Concept and Implementation of an Ontological Document Management System

Eric Simon, Iulian Ciorăscu and Kilian Stoffel
Information Management Institute
University of Neuchatel, Switzerland
{eric.simon|iulian.ciorascu|
kilian.stoffel}@unine.ch

Abstract

In this paper we describe a new architecture and implementation that allows the integration of a Document Management System with ontological support into a Web CMS. The architecture is motivated by the needs we identified in several real world projects. Our system is based on accepted open standards allowing anyone to replicate it or to use our system without restriction.

1. Introduction

Document Management Systems (DMS) are the basis of almost all business and management information systems. They are sustaining the pyramid of a company's internal knowledge. These systems facilitate the organizational learning and knowledge creation. They are designed to provide rapid document retrieval to knowledge workers, reduce error rates, control documents access, and significantly improve business performance. Document Management Systems try to give the user control over their companies' institutionalized knowledge.

Recent developments such as ODMA (Open Document Management API [ODMA 2005]) for simplifying the integration and interoperability of standard desktop applications with Document Management Systems, as well as standards for representing knowledge in open formats (e.g. OWL for ontologies [McGuinness 2004]) change the way DMS are perceived. They evolve from sophisticated search to more and more complex knowledge creation, management, control and distribution systems. The available knowledge has to be integrated into the companies business processes, products and services. This helps companies become more innovative and agile providers of high quality products and customer services.

This leads us to the third aspect important to the work described in this paper. Companies web sites are often alimented by content provided by a DMS. Especially Web sites targeting user support are based upon the companies internal DMS. This is of course also true for intranets providing up to date information to the companies collaborators. Most of the intra- and extranets are based on some form of Content Management Systems (CMS) [Cooper 2004, Latteier 2001]. The goal of these systems is of course the collaborative creation of documents and other forms of content. This process is very closely related to document management. On the one hand the content created in a CMS is often based on existing documents and on the other hand a CMS often provides new documents that should be integrated into the DMS. Therefore a tight integration of these two systems (DMS and CMS) is very important for an efficient management of the overall information and knowledge of a company.

In this paper we describe a new practical architecture and its implementation that allows the integration of the three main components mentioned above, i.e. a DMS with ontological support to structure the data that is integrated directly into a CMS. The architecture was motivated by the needs we identified in several projects in direct collaboration with institutions working in the health, bioinformatics, security, and linguistics domains. The goal was to find a system-architecture that is as open as possible in order to integrate new components as smoothly as possible. From an implementation's point of view we based our system on accepted open standards that would allow anyone interested in a similar system to replicate it or to use our system without restriction.

The remainder of the paper is structured in the following way. In the next section we describe the overall architecture of the proposed system. In the third chapter we show how the integration of ontologies can greatly improve the structuring capacity of a DMS. Chapter 4 describes how within our architecture we can deal with security issues. In Chapter 5 and Chapter 6 we will describe the implementation of the system followed by an illustrative application. Finally we will conclude and give some further directions.

2. The Architecture

There are two main reasons for defining an architecture such as the one presented here. Namely, openness (offering the possibility to integrate any existing tool) and the ease of the integration of the system into an existing information management infrastructure.

As outlined in the Introduction the rough architecture of the proposed system will resemble the structure of a CMS as shown in Figure 1. On one side the system is connected to all internal and external document sources such as DMSs, file systems, data bases or other data repositories storing documents. This information is internally restructured using ontologies in order to give the user more rapid and more accurate access to their documents. The access to the system is guaranteed through a web interface as shown in Figure 1.

This architecture in three layers facilitates the integration of existing document resources through the different access mechanisms offered by each system. The integration of the system itself into an existing infrastructure is facilitated as the third layer allows a standard http-integration. In the following we will give some further details on each layer. The implementation details will be given in Chapter 5.

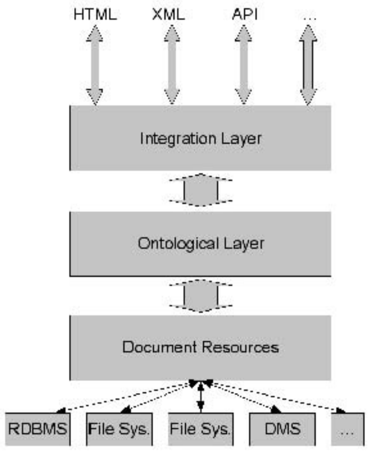


Figure 1. Overall architecture.

The first layer essentially provides the data integration functionality. This is done in two different ways.

One consists of an internal database that is part of our system. If documents are uploaded and they do not belong to any existing Document Management System, then they will be stored in the system's internal store. The functionality provided by this store corresponds roughly to that of an object oriented data base. The overall architecture is independent of the choice of this data base system. If any preferences for a given data base system exist then this system can easily be integrated as all interactions with the store are realized through standard APIs.

The other way consists of the integration of existing Document Management Systems. Our system provides an interface to integrate these systems. However, they have to provide an access mechanism to the documents and our system provides an interface to specify them. Once these specifications are given, the integration into the system is seamingless. In the upload procedures the user can specify in which store he wants to have his documents and in the search process it is possible to restrict searches to certain data bases. These are the only two places where the different stores are visible.

The second layer is the heart of the system. It provides all the semantics that is added to the system through the use of ontologies. The ontologies are used to fulfill mainly three tasks. Firstly, they are used to classify documents that are added to the system. Secondly, they are used to filter the data in the system, and finally they are used to formulate queries and present the results.

The third layer is constructed in such a way that it can easily be integrated into an existing Business Information System. The integration can be realized through a standard web interface or through a web service. These are the two mechanisms currently offered. Other mechanisms can easily be added.

As one can see this architecture is very close to a standard 3-tiers architecture.

3. The Ontologies

Using ontologies for the meta data repositories provides several advantages over classical approaches used in Document Management Systems:

a) Better organization of the data - Ontologies are inherently structured, and they add a semantic layer over the document repository, replacing classical keyword-like meta data annotation of documents with semantic annotations and contexts. This allows a better, more powerful organization of the documents, and facilitates scalability.

b) Increased expressiveness of the query language - The ontological layer also acts as a semantic index to enhance the expressiveness of the query language. One of the principal reasons for the popularity of ontologies over the last couple of years is their potential use for creating a semantic index for the web [Fensel 2003]. The basic idea for a semantic Document Management System remains the same as the one used for the web. It is however much easier to realize as the content of the document data base is known.

c) Humane readable presentation of search results - The results of the search queries are presented using the ontological relations, which allows for a presentation in context. This presentation facilitates the navigation within the results and facilitates quick drill downs. The time necessary for finding relevant documents is greatly reduced.

d) Easier analysis of the documents - Often the analysis of some characteristics of documents is necessary for applying techniques of document clustering or text mining. The semantic annotation enabled by ontologies can be very useful in disambiguating terms and increasing the overall performance of these techniques. As the size of the document corpora are constantly growing text mining is becoming increasingly important and therefore it is crucial to provide efficient support for these techniques.

3.1. Detailed explanation of ontology usage

Document Management Systems typically use RDBMS as an underlying technology for meta data storage, and indexes for efficient search. While sufficient in many cases, usually when the structure of the data does not go beyond a tree-like classification and/or the search requirements are of a keyword based kind, this approach presents limitations as soon as the interrelations between elements are more complex or the user wants to browse the data using advanced filtering techniques. As soon as more relations between words such as synonyms, meronyms, hyponyms, are used, ontological support is needed.

Typically, the most important aspect of a Document Management System is to allow the users to quickly and efficiently find and retrieve the documents, based on different techniques:

a) Keyword search - This is the most obvious way of searching for information, and works very well when the user knows quite precisely what he's trying to find and the related keywords to use [Pepe 2005]. The idea here is to sufficiently narrow the results using a combination of keywords, so that the user can pick the desired document from a small enough list at a single glance. The problems with this technique are two-fold: first the user has to have a very precise knowledge of the representation of the meta data and the domain, and the amount of documents has to be quite small and very well categorized using the corresponding keywords. Techniques such as synonyms, pruning, ranking etc. are able to improve the results, but within limits.

b) Repository browsing - If the amount of document is sufficiently small, and very well structured in form of a tree, it is then sometimes easier to just browse through the entire structure to find the required document. This is typically what a computer user does when searching for a document on a file system. The problem with such an approach is well known, as some documents tend not to be easily categorized in only one branch of a tree, and the vast amount of documents typically used in a Document Management System is usually much too large for this technique to be applicable.

c) Keyword filtering and browsing - This is a combined technique aimed at narrowing down the number of presented documents by first filtering the results based on a number of keywords, or more complex expressions if necessary, and presenting the information in context, typically a tree with contextual relations (representing a portion of a graph), to allow browsing and more efficiently finding the required information. This is where ontologies are the most relevant, mostly due to points a, b and c outlined above, as they provide the necessary semantic context in addition to the keyword index to present the data in a form suitable for browsing.

The approach we propose to use ontologies in this context is to replace the traditional indexes by an ontology layer, described in Section 3.2. This has the advantage to be simple in design and easy to use, while retaining the classical structure of the data itself (the documents), thus allowing to build on top of existing repositories or provide compatibility with other methods.

3.2. The Ontology Layer

Meta data is represented as attributes of a node in a graph. A node can be a “data node”, a document for example, or an “index node”, a node of any index structure built from the data. The first type of node is straightforward, it is simply an avatar of an actual entity (document), that can be structured as a tree like in a file system. The second type of node contains supplemental information built from the data, for instance an index or another ontological structured semantical information.

This way of structuring the information adds the power of inheritance of attributes and properties given by ontologies to the basic operations allowed by classical indexes. For instance, when a search is issued on a keyword index structure and the word is not found in any attribute of nodes in the corresponding level, it will be searched in all more general concepts of the structure, allowing to bring as result a superclass of documents. It is obvious how the expressiveness of the search can be increased using this technique.

Another big advantage of using ontologies is for maintainability and scalability of the overall system. Editing documents, adding new documents, coming up with new index entries can be easily envisaged, operations that are time consuming and error prone in a non hierarchical structure. Also, importing and exporting data is facilitated by the very structure of an ontological representation of meta data, which is a great advantage for the development of web services for example.

4. The Security Management

Like every system where the public (could be only a limited set of users) can access information that may or may not be sensible, sooner or later restrictions on what a user is allowed or denied to do must be imposed. We propose a role-based security model which is the most elegant way of giving permissions to a user, based not on who he is, but on what role he plays in the system. Furthermore, the model is well known [Abadi 1993] and implemented in Zope, the framework we used (see Section 5.2, page 11).

To formalize the security model we need four different notions:

User, *Role*, *Permission*, *Location*

A *User* is a unique name given to an agent (real person or application) that interacts with the system. His possible actions are grouped into *Permissions* (like *read/write/add* or even more specific ones, like *add document*). The possible places or contexts where the *User* actions are applied are called *Locations.* A *Permission* by itself is not valid without a location where it is supposed to be allowed or denied. While these three notions are sufficient to have a complete security model it is a very complex and rigid one. To maintain such a system is impractical even with a small user base. A new abstraction has to be added, namely *Roles* that are used to group together multiple (*Permissions,Locations*) pairs. It can be viewed as an abstract user or a group of users.

Each user is then given one or several *Roles.* All permissions given to a *Role* are in fact given to all *Users* that have that role, now or in the future. Therefore the definition of a security policy for the system is split in two parts:

1. defining *Roles*
2. defining *Users* and assign *Roles* to them.

Defining *Roles* is done at the application design level and not at runtime. It depends on the logic of the application and should be carefully designed and tested before the system goes into production. Designing good *Roles* within a system ensures that a user having that role has the freedom to do what is supposed to do but at the same time he is also restricted to exactly these operations.

However, managing *Users* and their *Roles* is not done at the application design level but at runtime. Of course, having a good definition of *Roles* at the application level is not sufficient and it does not ultimately protect sensitive data if *Role* assignment is not correctly executed by the systems managers.

There is another important aspect of security management: authentication. It treats the problem of relating a user in the system to an actual user in the real world by some mechanism such as login/password. That part, although it can be quite complex, is not covered in this article: we will focus primarily on *Roles* used in a Document Management System.

4.1. Example

We will identify several *Roles* in a Document Management System. These enumeration is by no means complete, it is just to exemplify the *Role* concept we defined.

- Guest user (normal user)

  He can browse/search the repository (or a part of it), make personal notes, personal virtual folders, etc. There could be different categories of guest users, depending on what they can see, and how restrictive the repository is. In the most cases, the guest users can see the whole repository. Depending on the repository it may be that guest users could have also write access on a "personal" part of the repository.
- Editor (user with limited write access)

  Like a guest user, he can search/browse the repository, add new documents, update documents.
- Supervisor (user with full write access)

  This role has all the rights of an editor, plus they could change/update the ontological structure of the repository.

Of course this is a very simple view on the security of a Document Management System, every role can be further refined upon the needs of the system.

5. The Implementation

In this chapter we will show some of the implementation details of the system we described.

5.1. The Implementation Platform

To implement a Semantic Document Management System we used a Content Management Framework, Zope [Latteier 2001]. A system could be built from scratch of course, but we found that Zope provides several important parts that we need to create our system. Zope's underlying object oriented storage was already used successfully in related systems such as Indico [Baron 2004].

Here are some of the reasons for using Zope CMF:

- **Web interface application/service** - Since we are targeting a web interface for our system, the natural choice is a web-based CMF. It provides its own dynamic XML-based web language that makes it very easy to create dynamic web interfaces.
- **Integrated security** - Instead of redesigning and implementing the security model from scratch we can use the already proven and tested security model provided by Zope CMF.
- **Object oriented storage** - Zope storage and object model follows an object oriented model and the ontological data can be mapped as such to the Zope storage model.
- **Scripting language** – The Zope CMF uses a scripting language for doing almost all of the dynamic content generation. The scripting language, Python, is widely used and greatly facilitates the integration of external resources written in other programming languages.
- **Open source** - Last but not least Zope is an Open Source product and therefore allows the development and installation of test systems without important up front investments. Also, it's supporting community makes it a very reliable environment.

There are several versions of Zope. At the time of the writing of this article, the stable tree is 2.x and the new redesigned development tree 3.x. We used the production version, keeping in mind the upcoming version 3 during the design phase, so that when version 3 will become production-stable, it will be very easy to port the system to it.

5.2. Semantic Document Management System Zope Product

The Zope way of creating deployable, self-contained web portals is by using Zope Products [Latteier 2001]. There are two ways to create a product, one is by using the management interface provided by Zope and the other one is creating the product by using Python classes and functions.

The first method is easier to use when creating small projects but it becomes quickly unmanageable when more than one person develop the system. Zope's own „undo“ feature does not match a fully featured Versioning System. So we decided to create the product as a python module.

The base class used to keep the document repository and the classification ontologies is derived from ZCatalog and performs all the indexing of the repository. There are also two other important classes, one to keep an ontological object (classification) and another class to define a Document entity with all meta data and its ontological classification. Real file documents are simple Zope file objects that belongs to this Document entity.

Web interface pages are written using the Zope Page Template Language and DTML.

The entry page of the web interface shows a main categorization of documents using a tree-like ontology (see Figure 2, page 13). For every node all documents that are related to it are shown.

If there is more than one ontological categorization the user is given the possibility to browse the documents using all available ontologies. Depending on the ontology structure it can be mapped into a tree, or a list used in filtering. If a user is authenticated and has the Editor role, he has the possibility to create new document entities, or edit existing documents. By editing existing document entities he can reclassify it, change their description, meta data, or even add/delete document files. However, he cannot change the ontological classification structure unless he has the Supervisor role.

Changing the ontological classification structure will consist in most cases in adding new categories, and further refining existing ones. The categories nodes don't have any associated documents. If a category is refined the user can use filters and searches to assign the documents from the general category to the more specific categories.

It may be possible however to delete a classification node. This is a case that needs to be treated with special care because all the documents from that category should be reclassified.

Indexing and Searching

For the search engine we used the Zope's built-in class ZCatalog that we extended in the main product class. We used several indexes with different types. Some of them we enumerate here:

– text based index to index the documents
– text based index for descriptions (of ontological nodes, documents)
– field based index for meta data (author, date, owner and other attributes)

The search engine uses the text based indexes to search in the documents from the repository or in their description. It will return as result a „rank-ordered" list of documents. However we used the search engine also to find specific ontology entities and afterwards show all documents belonging (or related) to this entity. The results are classified and presented using their ontological context thus giving the user more information for the documents (e.g. If a user has 20 documents as a result and they are in totally different categories he can easily choose the correct subset of documents by clicking on the category). However, if the result set is large, by showing the context he can narrow the search by restricting the result to only one or any other subset of categories.

The metadata indexes are also used to filter documents shown in a page. This case differs from the search only by interpretation, it uses the same mechanism.

Security Implementation

Zope's internal security model includes all we described above and provides even some further functionalities. In Zope the users cannot be assigned individual permissions. Zope users acquire permissions by using Roles. A Role can be assigned different permissions. Zope comes bundled with several predefined roles, however, for our project we created new roles: Guest, Editor, Supervisor just as described in the previous chapter. Zope's security model allows for further refinement by using „local roles“, i.e. allowing a user to have different roles depending on the location.

We also need to define permissions for the possible actions, like View, Create Document, Edit Document, Edit Ontology and assign these permissions to the corresponding roles.

We also considered using built in authentication and rely on the product logic to implement this part of the security model. But this approach is error prone and if a user succeeds in exploiting a bug in the application he would have full access to the database.

Using the underlying security model ensures data privacy on a lower level (base level) and even if a bug is found in the application, the data is still protected by Zope security model.

6. An Example

This architecture has already been used in web applications using the described implementation. These applications were mandates by various companies and are confidential, but one implementation is being developed as a collaboration within our university and is described shortly below.

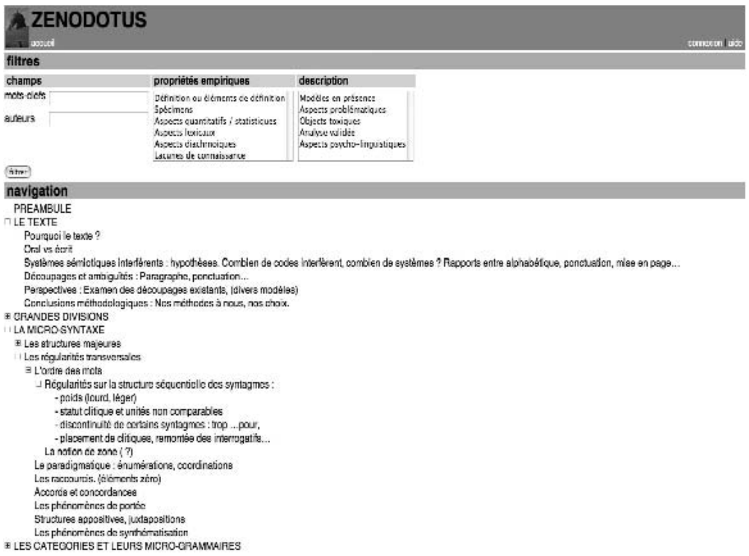
The system, called Zenodotus, is a knowledge repository for modern French, aimed at creating a kind of encyclopedia of articles about various aspects of the language. If we refer to Figure 1, the user interface consists as already mentioned in a web application, the ontological layer refers to the supplied classifications of concepts of modern French (see below) and the document resources are the actual documents, text and sound, or notices for bibliographical references.

The important part here is the classification, which is two-fold:

1. An actual table of content of all the concepts.
2. A grammatical structure, which can be further separated in two subcategories: empirical properties and description of the concepts.

The interface allows filtered browsing in the sense explained in Section 3.1, using keywords and filtering over the grammatical structure to narrow down the number of results in the table of content (see Figure 2). Furthermore, one can have a look at what the main ontologies look like for this system, namely in the two boxes on the top part and the tree in the navigation part at the bottom. The ontology is actually more than a tree, but information about other links is then represented in context when the user selects a node, to ease the readability.

The user management is very basic in this case, with only a few users allowed to submit or modify new documents and notices. There is a mechanism of retaining the owner of a particular document to allow tracking changes, and some privileged managers have rights over the whole content.



Définition ou éléments de définition
Spécimens
Aspects quantitatifs / statistiques
Aspects lexicaux
Aspects diachroniques
Lacunes de connaissance

Modèles en présence
Aspects problématiques
Objects toxiques
Analyse validée
Aspects psycho-linguistiques

PREAMBULE
LE TEXTE
Pourquoi le texte ?
Oral vs écrit
Systèmes sémiotiques interférents : hypothèses. Combien de codes interfèrent, combien de systèmes ? Rapports entre alphabétique, ponctuation, mise en page...
Découpages et ambiguïtés : Paragraphe, ponctuation...
Perspectives : Examen des découpages existants, (divers modèles)
Conclusions méthodologiques : Nos méthodes à nous, nos choix.
GRANDES DIVISIONS
LA MICRO-SYNTAXE
Les structures majeures
Les régularités transversales
L'ordre des mots
Régularités sur la structure séquentielle des syntagmes :
- poids (lourd, léger)
- statut clitique et unités non comparables
- discontinuité de certains syntagmes : trop ...pour,
- placement de clitiques, remontée des interrogatifs...
La notion de zone (?)
Le paradigmatique : énumérations, coordinations
Les raccourcis. (éléments zéro)
Accords et concordances
Les phénomènes de portée
Structures appositives, juxtapositions
Les phénomènes de synthématisation
LES CATEGORIES ET LEURS MICRO-GRAMMAIRES

Figure 2: The main interface

The interface to add and edit notices and documents is straightforward, but fixed. There will certainly be the need to allow for adding new structural information (new categories) inside the interface, but currently this is done outside the system.

7. Conclusion and Future Work

We presented in this paper a new architecture that allows an easy integration of a semantic layer into a company's Document Management System. The semantics is added to the system through domain ontologies. The architecture is designed in such a way that the integration of existing Document Management Systems as well as other document resources is done through their own access mechanisms. On the other hand the system itself can easily be integrated in the companies information system infrastructure using open standards.

Furthermore we have presented an implementation (proof of concept) of the system using an open source framework (Zope). This implementation is used in several of our projects and is available on our web site.

As currently several implementations are being evaluated, the further improvements will essentially be based on the user feedback. Developments that have currently started are mainly dealing with two aspects, namely the improvement of the user/security management and the adaptation of our architecture for other platforms such as .NET.

Based on the experience gathered so far we know that the integration of existing document sources into our system is easy to realize and the integration of the system itself into an existing information infrastructure poses no major problems. For the project where domain ontologies were available already, the expressiveness of the DMS was greatly improved in a very short amount of time. Using standards such as OWL was of great help as large libraries of ontologies exist already. As conclusion one can say that the proposed architecture and implementation seems to overcome several of the major drawbacks of classical DMSs and can easily be integrated into en existing infrastructure.

8. Acknowledgments

This work was supported by the Swiss National Science Foundation, project number 200021-103551/1.

9. References

[Abadi 1993] M. Abadi, M. Burrows, B. Lampson and G. Plotkin, *A Calculus for Access Control in Distributed Systems*, ACM Transactions on Programming Languages and Systems, 15(4):706--734, Oct. 1993.

[Sutton 1996] Michael Sutton, *Document Management for the Enterprise: Principles, Techniques, and Applications*, John Wiley & Sons, New York, 1996.

[Fensel 2003] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster (eds.): *Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential*, MIT Press, Boston, 2003.

[ODMA 2005] AIIM International, *ODMA 2.0 Specifications and Software,* **http://ODMA.info**, 2005**.**

[McGuinness 2004] Deborah L. McGuinness, Frank van Harmelen, *OWL Web Ontology Language* W3C Recommendation 10 February 2004.

[Cooper 2004] Cameron Cooper, *Building Websites With Plone*, PACKT, UK, 2004.

[Latteier 2001] Amos Latteier, Michel Pelletier, *The Zope Book*, New Riders, 2001.

[Ciorascu 2005] I. Ciorascu, E. Simon, K. Stoffel, “An Ontological Document Management System in Zope”, Technical Report, University of Neuchâtel, 2005.

[Klischewski 2003] Ralf Klischewski, “Towards an Ontology for e-Document Management in Public Administration– the Case of Schleswig-Holstein”, Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS’03), IEEE, 2002.

[Pepe 2005] A. Pepe, T. Baron, M. Gracco, J.-Y. Le Meur, N. Robinson, T. Simko, M. Vesely, “CERN Document Server Software: the integrated digital library”, ELPUB 2005 conference, Heverlee (Belgium), 2005.

[Baron 2004] T. Baron, J.B. Gonzalez, J-Y. Le Meur, H. Sanchez, V. Turne, ”INDICO - the software behind CHEP 2004”, In CHEP 04, Switzerland, 2004.