Submited to EuroPython 2004 - Göteborg, Sweden on June 7-9, 2004

An acquisition, search and retrieval system based on Zope/Plone

Marco Andreini – Cristian Lucchesi – Maurizio Martinelli – Giuseppe Vasarelli

Institute for Informatics and Telematics (IIT)
Italian National Research Council (CNR)
Area della Ricerca di Pisa – Via G.Moruzzi, 1- 56124 Pisa - Italy
Email: m.andreini@iit.cnr.it, c.lucchesi@iit.cnr.it,
m.martinelli@iit.cnr.it, g.vasarelli@iit.cnr.it

Abstract. The amount of paper documents that need to be digitalized is huge. It is useful to have a system to capture, search and retrieve them on-line in a simple way.In this paper we present an acquisition and information retrieval system based on Zope/Plone, that allows a quick definition of customized data type and an easy management of the storage of digitalized documents. Extending Archetypes, it is possible to obtain the relevant interfaces and dynamic validations that allow multiple users to input such documents in a simple and quick way. In addition, the python client, which has been designed to work on HTTP/HTTPS, automates the acquisition phases and the delivery of the data to the server. Making the storing of data independent from the ZODB (the limit of which is highlighted by our benchmarks) and making it be dependent just on the transactional file-systems and on the Postgresql DBMS, it is possible to support a good scalability even for millions of documents and for hundreds of GigaBytes of images. The architecture is fully compliant with web standards and with its design principles. The approach of this paper is applied to a real case study regarding the acquisition, search and retrieval of millions of paper documents belonging to the Italian Registry of the “it” ccTLD, managed by IIT-CNR.

Introduction

The problem of filing and storing paper and electronic documents is of great concern for public administration as well as private companies.

In this article we present 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. Documents of the same kind are grouped together into “collections” in order to improve data storage and search.

Lastly, we present several conclusions and a plan for further development of the project.

The architecture is fully compliant with web standards and with its design principles.

The system we describe is called ArchEle (in Italian, Archiviazione Elettronica). The system's architecture is represented in *Figure 1*.

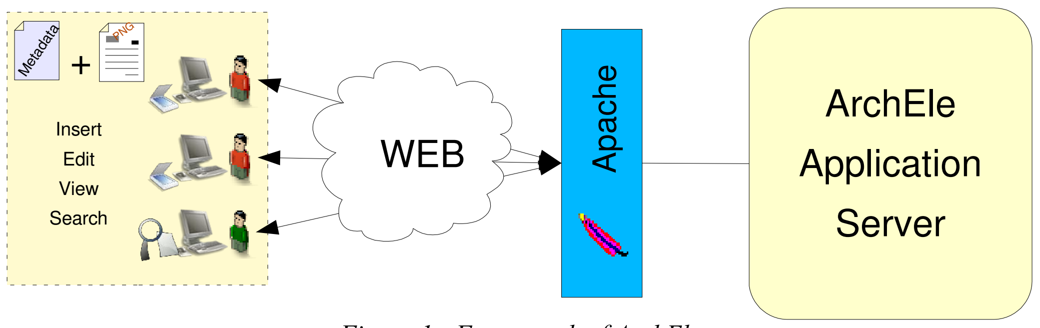

Figure 1 - Framework of ArchEle

Requirements

When designing the system the following requirements were considered:

- scalability, even for millions of documents: for example, the amount of paper documents in the public administration (at least in Italy) is enormous;
- scalability for gigabytes of images as well: in the case of paper acquisitions, the average size of a black- and- white document in A4, at 300 dpi, in PNG format is around 80 Kbyte;
- scalability, for both insertions and retrieval: the system should be accessible via Web and have short access times;
- types that can be personalized: on every type of document to be catalogued, it should be possible to perform a search in different fields (the document's metadata);
- acquisition by scanner: filing must take into account the acquisition of documents by scanner, an operation that should be integrated as much as possible with the insertion of metadata in the application;
- expandability and maintenance: if necessary, the system should be able to be integrated with other applications or external data sources;
- interface usability: interface design should be centered on user needs in order to shorten the time it takes to learn about the system and its uses;
- rapidity: the documents to be acquired may be very numerous; therefore procedures are needed that permit the system's users to perform the operation quickly.

Design Choices and General Architecture

Python as a reference language

Since the system's architecture was intended to be as expandable as possible, the reference language chosen was Python [PYTHON], which allowed us to:

- have a consolidated development language;
- obtain the rapid development of the applications thanks to the high degree of abstraction and the available libraries;
- render the system multiplatform;

- have a system with good performance;
- be able to perform frequent re-factoring of the code without having to re-write or fill out the entire application.

Zope as a framework

Since the application would be using the Web as the main means of interconnecting various parts of the system, we decided to use Zope [ZOPE] as a framework which permits writing web applications, thanks to its solid object oriented structure and on a series of products that utilize Zope as a platform for developing the server part of our application.

Among other things Zope already provides:

- a database for objects
- users, groups and roles;
- integration with other DBMS
- a language for defining templates (*TAL*),
- A scripting language for the application's logic (Python)

Plone as a layer for development of the application

For creating the web interfaces we used Content Management Plone [PLONE] which among other things offered:

- an efficient and elegant framework for navigation, based on folders and their contents rather than on connections between html documents;
- a simple tool for creating structured documents, including complex ones;
- users and groups of users that can be managed using the Plone interface;
- graphics and templates for presentations that can be easily personalized based on XHTML [XHTML] and CSS [CSS2] technologies, with particular attention to the tool's accessibility and usability;
- web interfaces for administering sites created with Plone;
- an integrated cataloguing system (portal catalog) that permits easy and powerful search for documents that are structured differently;
- indexing files in various formats: *.doc, .xls, .sxw, .ppt, .pdf ;*
- control of document flow (Workflow);
- control of security and pre-configured roles;
- a series of types with structured content;
- multilingual support for the interface (already translated into a great many languages).

Archetypes for creating dynamic types

Archetypes [ARCHDG] provided us with a degree of abstraction appropriate for building applications based on Plone/CMF (Content Management Framework). Its main feature is that it provides a common architecture for building structured objects. This construction is based on the definition of the type's schema.

Among other things Archetypes offers:

- automatic generation of forms;
- available libraries of types of fields;
- libraries of widgets (elements for generating html) for the forms;
- libraries of validators for the fields;
- easy integration and personalization of elements of the schema: fields, widgets and validators.

Others parts of the system

The webserver Apache was used as a frontend for the application in order to take advantage of some its characteristics: the *mod_proxy*, the *mod_gzip* and the SSL connections.

SQL was utilized for integrating external databases with the application, thus facilitating the task of the application's users who could make use of the information available in pre-existing databases. SQL is only one of the possible forms of communication with external sources of data. The application's framework also permits integration by means of other protocols, such as: *SOAP, XML-RPC,* etc.

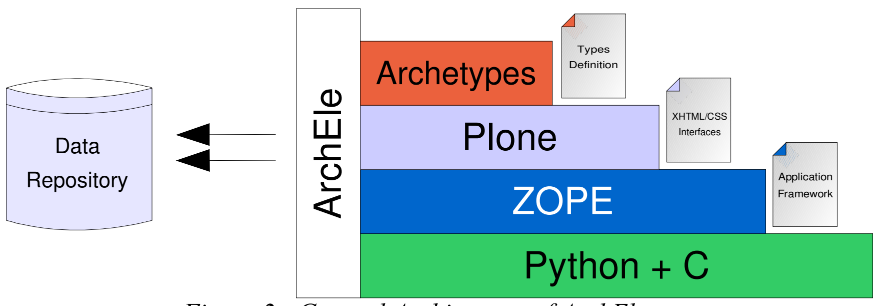

Figure 2 - General Architecture of ArchEle

Figure 2 illustrates the levels of the architecture of ArchEle.

General Features

The general architecture permits rapid development of Content Management and Document Management applications, thanks to the solutions offered by Plone, as well as the rapid creation of new types of objects at the application level, which Archetypes provides.

The startup phase of a new document type simply consists of the creation of its Archetypes schema. The documents can be catalogued in folders, searched for, and managed by means of workflow, all thanks to the functionalities of Plone.

The high qualitative degree of our application base and its easy expandibility have allowed us to concentrate quickly on the integration between documents acquired by scanner and Zope/Plone.

Thanks to the object BaseFolder of Archetypes, which is capable of containing other objects within, creating types of documents that contain their associated images is immediate. The document therefore consists of two levels of information:

- metadata defined by the Schema (Archetypes);
- the images contained within the object;

In order to not overload the ZODB, ExtFile/Ext/Image can be utilized to memorize the acquired images, while for metadata one can use the storage provided by Archetypes (AttributeStorage, MetadataStorage, PostgreSQLStorage, MySQLSQLStorage, ...).

In order to complete the business logic the necessary code for acquisition should be added:

- methods for activating the action of acquisition by sending the client (the browser) a suitable *Content-Type;*
- methods for evaluating the client's credentials (authorizations);
- methods for storing data in the application;
- actions for acquisition in the modification template and in the template concerning content management.

These parts can be implemented as Skin in a Zope product (ArchEle), which we have also used to contain the other extensions.

Clients for acquisition by scanner

In order to acquire documents directly from the scanner and send them to the server, a small software program in Python was created for these tasks. Until recently, only a version of the client for Windows had been developed, but thanks to the SANE (PIL-SANE) project it is very simple to implement a version for Linux.
To develop the graphic portion we adopted the framework wxPython [WXPYTHON] which permits building graphic interfaces that can be carried on several platforms. The wxWindows environment on which wxPython is based is distributed for Linux (in the GTK+ environment) for Windows as well as other platforms.

For acquiring images from the scanner we used the TWAIN Module for Python [TWAINPY], which can interface in a Windows environment with nearly all scanner types on the market. The code needed for this functionality is reduced to a few lines that perform the task of transferring the data of the Twain driver to the program itself. In this phase we also took into consideration that the application would acquire various images (FAX or other paper documents) by means of a scanner with Automatic Document Feeder (ADF).

Acquiring images by scanner consists of the following phases (*Figure 3*):

1. the operator requests the acquisition by the actions "*save"* or *"save and acquire";*
2. the application provides an http response with *Content-Type* ad hoc (i.e., application/x-isaac) containing the credentials of the current user, the required version of the client and the URI to which the acquired images should be sent;
3. thanks to the addition of an entry to the Windows Register [WINREG] set at the time of installation of the client and which associates the acquired client with the abovementioned content-type, the browser automatically runs the client, providing it with a temporary file containing the ArchEle response;
4. the client requests the scanner for the acquisition of images (note that the scanners in our case have an Automatic Document Feeder);
5. using Twain, the client acquires the images from the scanner, available in BMP format, in an asynchronous way;
6. the client converts the images from BMP to PNG.

The client sends the PNG images to the URI provided by the application server. Together with the images it also sends the credentials that had been sent to it initially.

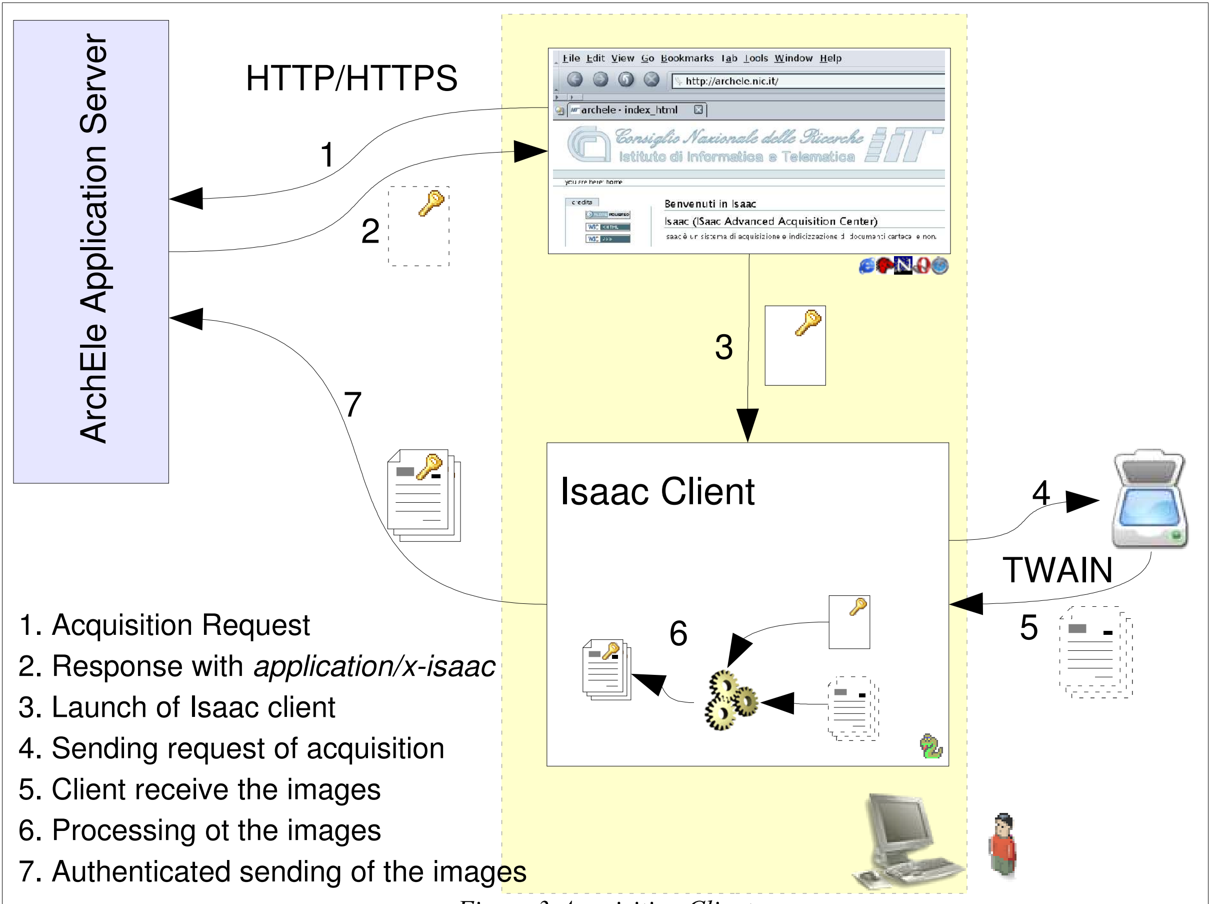
ArchEle Application Server
HTTP/HTTPS
1
2
3
7
Isaac Client
4
TWAIN
5
6
1. Acquisition Request
2. Response with *application/x-isaac*
3. Launch of Isaac client
4. Sending request of acquisition
5. Client receive the images
6. Processing ot the images
7. Authenticated sending of the images

Figure 3-Acquisition Client

Limits of this architecture

When using the abovementioned application for millions of documents, filing objects within the ZODB can be problematic. Although solutions have been attempted for reducing the load on the ZODB (for example, by memorizing the images in ExtImage) performance problems were encountered due to the insertion of millions of Archetypes objects. We performed a series of surveys on various indicators, from which analyses were made, several of which are significant:

- for the insertion of each object a series of transactions are performed, that cause an overall increase in the size of the ZODB by approximately 160 KiB; in these transactions the new Archetypes objects, the modifications of the BTreeFolder that contains it, the ExtFile of the images and the additions to the indices (portal_catalog) are all present;
- with this model the ZODB grows by up to 250 MiB per day, which makes a daily "pack" necessary;
- the Archetypes objects – even those with small fields – are about 6 KiB in size after the "pack", of which only a small part (average about 300 Bytes) are the metadata and the other ones are structures of Archetypes objects;

- the server was subjected to a considerable load due to the elaboration of a great many objects involved in each operation; this led to long waiting-periods during the normal work of the three operators: the load average of the server was always high (~*3*);
- elaboration times very clearly depend on the increase in the number of objects contained and catalogued.

In figure 4 we show the steady increase of ZODB size *(packed)* according to the number of objects contained. From this, we can assume the future size of ZODB size and therefore the load of the entiry system.

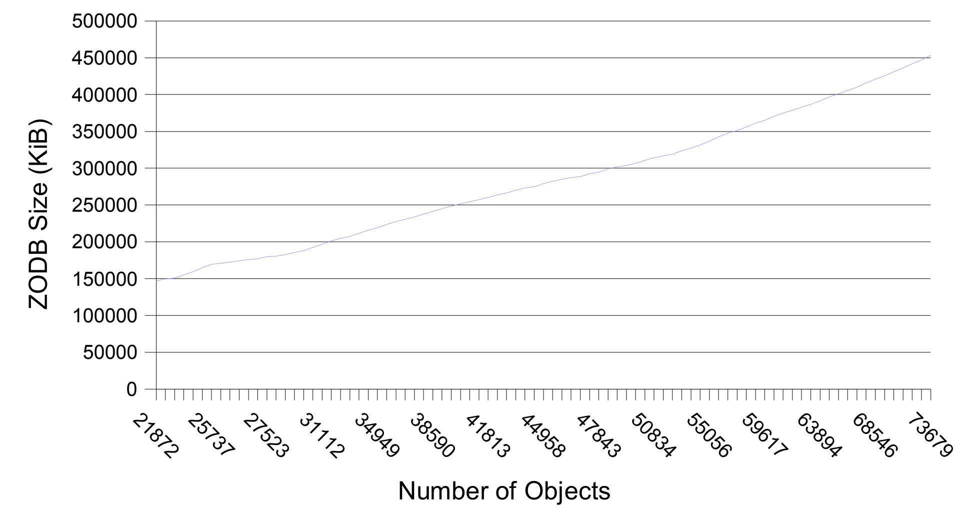

Figure 4- ZODB Size/Number of Objects

Figure 5 represents the graphic of the statistics concerning the time in the creation of the objects according to the number of items already stored in the database. The creation time was:

- < 1 second (good): when the objects stored were less than 25000 (early stage of development of ArchEle);
- > 5 seconds (ugly): when the objects were between 25000 and 130000, depending on number of objects in ZODB;
- < 1 second (good): when the objects were more than 130000, and the documents were stored completely out of ZODB (last development of ArchEle).

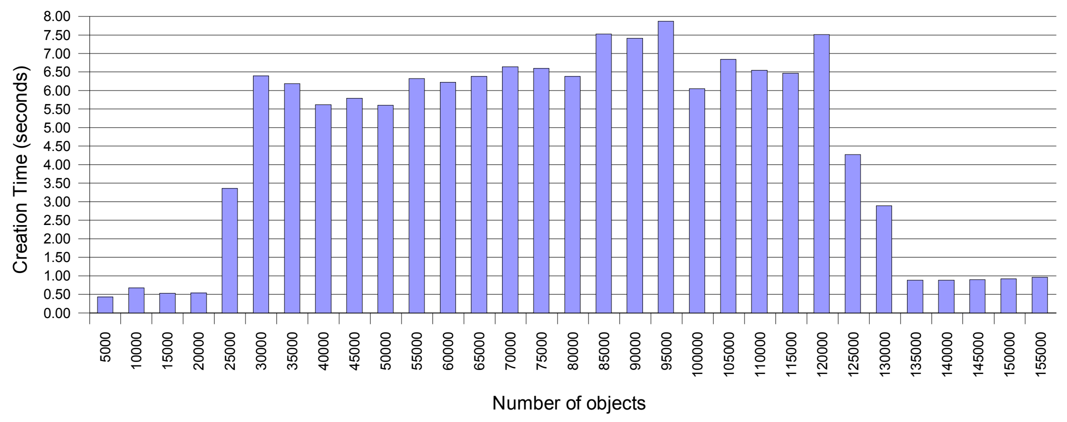

Figure 5- Creation Time/Number of objects

Obtaining the requested scalability

To overcome the above mentioned limits and keep the code compatible with Archetypes – and thus to follow future developments – a new storage class was created that allows the filing of millions of documents without relying on ZODB (*Zope Object DataBase*).

All fields with SQL storage depend on the UID (an object method). In practice, the UID is the key of a table containing the values of various fields with SQL storage. A class base in ArchEle was implemented that redefines the UID method, so that it can depend directly on the session (cookie). A new storage system was also created starting from the PostgreSQLStorage, by which the values that are read and/or written in the current instance will be relative to the UID value specified in the session.

The result is that with a small amount of code it is possible to utilize the logic of Archetypes without however effectively adding any object in the ZODB for each new document. This obviously makes the system faster, but just as obviously one loses the biunivocal relationship between objects in the ZODB and documents acquired. In the Zope database, there is only one Archetypes object for each type of document (collection). In order to select one of the elements of the collection the UID in the session is exploited. In a logic starting from the *FlyWeight* pattern [PATTERNS], one or more Factory Objects for each collection exist, containing within the relation of selecting the elements: the UID in session.

A further step in the development of ArchEle was the development of a class that permits storing directly in the filesystem (in the ExtFile), using a variable method – usually the UID method – to obtain the physical path of the documents in the filesystem. Note: this class is implemented using the *path.py* [PATHPY] module and re-uses many of the constructs already found in Zope.

The storage of these images is still usable in an architecture consisting of ZEO and many Zope (ZEO Client), providing shared access to the same filesystem (we use NFS [NFS] [RFC1813]).

In ArchEle, due to the flexibility of the “collective” products [COLLECTIVE], a code was written that implements an additive transformation that on fly converts *tiff* into *png* (see PortalTransform).

To reduce the elaboration needed at the server level, a version of the client (Isaac) was subsequently written that carries out the phase of conversion (via Python Imaging Library [PIL]) from BMP to PNG format directly on the computers of the operators.

Case Study

This application has been successfully used at the Italian Registry of the “it” ccTLD, managed by IIT-CNR. The application files the paper documents regarding Letters of Assumption of Responsability for the registration of domain names and letters for Changing Provider/Maintainer.

Filed using this application were:

- 186000 Changes of Provider Maintainer, with approximately 165000 associated images (several Changes had reference to the same images) for a total of 16GiB of images;
- 213000 Letters of Assumption of Responsibility with approximately 503000 associated images for a total of 28GiB of images.

It is estimated that soon the following will be filed within the application:

- about 400000 Changes of Provider Maintainer for a total of approximately 35GiB of images;
- about 700000 Letters of Assumption of Responsibility for a total of approximately 92GiB of images.

The application functions efficiently on an IBM e-family server with XEON Processor at 700MHz, 1GiB Ram, 2 SCSI disks of 9GB in Raid 1 and box of external IBM disks of 260GiB.

The software programs used on the server are: Debian GNU/Linux (sarge) with kernel 2.6.5, Zope 2.7, Plone 2.0, Archetype 1.3-cvs, Postgresql 7.4.2, FileSystem ReiserFS v3.6 and XFS.

The application was used simultaneously by three specialized operators with the task of acquiring paper data, and by the operators of the registry that consulted the data in the application.

To increase the velocity of the insertion of metadata associated with the documents and to have more control over them, ArchEle was integrated with the database of the domains of the Italian Registry. The problem of the acquisition of paper documents had been addressed previously by the IIT using a proprietary standalone application for Windows, acquiring an average of 1775 documents per month for each data-entry worker. With ArchEle, using the same hardware and no licensing costs, the data were decidedly improved: in less than a year from the onset of the development about 190000 documents were acquired, averaging about 6640 documents per month for each data-entry worker (+270% productivity).

The IIT institute has also used this method (Python+Zope+Plone) for other internal applications.

Future Developments

Since in 2004 the Italian legislation introduced the obligatory use of the Information Protocol in Public Administration, the application could allow extensions for:

- affixing a univocal protocol number;

- workflow management of the documents acquired;
- digital signature of documents by the Isaac client;
- negotiation of digital certificates to permit accreditation of the clients.

To (re)introduce the workflow and other related characteristics correlated with the objects of Zope a different, more Object Oriented strategy for storage is probably needed. In this sense it is possible to evaluate the use of APE which, at the time the first part of the code was written, was at an initial stage and was not very well – integrated with Archetypes. In fact this would permit the use of a coherent layer for storing the metadata, images and all other properties of the objects.

At this time, the application's purpose is managing the paper documents acquired, for which it is unnecessary to carry out a full text search, but from the vantage point of utilizing the application for the management of documents in a non-paper format (MsWord, PDF, XML, ...) as well, we are working to test the scalability of indexing document contents by means of the Zope ZCTextIndex and discover solutions that would allow us to index the content of hundreds of GigaBytes of documents as well.

The example presented here illustrates the use of Python, Archetypes/Plone/CMF and Zope for rapid development of efficient web applications that are different from those dealing with content-management only.

References

[PYTHON]	Python Home Page, , http://www.python.org/
[ZOPE]	Home Page Zope, , http://zope.org
[PLONE]	Home page Plone, 2004, http://plone.org
[XHTML]	W3C, XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), W3C Recommendation, 2000-2002
[CSS2]	W3C, Cascading Style Sheets, level 2, 1998
[ARCHDG]	Sidnei da Silva, Archetypes Developers Guide,
[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/
[WINREG]	Microsoft, Description of Microsoft Windows Registry - Articolo Microsoft Knowledge Base - 256986, 2004
[PATTERNS]	Eric Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software , 1995
[PATHPY]	The path Python module, 2004, http://www.jorendorff.com/articles/python/path/
[NFS]	Network File System, 2004, http://nfs.sourceforge.net/
[RFC1813]	Callaghan, Pawlowski, Staubach Sun Microsystems, Inc., RFC 1813 - NFS Version 3 Protocol Specification, 1995
[COLLECTIVE]	Collective CMF/Plone on Sourceforge, , http://sf.net/projcets/collective
[PIL]	Python Imaging Library (PIL), 2004, http://www.pythonware.com/products/pil/