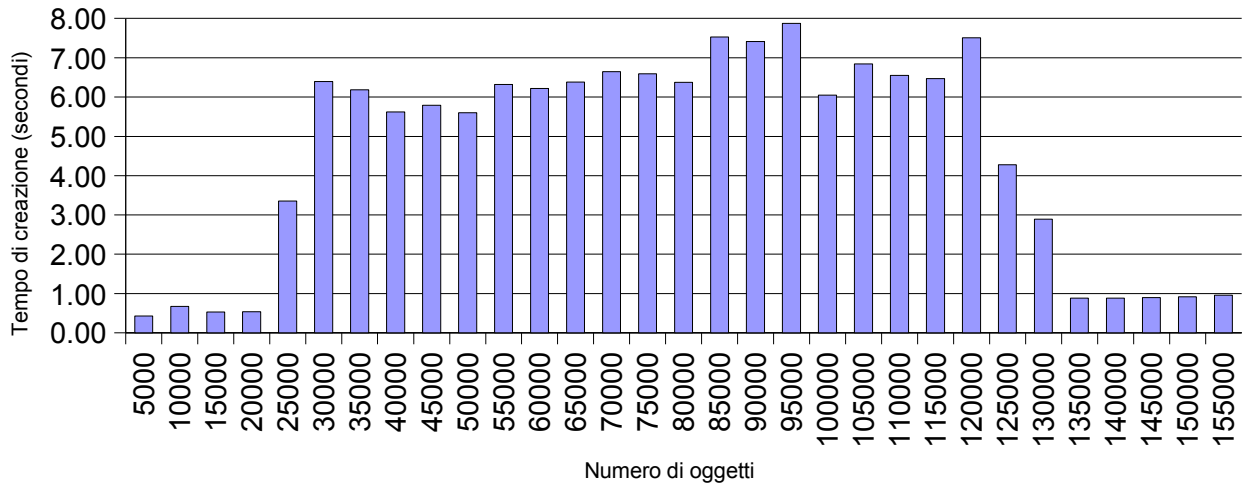

Figura 6-Tempo di creazione in funzione del numero di oggetti.

3.3 Ottenere la scalabilità richiesta

Per superare i limiti sopra esposti e mantenere il codice compatibile con Archetypes – in modo da poterne seguire gli sviluppi futuri – si è realizzato un nuovo storage che permette di archiviare milioni di documenti senza appoggiarsi sullo ZODB.

Tutti i field con storage SQL dipendono dallo UID (un metodo dell'oggetto). In pratica lo UID è la chiave di una tabella contenente i valori dei vari field con storage SQL. Si è implementata una classe base in ArchEle che ridefinisce il metodo UID, in modo da farlo dipendere direttamente dalla sessione (cookie). Si è anche realizzato un nuovo storage partendo dal PostgreSQLStorage, attraverso il quale i valori che vengono letti e/o scritti nell'istanza corrente saranno relativi all'UID specificato nella sessione.

Il risultato ottenuto è che con poco codice si riesce a utilizzare la logica di Archetypes, senza però istanziare effettivamente, per ciascun nuovo documento, alcun oggetto nello ZODB. Ciò ovviamente rende il sistema più veloce, ma altrettanto ovviamente si perde la relazione biunivoca tra oggetti nello ZODB e documenti acquisiti: nel database di Zope c'è un solo oggetto Archetypes per ogni tipo di documento (**collezione**) ed è sfruttata la possibilità di impostare nella sessione lo UID per selezionare uno degli elementi della collezione. In una logica che prende spunto dal FlyWeight pattern [PATTERNS] esistono uno o più FactoryObject per ogni collezione, che contengono al proprio interno la relazione di selezione degli elementi: lo UID in sessione [ARCHELE].

L'immagazzinamento dei file (o immagini) è ancora utilizzabile in una architettura composta da ZEO e più Zope (ZEO Client), fornendo un accesso condiviso allo stesso filesystem (noi usiamo NFS [NFS] [RFC1813]).

3.3.1 ArcheleBaseType

La classe ArcheleBaseType è la classe base da estendere per scrivere tipi che supportino le caratteristiche di base del prodotto ArchEle. La classe ArcheleBaseType estende la classe di Archetypes BaseFolderMixin in modo da avere a disposizione i metodi delle normali folder.

Le istanze degli oggetti ArcheleBaseType contengono un oggetto di tipo ExtFolder, istanziato al momento della creazione dell'oggetto ArcheleBaseType. L'oggetto di tipo ExtFolder contiene tutti i file e le immagini associate all'istanza corrente.

L'interfaccia della classe è la seguente:

```
class IArcheleBaseType(Interface):
    """ Interface for ArcheleBaseType object """

    def __init__(*args, **kw):
        """ Initialize the Class """
    def Title():
        """ Return the title of the instance """
    def unset(instance, **kwargs):
        """ Unset the value for this field using the underlying storage """
    def getSubObject(name, REQUEST, RESPONSE):
        """ getSubObject return None to minimize conflict error """
    def currentUser():
        """ Return the current user numericID """
    def clearSession():
        """ Clear session """
    def manage_beforeDelete(item, container):
        """ Called before Delete an object """
    def deleteInstance(uid):
        """ Delete current instance """
    def createInstance(uid, temporary=False):
        """ Create a new instance, temporary if temporary is true, with the
            given uid """
    def isMaster():
        """ Return true if the current object is master """
    def assertIsInstance():
        """ check if master and in case redirect to searchForm """
    def setUIDTemporary(uid):
        """ set UID temporary """
    def UID():
        """ Search the UID into request and cookies then set it
        in a cookie. If UID is not present generate it and set it in a cookie.
        Return the UID """
    def masterUID():
        """ Return the master uid """
    def manage_addObjects(files, REQUEST=None):
        """ add objects to the current object """
    def manage_addObject(file, REQUEST=None):
        """ add an object to the current object"""
    def reindexObject(*args, **kwargs):
        """ fake-reindex -> The objects are not indexed """
    def dummySequence(seq, length):
        """ used for sql-batch """
```

Le istanze degli oggetti di tipo ArcheleBaseType non vengono memorizzate all'interno dello ZODB come avviene normalmente in Archetypes, ma vengono memorizzate tramite l'SQL e il filesystem. Per conoscere quale istanza si vuole prelevare non si può quindi utilizzare il path dell'oggetto nello ZODB (visto nello ZODB c'è un solo oggetto FactoryObject per tipo), ma si è scelto di utilizzare il meccanismo dei cookie per tenere traccia dell'istanza su cui si vuole lavorare.

La classe ArcheleBaseType definisce anche un proprio “Schema” di base, l'ArcheleBaseSchema:

```
ArcheleBaseSchema = Schema((
    StringField('id',
                required=0,
                mode="rw",
                accessor="getId",
              mutator="setId", default=None,
                widget=ComputedWidget(label_msgid="label_name",
                                      visible={'search': 'invisible', 'view':
'invisible'},
                                      description_msgid="help_name",
                                      i18n_domain="plone")),
    IntegerField('creator',
                 searchable=0,
                 storage=SQLStorageOnly(),
                 required=0,
                 default_method='currentUser',
                 widget=StringWidget(label='creator',
                                     visible={'edit': 'invisible',
                                              'view': 'invisible',
                                              'search': 'invisible'},
                                     description='Creatore')),
     DateTimeField('creationTime',
                   searchable=0,
                   storage=SQLStorageOnly(),
                   required=0, default_method='now',
                   widget=StringWidget(label='Creation Time',
                                       visible={'edit': 'invisible',
                                                'view': 'visible',
                                                'search': 'invisible'},
                                       description='Data di creazione')),
    ), marshall=RFC822Marshaller())
```

L'ArcheleBaseSchema tiene conto delle informazioni comuni, come l'identificatore, il creatore e la data di creazione dell'istanza, informazioni normalmente presenti nello ZODB per ogni oggetto Persistent, ma che nel nostro caso era necessario memorizzare diversamente.

Già nell'ArcheleBaseSchema si fa uso di uno “Storage” appositamente scritto per far si che i dati siano immagazzinati solo nel DB, l'SLQStorageOnly che descriveremo nel prossimo paragrafo.

3.3.2 SQLStorageOnly

Visto che Archetypes, adattando lo storage SQL, inserisce e cancella le righe SQL solamente quando si crea e si elimina un oggetto (con modifica dallo ZODB), si è deciso di modificare questo comportamento introducendo un nuovo Storage, l'SQLStorageOnly.

L'SQLStorageOnly estende il PostgreSQLStorage di Archetypes modificandone alcune caratteristiche; in modo particolare si identifica ogni istanza per mezzo di un riferimento restituito dal metodo UID della classe ArcheleBaseType.

3.3.3 ExtFolder

Scopo principale di questo modulo è di gestire l'immagazzinamento di file ed immagini direttamente sul filesystem, tenendole legate agli oggetti dei tipi ArcheleBaseType.

L'ExtFolder si occupa inoltre di effettuare una suddivisione dei file sul filesystem su più livelli di directory in base all'UID o a un metodo configurabile per ciascun tipo ArchEle.

Le ExtFolder così create garantiscono la scalabilità anche nel caso di milioni di oggetti inseriti, oltre a fornire tutti i metodi tipici delle Folder di Zope. Per aggiungere alcune caratteristiche specifiche delle immagini (come determinare il content-type) si è implementato la classe TemporaryFSObject.

Di seguito sono riportate le interfacce delle due classi:

```
class ITemporaryFSObject(Interface):
    """ Interface for TemporaryFSObject """

    #TemporartFSObject extend OFS.Image.Image

    def __init__(filepath):
            """ initialize the Object by its filesystem path, the width, height, size,
        mimetype is fecthed by the file on the filesystem """
    def title_or_id():
        """ Return the title or the id of the object """
    def absolute_url():
        """ Return the absolute url of the object """
    def get_size():
        """ Return the size of the object in Byte """
    def getId():
        """ Return the Id of the object"""
    def ModificationDate():
        """ Return the mtime """
    def getData():
        """ Return the data """
    def index_html(REQUEST=None, RESPONSE=None, icon=0):
        """ Set the right content type in the request, convert the image in png
        if necessary and then return the data.
        If icon is True return the icon of the object. """


class ExtFolder(Interface):
    """ExtFolder
    store files and image on Filesystem
    """

    #ExtFolder extend OFS.SimpleItem.SimpleItem, OFS.CopySupport.CopyContainer

    def __init__(id, basePath=None, method='UID'):
        """ Initialize the instance.
        The method parameter is used to decide where to store the object. """
    def setBasePath(basePath):
        """ Set base path """
    def setIdMethod(method):
        """ Set id-method """
    def _genId(content_type):
        """ Generate serial id from content-type """
    def _path():
```

```
    """ Return the filesystem Path of the object """
def manage_addObject(file):
    """ Add object """
def manage_renameObject(old, new):
    """ manage rename Object """
def manage_delObjects(ids):
    """ manage_delObjects """
def _getOb(key, default=_marker):
    """ get object (temp). Return the TemporaryFSObject """
def objectValues(spec=None):
    """ Return object values like in Zope folder """
def objectIds(spec=None):
    """ Return Object IDS like in Zope folder """
def objectItems(spec=None):
    """ Return object-items like in Zope folder """
def index_html(REQUEST=None, RESPONSE=None):
    """ index_html. Return ''. """
def __getitem__(key):
    """ get-item. Return the object"""
def __getattr__(key):
    """ getattr """
```

3.4 Trasformazioni aggiuntive

Con Plone è fornito un prodotto, il PortalTrasform, che permette di effettuare trasformazioni di formato da un tipo di dato ad un altro. Esempi di tipiche trasformazioni sono html_to_text, python_to_text, word_to_html.

Per alcune nostre necessità era utile memorizzare immagini in formato TIFF per poi visualizzarle tramite il browser; per questo scopo si è realizzato un convertitore, che grazie al PortalTransforms, è stato utilizzato per trasformare le immagini da TIFF a PNG.

Il convertitore fa uso per la conversione del programma tiff2png [TIFF2PNG].

3.5 Skin

Per questo prodotto sono stati creati una serie di ZPT e di script python basati in larga misura su quelli di Archetypes; le modifiche e le aggiunte permettono l'integrazione con le classi realizzate: interfacce a ArcheleBaseType, ExtFolder e il resto della Business Logic.

3.5.1 Creazione degli oggetti

Per mantenere la coerenza con gli altri oggetti del Plone e non istanziare nuovi oggetti ArchEle si è personalizzato il metodo di creazione (createObject.cpy). Per gli oggetti di tipo ArchEle invece della normale codice di creazione:

- si genera un identificatore univoco (UID) per mezzo di una sequenza PostgreSQL che gestisce per noi la concorrenza ed univocità nella generazione di id;
- si crea l'istanza SQL del documento dal FactoryObject (tramite il metodo createInstance).

3.5.2 Upload delle immagini

Lo script “uploadFiles” inserito negli skin si occupa di prelevare i file passati dal client tramite una post HTTP/HTTPS e di inserirli all'interno dell'oggetto corrente.

L'istanza in cui inserire i file viene determinata dallo script tramite il cookie che il client spedisce insieme ai file.

3.5.3 Lancio dell'acquisizione

Per far si che il client di ArchEle (Isaac) sia attivato il server spedisce il file “acquire.isaac” al client con content-type “application/x-isaax”; dentro questo file vengono inserite le informazioni necessarie per l'autenticazione dell'utente (tramite il cookie associato) e le informazioni relative al documento a cui aggiungere le immagini acquisiste da scanner, cioè il cookie con l'UID dell'oggetto in cui inserire le immagini.

3.6 Il client: Isaac

Per acquisire i documenti direttamente dallo scanner e inviarli al server si è realizzato un piccolo software in python che svolgesse questi compiti. Ad oggi è stata sviluppata una versione del client per Windows, ma grazie al progetto SANE (PIL-SANE) è molto semplice implementarne una versione per Linux.
Per svilupparne la parte grafica si è adottato il framework wxPython [WXPYTHON] che permette la creazione di interfacce grafiche portabili su più piattaforme. L'ambiente wxWindows su cui il wxPython si basa, viene distribuito per Linux (in ambiente GTK+), per Windows ed anche per altre piattaforme.

Per l'acquisizione delle immagini dallo scanner si è utilizzato il TWAIN Module per python [TWAINPY] che permette di interfacciarsi in ambiente Windows con praticamente tutti gli scanner in commercio. Il codice necessario a tale funzionalità è ridotto a poche righe che espletano il compito di passare i dati dal driver Twain al programma stesso. In questa fase si è anche tenuto conto che l'applicazione avrebbe dovuto acquisire più immagini (FAX o altri documenti cartacei) per mezzo di scanner con alimentazione a modulo continuo.

L'acquisizione delle immagini tramite lo scanner passa dalle seguenti fasi (come illustrato nella Figura 7):

1. L'operatore richiede l'acquisizione tramite le azioni “Salva” e “Salva e Acquisisci”.
2. L'applicativo fornisce una risposta HTTP con *Content-Type* ad hoc (i.e. *application/x-isaac*), che contiene le credenziali dell'utente corrente, la versione minima del client necessaria e l'URI a cui dovranno essere inviate le immagini acquisite.
3. Grazie all'aggiunta di una voce al registro di Windows [WINREG], effettuata al momento dell'installazione del client e che associa al suddetto content-type il client di acquisizione, il browser avvia automaticamente detto client fornendogli un file temporaneo contenente la risposta di ArchEle.
4. Il client richiede l'acquisizione delle immagini allo scanner (notare che gli scanner nel nostro caso sono con Automatic Document Feeder).
5. Il client via Twain acquisisce in modo asincrono le immagini dallo scanner, disponibili in formato BMP.
6. Il client converte le immagini da BMP a PNG.
7. Il client invia le immagini PNG all'URI fornito dall'application server. Insieme alle immagini invia anche le credenziali che gli sono state passate inizialmente.

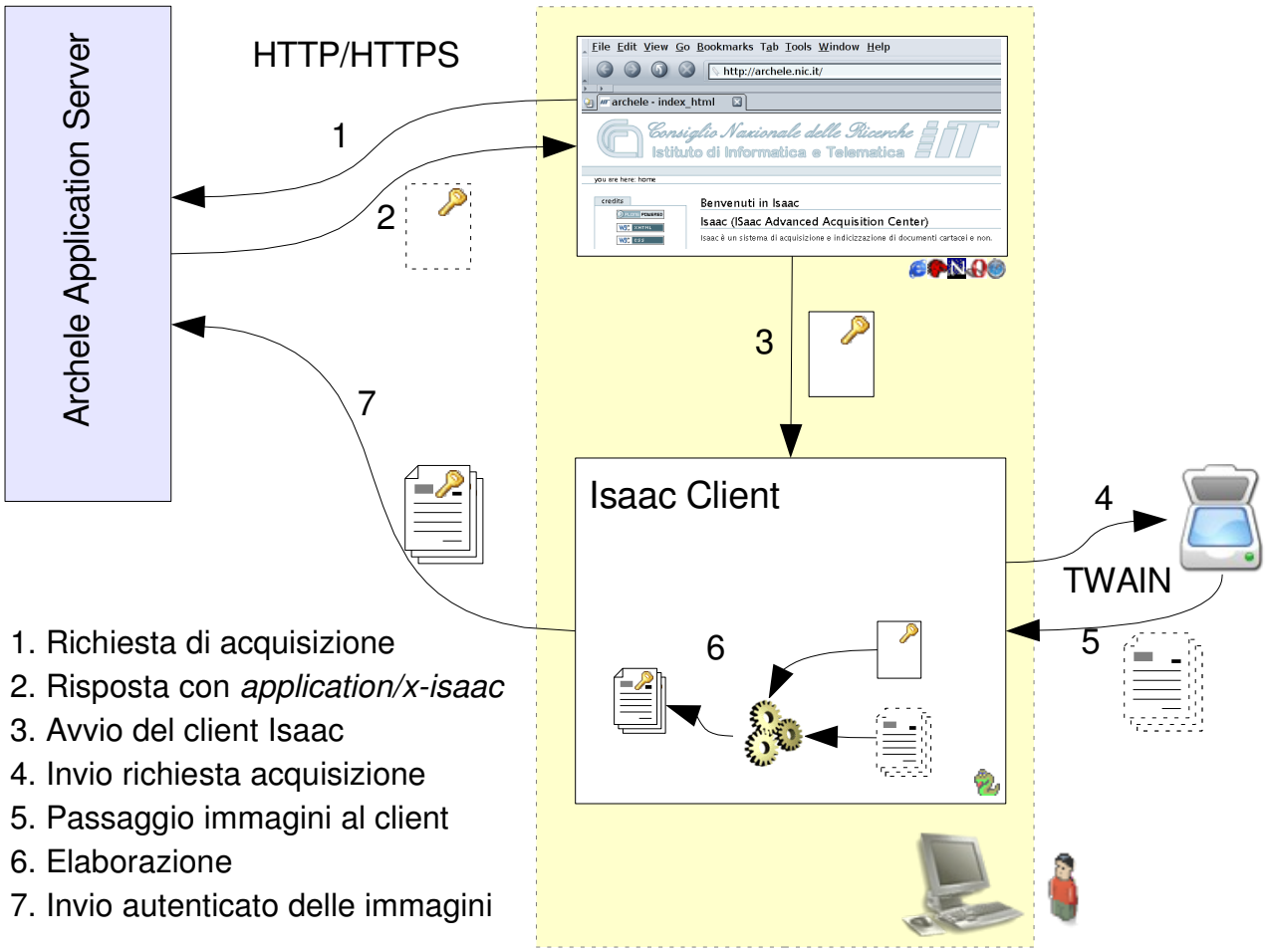

Figura 7-Funzionamento Isaac client

La conversione da BMP a PNG viene effettuata dal client usando la libreria Python [PIL]. Durante lo sviluppo del client si è notato che la fase di conversione non era demandabile al server, se non introducendo su quest'ultimo un carico di lavoro molto elevato che faceva rallentare le altre fasi di elaborazione. L'uso di questa conversione direttamente sul client permette una maggiore scalabilità in funzione del numero di client e di operazioni di acquisizione.

Tutti i dati e le immagini relative all'acquisizione e alle fasi successive vengono elaborati e mantenuti in memoria, evitando l'uso di file temporanei e del filesystem del client in genere. Questi fatti unito al fatto che l'autenticazione del software di acquisizione è basata sui cookie ricevuti dal browser al momento dell'acquisizione e passati su un file temporaneo al programma stesso, riduce al minimo le scritture su disco di dati sensibili – potenzialmente legati a dati personali.

Per eventuali porting su piattaforma Linux è valutabile la realizzazione dell'interfacciamento via SANE (standard sotto Linux) per l'acquisizione di immagini.

Infine si è prodotto con py2exe [PY2EXE] un eseguibile di circa 3 MegaByte che racchiude il codice e le librerie necessarie al funzionamento del client, in modo da ridurre i tempi di avvio del client, che è lanciato centinaia di volte in un giorno su ciascuna postazione, e minimizzare i tempi di installazione.

3.7 Case Study

L'applicazione archivia i documenti cartacei relativi a vari tipo di documenti gestiti dal Registro del ccTLD “.it”.

Sono stati archiviati tramite l'applicazione:

- 513.500 Cambi Maintainer, con circa 505.000 immagini associate (alcuni Cambi hanno riferimenti alle stesse immagini), per un totale di 37 GB di immagini;
- 213.500 Lettere Assunzione Responsabilità con circa 503.000 immagini associate per un totale di 44 GB di immagini (trasportati dal vecchio applicativo);
- 105.000 Cambi Maintainer e Registrante, con circa 105.000 immagini associate, per un totale di 7 GB di immagini;
- 118.500 No Provider, con circa 118.500 immagini associate, per un totale di 13GB di immagini.

L'applicazione funziona efficientemente su un server HP con doppio Processore XEON a 3.4 GHz, 4GB RAM, 2 dischi SCSI da 72GB in RAID-1 e un box di dischi esterno da 260 GB.

I software utilizzati sul server sono: Debian GNU/Linux con kernel 2.6, Zope 2.7, Plone 2.0, Archetypes 1.3, Postgresql 8.1, FileSystem ReiserFS v3.6 e XFS.

L'applicazione è utilizzata contemporaneamente da 3 operatori dedicati che hanno il compito di acquisire i dati cartacei e dagli operatori del registro che consultano i dati presenti nell'applicazione.

Per aumentare la velocità con cui vengono inseriti i metadati associati ai documenti ed avere un maggior controllo sugli stessi, ArchEle è stata integrata con il database dei domini del Registro. La problematica dell'acquisizione dei documenti cartacei precedentemente era stata affrontata dallo IIT attraverso una applicazione proprietaria standalone per Windows, con la quale erano stati acquisiti mediamente circa 1775 documenti al mese per ogni inseritore. Con ArchEle, lo stesso hardware e nessun costo per le licenze, i dati sono decisamente migliorati: a un anno dall'avvio dello sviluppo erano stati acquisiti circa 280.000 documenti, con una media di circa 7700 documenti al mese per ogni inseritore (+330% di produttività).

3.7.1 I Cambi MNT e le LAR

Per il Registro sono stati scritti due tipi di documenti:

- i cambi maintainer (cambi MNT);
- le lettere assunzione responsabilità (LAR).

Figura 8-ArchEle: scelta della tipologia di documenti su cui operare.

Alcuni schemi nell'applicazione sono i seguenti:

- **LAR:**

```
schema = ArcheleBaseSchema + Schema((
    StringField('dominioLAR',
                searchable=0,
                storage=SQLStorageOnly(),
                required=1,
                widget=DomainWidget(label='Dominio',
                                    description='Inserisci un valore per il dominio',
                                    matchedMethod='matchedDomainsLAR',
                                    countMatchedMethod='matchedDomainsLARCount',
                                    )
                ),
    DateTimeField('dataLAR',
                  searchable=0,
                  required=1,
                  storage=SQLStorageOnly(),
                       validators=('isValidDate', ),
                       widget=CalendarWidget(label='Data della LAR',
                                             description='Inserisci una data per la LAR')
                  ),
    StringField('providerLAR',
                searchable=0,
                storage=SQLStorageOnly(),
                required=1,
                widget=StringWidget(label='Provider / Maintainer',
                                    description='Inserisci un provider/maintainer')
```

```
                ),
    IntegerField('numeroLAR',
                searchable=0,
                storage=SQLStorageOnly(),
                required=1,
                validators=('isInt', ),
                widget=StringWidget(label='Numero LAR',
                                    description='Inserisci un numero per la LAR')
                ),
     StringField('ticket',
                 searchable=0,
                 storage=SQLStorageOnly(),
                 required=0,
                 widget=StringWidget(label='ticket',
                                     visible={'edit': 'hidden',
                                              'view': 'invisible',
                                              'search': 'invisible'},
                                     description='ticket di Endy'))
    ), marshall=RFC822Marshaller())
```

- **CambiMNT:**

```
schema = ArcheleBaseSchema + Schema((
     StringField('dominioCambio',
                 searchable=0,
                 storage=SQLStorageOnly(),
                 required=1,
                 widget=DomainWidget(label='Dominio',
                                     description='Inserisci un valore per il dominio',
                                     matchedMethod='matchedDomainsCambi',
                                     countMatchedMethod='matchedDomainsCambiCount',
                                     )
                 ),
    StringField('numeroCambio',
                searchable=0,
                storage=SQLStorageOnly(),
                required=1,
                validators=('isInt', ),
                widget=StringWidget(label='Numero Cambio',
                                    description='Inserisci un numero per il cambio')
                ),
    DateTimeField('dataCambio',
                  searchable=0,
                  storage=SQLStorageOnly(),
                  required=1,
                       validators=('isISODate', ),
                       widget=CalendarWidget(label='Data del cambio',
                                             description='Inserisci una data per il cambio nel
formato ISO (anno/mese/giorno)')
                  ),
    StringField('providerVecchio',
                searchable=0,
                storage=SQLStorageOnly(),
                required=1,
                widget=StringWidget(label='Provider / Maintainer Vecchio',
```

```
                                    description='Inserisci un provider/maintainer')
                ),
    StringField('providerNuovo',
                searchable=0,
                storage=SQLStorageOnly(),
                required=1,
                widget=StringWidget(label='Provider / Maintainer Nuovo',
                                    description='Inserisci un provider/maintainer')
                ),
    StringField('ticket',
                 searchable=0,
                 storage=SQLStorageOnly(),
                 required=0,
                 widget=StringWidget(label='ticket',
                                     visible={'edit': 'hidden',
                                              'view': 'invisible',
                                              'search': 'invisible'},
                                     description='ticket di Endy')
                 ),
    IntegerField('cambioMaster',
                  schemata='Cambi Multipli',
                  storage=SQLStorageOnly(),
                  searchable=0,
                  required=0,
                  widget=StringWidget(visible={'edit': 'hidden',
                                               'view': 'invisible',
                                               'search': 'invisible'})
                  ),
    StringField('dominiMultipli',
                schemata='Cambi Multipli',
                searchable=0,
                required=0,
                accessor='getDominiMultipli',
                mutator='insertDominiMultipli',
                widget=DominiMultipliWidget(label='Domini Multipli',
                                            description='Verifica i nomi dei domini da
inserire',
                                            visible={'edit': 'visible',
                                                     'view': 'invisible',
                                                     'search': 'invisible'})
                ),

     ),
     marshall=RFC822Marshaller())
```

Come si vede dal codice lo schema è l'estensione dell'ArcheleBaseSchema che tiene traccia delle “informazioni base”, ogni ulteriore field che deve essere memorizzato ha come Storage “SQLStorageOnly”.

Dallo schema del cambioMNT si può anche vedere che viene utilizzato un widget appositamente creato, il DomainWidget il cui funzionamento viene illustrato nei prossimi paragrafi. Questo widget ha permesso l'integrazione dei dati dei domini e dei cambi della del Registro con quelli di questa applicazione.

3.8 Integrazione con i dati del Registro

Grazie alla collaborazione con gli sviluppatori delle applicazioni del Registro è stato possibile mettere a disposizione i dati in possesso del Registro relativi ai Cambi Maintainer, ai Cambi Maintainer e Registrante ed alle LAR tramite ArchEle.

Questi dati inseriti nel database SQL di ArchEle, sono stati integrati nel processo di inserimento dei Cambi e delle LAR in modo da:

- diminuire gli errori di digitazione manuale dei dati;
- verificare eventuali incongruenze tra i documenti cartacei acquisiti ed i dati in possesso del Registro;
- velocizzare notevolmente l'inserimento dei documenti.

3.8.1 Interfacce per l'integrazione

Durante l'inserimento dei Cambi e LAR è possibile ricercare tramite il nome del dominio tutte le operazioni eseguite per i domini che corrispondono al criterio selezionato. All'utente viene presentato un pulsante cerca posizionato accanto alla casella per l'inserimento del dominio, tramite il quale è possibile fare ricerche sulle operazioni effettuate (Cambio o LAR).

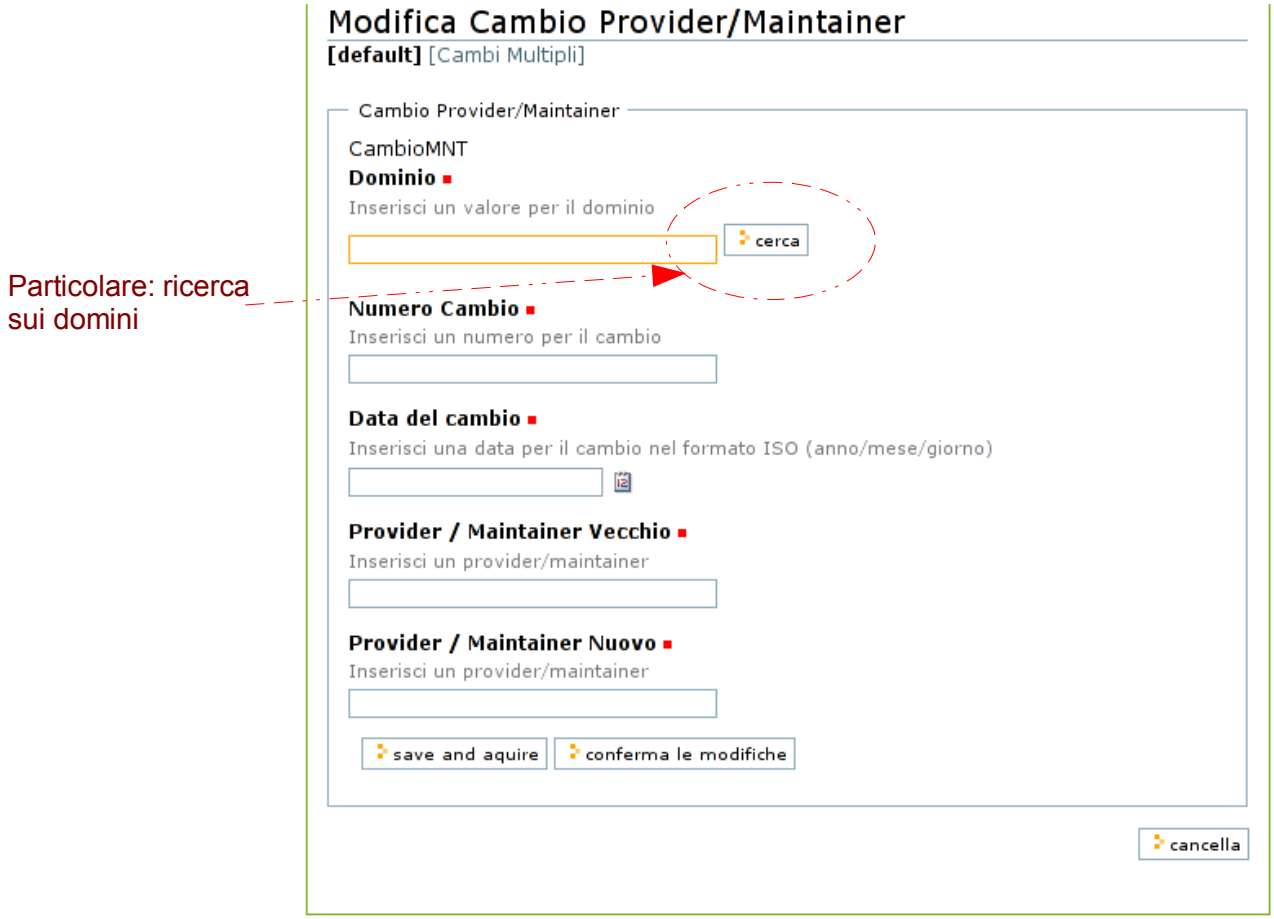

Figura 9-Interfaccia di modifica/inserimento di un Cambio Maintainer.

La ricerca mostra i Cambi (oppure LAR a seconda di cosa stiamo inserendo) relative ai domini che cominciano con le lettere inserite nel campo Dominio.

Nel caso i risultati trovati siano “pochi” - meno di 6 - viene mostrata una ***select*** che visualizza tutte le operazioni trovate che corrispondono a domini che cominciano con le lettere inserite.

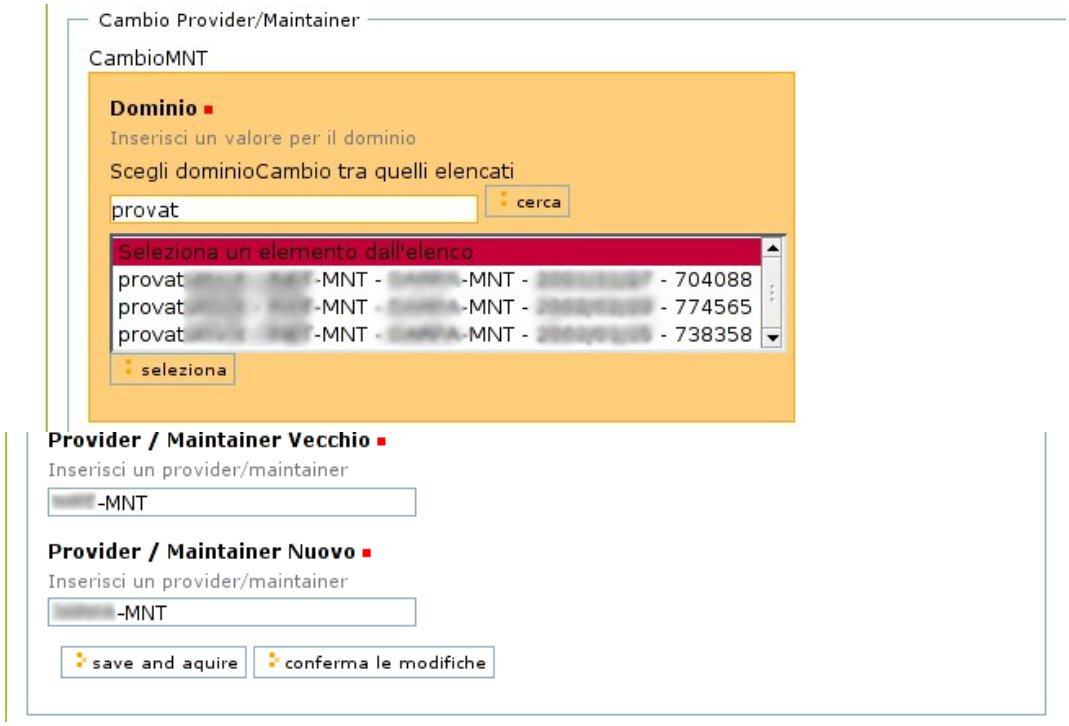

Figura 10-Dettaglio del riempimento campi da operazione

Scegliendo una tra le operazioni visualizzate, vengono assegnati ai valori relativi quelli dell'operazione selezionata.

Nel caso i risultati trovati siano “molti” - tra 6 e 30 - viene mostrata una finestra a tendina che mostra la lista delle operazione corrispondenti ai dominio cercato. Anche quest'ultima permette di selezionare un'operazione assegnando ai campi quelli dell'operazione selezionata.

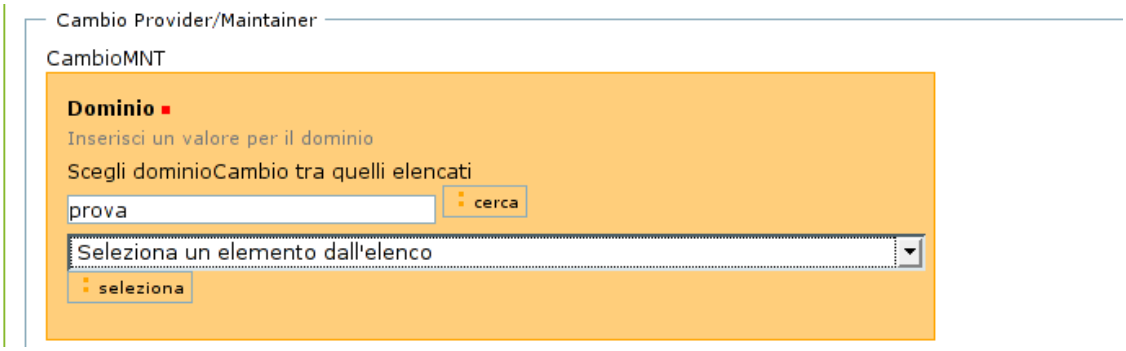

Figura 12-Dettaglio selezione multipla in modifica

Nel caso i risultati da visualizzare siano “troppi” - più di 30 - viene mostrato un messaggio che avverte l'utente indicando il numero di operazioni trovate; naturalmente è possibile raffinare la ricerca e quindi arrivare ai casi “pochi” o “molti”.

Se la ricerca non ha prodotto risultati utili si è avvisati che non ci sono operazioni corrispondenti a domini che cominciano con la stringa fornita.

3.8.2 Gli aggiornamenti dei dati

I dati relativi alle operazioni del Registro sono mantenuti nel database dell'applicazione che viene aggiornato automaticamente con scadenza mensile.

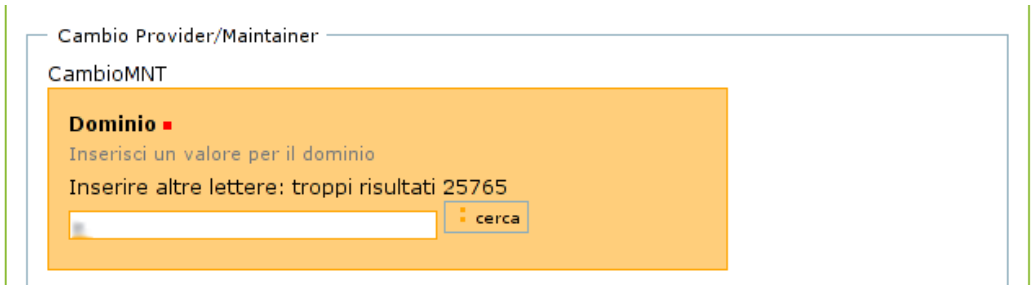

Figura 13-Dettaglio ricerca dominio in modifica Cambio Maintainer

3.8.3 I cambi Multipli

Per i fax contenenti richieste di Cambi Maintainer multipli (anche migliaia) per domini diversi è stata sviluppata una procedura *ad hoc*.

Poiché ogni cambio provider ha associato un unico dominio, la soluzione immediata di acquisire il documento cartaceo (fax) del cambio multiplo una volta per ogni dominio indicato nel fax stesso, è praticabile, ma prevede un grosso lavoro manuale da parte degli operatori che si occupano dell'inserimento.

Si è perciò realizzato un meccanismo che, selezionando un cambio provider (chiamato **cambio principale**), mostra tutti i cambi effettuati nella stessa data e con lo stesso provider (vecchio e nuovo); questi dati sono importati dal database del Registro.

Il passo successivo è la procedura di associazione delle operazioni, selezionati dalla lista precedentemente citata, al cambio principale.

In questo modo tutti i “cambi multipli” inseriti sono collegati al “cambio principale” e ne condividono le immagini associate, con un effettivo risparmio di tempo nell'inserimento e di spazio disco per le immagini. La modifica delle immagini associate ad uno qualunque di questi “cambi multipli”, sia condivisa con tutti gli altri che hanno lo stesso *cambio principale*, anche se i metadati associati (nome-dominio, ...) sono distinti.

Figura 14-Selezione domini nei cambi multipli

3.9 Verifiche

L'applicazione utilizzata per l'archiviazione elettronica dei documenti cartacei dal Registro conterrà milioni di documenti (al momento della scrittura di questo documento ne contiene già 950.000), quindi c'è stato bisogno di scrivere delle procedure di controllo per verificare la coerenza dei dati inseriti da parte degli inseritori.

I controlli sono stati studiati per i dati specifici del Registro quindi non fanno parte del “prodotto base”.

Per ogni mese inserito i controlli e verifiche che ogni operatori abilitato possono consultare sono:

- la lista dei documenti, di un determinato tipo, inseriti;
- il numero dei documenti inseriti con il riferimento all'ultimo per il mese selezionato;
- la lista dei Cambi/LAR che hanno lo stesso numero di cambio all'interno dello stesso mese (il numero dovrebbe essere univoco all'interno di un mese, escludendo i cambi multipli descritti in precedenza);
- la lista dei Cambi/LAR con uguali dominio, data del cambio, maintainer vecchio e nuovo.
- la lista dei numeri di cambio che risultano non utilizzati all'interno del mese;
- l'elenco di documenti senza immagini associate;
- l'elenco di documenti senza dati associati, ma con immagini associate.

3.10 Integrazione con gli altri dati del Registro

Nell'applicazione è stato importato l'archivio risultante dal lavoro di acquisizione delle LAR, avvenuta attraverso il software PEGASUS (proprietario), con il quale sono state immagazzinate oltre 200.000 LAR.

PEGASUS utilizza un'architettura Microsoft SQLServer + FileServer, per lo storage dei dati acquisiti con il client standalone realizzato in Delphy e funzionante sotto Windows. Dopo aver effettuato il reverse engineering della parte del software che si occupa dello storaging prelevando i dati dal DB MSSQL, si è scritto un piccolo modulo per prelevare dal filesystem i file associati ad ogni documento. Per mezzo di un software appositamente sviluppato si è proceduto alla trasformazione dei dati e al successivo inserimento in ArchEle.

Dall'integrazione dei dati vecchi e nuovi inseriti dal personale addetto, si ha la possibilità di usare un'unica applicazione per effettuare ricerche, aggiornamenti, inserimenti e quant'altro sia necessario.

4 La scalabilità dell'applicazione

L'applicazione, per il progetto del Registro, attualmente è in funzione su un server HP con doppio processore Intel Xeon da 3.4GHz, 4 GB di Ram ed uno storage esterno da 250GB. L'applicazione non ha problemi di velocità con 3 operatori che inseriscono circa 1500 nuovi documenti al giorno ed un numero variabile di operatori che consultano l'applicazione.

L'architettura dell'applicazione è scalabile anche per un numero più elevato di postazioni che effettuano inserimento/consultazioni, infatti l'architettura estremamente modulare permette di suddividere l'intero sistema in molti sottosistemi.

I sottosistemi coinvolti sono:

- Load Balancer / Proxy

  si occupa dello smistamento delle richieste degli utenti ai vari application server; inoltre mantiene una cache degli oggetti richiesti più frequentemente;

- ZEO Server

  fornisce utenti e gruppi del sistema, permessi e ruoli e struttura logica dei documenti;

- Business Logic

  è l'insieme degli application server che si occupano, una volta ricevute le richiesta dal Load Balancer/Proxy, di verificare utenti e ruoli dallo ZEO Server, di costruire le interfacce richieste, di effettuare le validazione, di restituire i dati richiesti, oltre che effettuare l'inserimento delle immagini sul filestorage system e sul DBMS system;

- FileStorage System

  si occupa di effettuare lo storage dei dati sul filesystem; gli storage possono essere vari e riuniti sotto unità logiche tramite EVMS [EVMS]. La trasmissione dei dati con gli application server della Business Logic avviene tramite NFS (Network File System)

- DBMS System

  è l'insieme dei DBMS che si occupano dello storing e retrieving dei metadati. Ogni tipo di documento utilizzato in ArchEle può avere il suo DBMS. I DBMS scambiano i dati con gli application server della Business Logic tramite protocollo TCP/IP (anche con layer sicuro - *SSL*).

L'architettura è descritta nella Figura 15-Scalabilità ArchEle che segue.

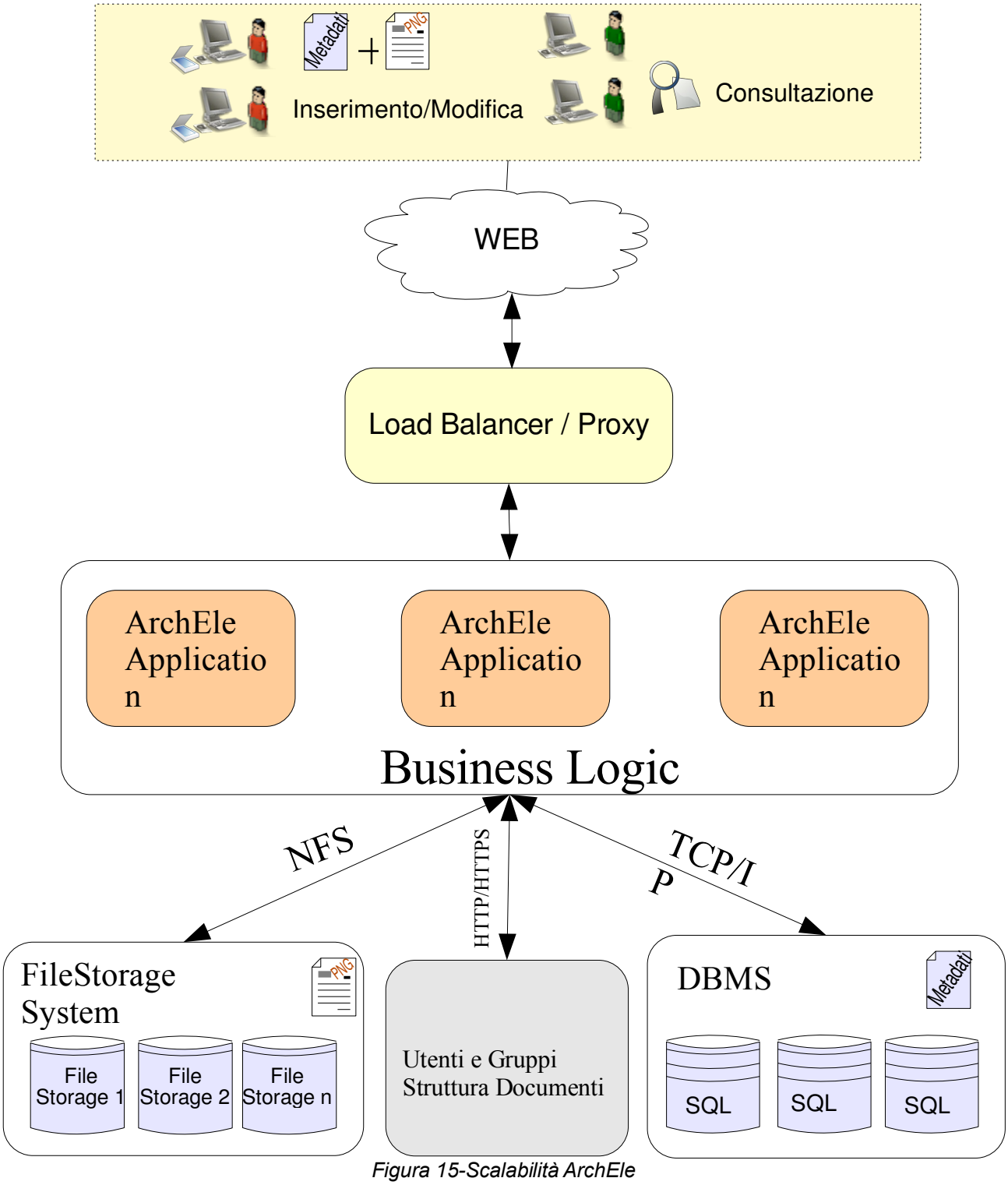

Figura 15-Scalabilità ArchEle

5 Sviluppi futuri

5.1 Personalizzazioni via Web e UML

Utilizzando alcuni prodotti aggiuntivi, attualmente in sviluppo, è possibile creare e manipolare gli Schema di Archetypes in modo semplice anche da parte di utenti non esperti. Questi prodotti permettono, tramite interfacce grafiche, la definizione degli Schema da utilizzare dentro ArchEle.

Esistono dei prodotti correlati ad Archetypes che permettono la creazione di Schema in modo non python:

- ArchGenXML

  Un software che genera moduli per Plone basati sul framework di Archetypes a partire da modelli UML che utilizzando file XMI (.xmi, .zargo, .zuml) e file XSD (XMLSchema).

- ArcheBuilder

  Un generatore di schema per Archetypes basato su XUL (il linguaggio XML utilizzato per sfruttare le componenti grafiche di Mozilla).

Inoltre esistono gestori di Schema via Web che all'occorrenza possono essere integrati con ArchEle per permettere la modifica degli Schema usando l'interfaccia di Plone.

5.2 Full Text Search

Il Plone mette a disposizione tramite il proprio portal_catalog l'indicizzazione del contenuto dei file associati ai vari documenti che sono memorizzati all'interno dell'applicazione stessa. I tipi di file indicizzabili sono:

- Portable Document Format
- Documenti MS Word
- Documenti Excel
- Documenti PowerPoint
- Documenti Openoffice
- Documenti HTML
- Documenti di Testo generico
- Documenti in Rich Text Format.

L'architettura per l'indicizzazione dei documenti è ad ogni modo estendibile.

Gli indici degli allegati indicizzati vengono inseriti nello ZODB.

Nella nostra applicazione l'indicizzazione dei file è stata disattivata poiché non era interessante per il nostro utilizzo e non abbiamo verificato la scalabilità su milioni di oggetti indicizzati.

Sfruttando il modulo interno di Zope ZCTextIndex è possibile creare un sistema di indicizzazione che estrae i contenuti dai documenti nei formati attualmente supportati da Plone e li immagazzina in uno storage. Per l'indicizzazione del contenuto dei documenti è anche possibile utilizzare il software del progetto OpenFTS (Open Source Full Text Search engine basato su PostgreSQL).

5.3 Workflow documentale

Il Plone comprende al suo interno un modulo per la gestione del flusso di lavoro – DCWorkflow – con il quale si possono definire:

- gli stati delle oggetti (*pubblicato, privato, da approvare, ...);*
- le transizioni di stato (*privato → pubblico*, ...);
- le azioni associate ad una transizione di stato (*spedire una email*, ...);
- le variabili di stato (*data di approvazione, responsabile*, ...).

Vista l'architettura di ArchEle si è preferito non utilizzare i Workflow sia per evitare complicazioni non richieste in questa applicazione, sia per limitare le problematiche relative all'integrazione del DCWorkflow con SQL. È anche ipotizzabile una integrazione con le specifiche (XML/Semantic Web) presenti e future.

Nel caso di utilizzo di ArchEle per la gestione del flusso di lavoro dei documenti è possibile prevedere l'integrazione delle tecnologie esposte.

5.4 Firma elettronica dei documenti

Nell'ambito della conversione da cartaceo a documento elettronico un passo importante è l'adozione della firma digitale che permetterebbe, con il sostegno della legislazione, di utilizzare anche per scopi ufficiali la copia elettronica.

Per far si che i documenti memorizzati sul server siano firmati digitalmente è possibile prevedere l'integrazione di un software per la firma digitale all'interno del client Isaac.

Questo è necessario per permettere di firmare i documenti acquisiti, in modo da certificare chi è l'autore dell'acquisizione.

Bibliografia

[XHTML] W3C, XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), W3C Recommendation, 2000-2002

[CSS2] W3C, Cascading Style Sheets, level 2, 1998

[PNG] W3C, PNG (Portable Network Graphics) Specification, 1996

[WXPYTHON] Home page wxPython: a blending of the wxWidgets C++ class library with the Python programming language, 2004, http://www.wxpython.org/

[TWAINPY] Python TWAIN Module, 2004, http://twainmodule.sourceforge.net/

[PIL] Python Imaging Library (PIL), 2004, http://www.pythonware.com/products/pil/

[DEBIAN] Home page Debian, , http://www.debian.org/

[ZOPE] Home Page Zope, , http://zope.org

[PLONE] Home page Plone, 2004, http://plone.org

[W3C-7P] W3C in 7 points, , http://www.w3.org/Consortium/Points/

[PYTHON] Python Home Page, , http://www.python.org/

[ZOPEBK] Zope Book 2.6 Edition, 2004, http://www.zope.org/Documentation/Books/ZopeBook/

[PLONEBK] Plone Book, 2003-2004, http://plone.org/documentation/book

[DC] Dublin Core, 2004, http://www.dublincore.org/

[ARCHDG] Sidnei da Silva, Archetypes Developers Guide,

[ADVZODB] Advanced ZODB for Python Programmers, 2003, http://zope.org/Documentation/Articles/ZODB2

[REISERFS] ReiserFS Home page, 2004, http://www.namesys.com/

[XFS] SGI, XFS Filesystem, 2004

[JFL] Journalling Filesystems for Linux, , http://www.linuxgazette.com/issue68/dellomodarme.html

[EVMS] IBM, Enterprise Volume Managament System (EVMS), 2004

[MVCC] Multi-version concurrency control for ZODB, 2004, http://zope.org/Wikis/ZODB/MultiVersionConcurrencyControl

[EXTFILE] ExtFile/ExtImage Product for Zope, , http://www.zope.org/Members/MacGregor/ExtFile

[PATTERNS] Eric Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software , 1995

[ARCHELE] M. Andreini, C. Lucchesi, M. Martinelli, G. Vasarelli, ArchEle: an acquisition search and retrival system based on Zope/Plone., 2004

[NFS] Network File System, 2004, http://nfs.sourceforge.net/

[RFC1813] Callaghan, Pawlowski, Staubach Sun Microsystems, Inc., RFC 1813 - NFS Version 3 Protocol Specification, 1995

[TIFF2PNG] Greg Roelofs, Willem van Schaik, Tiff2png onverts a Tagged Image File Format (TIFF) file into a Portable Network Graphics (PNG), 2004

[WINREG] Microsoft, Description of Microsoft Windows Registry - Articolo Microsoft Knowledge Base - 256986, 2004

[PY2EXE] Convert python scripts into standalone windows, 2004, http://starship.python.net/crew/theller/py2exe/

Indice delle illustrazioni

Figura 1-Livelli di ArchEle.....5
Figura 2-Zope.....7
Figura 3-ArchEle: interfaccia di amministrazione in Zope.....9
Figura 4-Interazioni con ArchEle.....16
Figura 5-ZODB vs. Oggetti.....17
Figura 6-Tempo di creazione in funzione del numero di oggetti.....18
Figura 7-Funzionamento Isaac client.....24
Figura 8-ArchEle: scelta della tipologia di documenti su cui operare.....26
Figura 9-Interfaccia di modifica/inserimento di un Cambio Maintainer.....29
Figura 10-Dettaglio del riempimento campi da operazione.....30
Figura 11-Dettaglio selezione multipla con pochi domini.....30
Figura 12-Dettaglio selezione multipla in modifica.....30
Figura 13-Dettaglio ricerca dominio in modifica Cambio Maintainer.....32
Figura 14-Selezione domini nei cambi multipli.....32
Figura 15-Scalabilità ArchEle.....35