UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in Ingegneria dell'Informazione

Tesi di Dottorato di Ricerca

Enterprise knowledge management: introducing new technologies in traditional Information Systems

Autore:

Jacopo Viotto _____

Relatori:

Prof. Francesco Marcelloni _____

Prof. Andrea Tomasi _____

Ing. Salvatore Parisi _____

Anno 2010

SOMMARIO

I sistemi per la gestione della conoscenza che vengono presentati negli articoli di ricerca vengono raramente implementati nelle realtà aziendali, almeno su larga scala. Le aziende spesso sono legate ai sistemi che già possiedono, e non possono o vogliono rivoluzionare la situazione per far posto a soluzioni completamente nuove. Date queste premesse, questo lavoro approfondisce varie piccole modifiche che possono essere applicate ai Sistemi Informativi già presenti in modo da migliorarli con nuove tecnologie senza grandi trasformazioni né discontinuità di servizio. L'argomento è l'interoperabilità, con un particolare accento sulla promozione dello standard ebXML sui registri. E' stata definita un'interfaccia universale per la gestione documentale, ed i sistemi ad essa conformi sono stati organizzati in un'architettura appositamente ideata per il supporto ad ebXML. Questo ha permesso la manipolazione standardizzata di sistemi documentali legacy.E' stato inoltre affrontato l'argomento strettamente correlato della gestione semantica della conoscenza. Abbiamo sviluppato un sistema di integrazione di tool semantici all'interno di repository tradizionali con basso impatto architetturale. Infine, abbiamo discusso un nuovo problema interno alla categorizzazione di documenti, ed un nuovo tipo di ontologia che può essere utilizzata in tal contesto.

ABSTRACT

Knowledge management systems described in research papers are rarely seen implemented in business realities, at least on a large scale. Companies are often tied to existing systems and cannot or would not revolutionize the situation to accommodate completely new solutions. Given this assumption, this work investigates several small-scale modifications that could be applied to in-place Information Systems so as to improve them with new technologies without major transformations and service discontinuities. The focus is interoperability, with a particular stress on the promotion of the ebXML registry standard. A universal interface for document management was defined, and the conforming "interoperable" DMSs were arranged within an architecture explicitly designed for ebXML-compliant access. This allowed standards-based manipulation of legacy DM systems. The closely related topic of Semantic knowledge management was also tackled. We developed Semantic tools integration for traditional repositories with low architectural impact. Finally, we discussed a novel issue in document categorization, and a new kind of ontology that could be used in that context

TABLE OF CONTENTS

1. INTRODUCTION

This dissertation focuses on the complex relationship between the academic and the business world, and outlines several viable solutions for their integration. The starting point of this investigation is the consideration that most systems proposed in literature do not actually get implemented in the real world. Many technical and non-technical impediments exist: the biggest one, encompassing all the others, is the requirement of a complete substitution of the Enterprise Information System, or some part of it. We thus propose a different perspective, based on small modifications of existing systems. We show how this approach is able to boost their performance while having a marginal impact on overall architecture, and, most importantly, without disrupting in-place services.

The central issue tackled in this document is interoperability. In this context, we foster the transition towards open formats and methods through the adoption of ebXML specifications for e-business. They form a largely agreed-upon family of standards, though not as widely supported in practice, and we made an effort to seamlessly integrate their registry/repository section in common environments including legacy systems. Interoperability also refers to data integration, thus semantics. We managed to introduce Semantic awareness in traditional document management systems, including an ebXML registry, with little influence on the existing scenario. Dealing with semantic tagging, we finally tackled the issue of a structured framework for document characterization.

This first chapter provides a gentle presentation of key concepts related to the enterprise world and its relationship with Information and Communication Technologies (ICT).

In the second chapter we begin the discussion on interoperability issues in modern companies, both from a syntactic and semantic point of view. The ebXML family of industry standards is introduced here.

The third chapter focuses on Semantic Knowledge Management, its benefits on the overall performance of an Enterprise Information System, and the available solutions for unobtrusive implementation of semantic tools.

1.1. Enterprise Information Systems

An Information System (IS) may be defined as the set of people, processes and technologies revolving around the manipulation of information. All activities concerned with the lifecycle of knowledge, including acquisition, creation, storage, transformation, distribution and display, are performed throughout an IS. The high-level goal of such a system is to deliver the right piece of information to the right person at the right time: a valuable service in the general case, a strict requirement in the world of business. The efficiency of this procedure, in fact, determines how fast and how well managers, and the company as a whole, can react to changes in the market and in the surrounding environment. A well-organized database benefits operational activities, too: it allows extensive sharing and reuse of data, documents, experiences and ideas, eventually cutting down costs and response

times in day-to-day processing. As a result, a high-quality IS represents a significant competitive advantage over other players, especially for companies having to do with rapidly evolving domains.

The term “technology” in the previous definition is used in a very broad sense, not necessarily related to informatics; though not as crucial as in recent years, the need for up-to-date knowledge has always existed within corporations, long before the advent of computers and software; for instance, telephones empower news delivery since the nineteenth century. However, in the present scenario, Information and Communication Technology (ICT) is an indispensable tool. The amount of today’s information sources is so huge it has become unmanageable for human beings. The high dynamicity of the global market requires constant monitoring, in a way that is not achievable without the support of automation. The same goes for the cataloguing and transformation tasks, not to mention the intricacies (and costs) of storing, retrieving and delivering paper documents or magnetic tapes. Furthermore, the leading productive sector worldwide is now the tertiary segment. For organizations focused on offering services, effectively managing data is a necessity rather than an advantage, as their very business revolves around that. In conclusion, information has become the most important asset for a company, and old analog methods are simply not good enough to handle all the current complexities.

1.1.1. A pyramid of systems

Within a modern enterprise, there are a large number of heterogeneous, often interdependent, Information Systems: from organization-wide systems for strategic decision support or customer management, down to very specific operational tools. Their stratification reflects the arrangement of business activities known as Anthony’s pyramid [1], a scheme which depicts the typical situation of a generic corporation. Anthony grouped these activities into three layers, each characterized by a different level of abstraction, different goals and different variables.

The company’s foundations are made up by *operational activities*, embracing both physical processes directly related to the creation of revenue, such as the production of goods, and other kinds of processes related to the fulfillment of basic needs, such as billing and HR management. Operational activities are aimed at the solution of straightforward problems: well-defined variables, quantitative data, and clear, short-term goals. They are *structured*, meaning that they involve decisions which can be easily reduced to a deterministic set of rules (an algorithm), and they get executed very frequently, so as to process large volumes of data [3]. As such, they are particularly well suited to automation, for both the relative ease of implementation and the far greater speed and precision automation allows as opposed to manual approaches.

At the top of the pyramid, the management is concerned with the company’s *strategy*, the most abstract and complex process of all. The development of this long-term plan requires sound analytic skills, in order to understand the existing situation, and the capacity of making accurate predictions about the future. Nonetheless, good predictions do not grant success, since there is no assurance that they will actually come true. Managers have to cope with high degrees of

uncertainty: fuzzy variables, qualitative data, unclear goals. A fundamental characteristic of high-level strategies is therefore flexibility, that is, the capacity to quickly adapt to changes, as a defense against the possible occurrence of unforeseen events. Strategic activities are highly *unstructured*: the complex nature of input data and the variability of problems and objectives make it impossible to define an algorithm to perform the decisional process. In fact, managers mainly handle exceptions, which are obviously out of any predictable process.

Tactical activities sit halfway between long-term, enterprise-wide strategies and simple day-to-day processes. They are intended for the realization of the mid-term scheduling that is required to put top management plans in practice, and usually involve *semi-structured* decisions, i.e. decisions with both programmable and non-programmable elements.

The pyramid metaphor, besides depicting the stratification of business tasks, also shows the gradual decrease in the number of functions from the bottom layer to the top. At the bottom of the pyramid, we can find a wide set of operational applications, while at the top only few enterprise-wide activities exist. This happens because, as we go up the different layers, processes get more and more abstract, thus integrating more and more concrete operations.

Sometimes this strict hierarchal arrangement is excessively rigid, and is not able to adequately support information flow in exceptional situations. When similar events occur, horizontal information systems may be created, through the creation of task forces and workgroups. Such organizations may also have a permanent nature.

1.1.2. Managing documents

Documents have always been the basic unit of information interchange within companies. Traditional management of paper documents worked well for years, but for the time being it is not sufficient anymore. There are two main reasons:

- Ever-increasing amount of information available
    - The volume of produced/acquired documents constantly grows
    - Management of physical documents has enormous costs for classification, storage, retrieval, duplication, transmission, disposal…
    - Paper archives grow quickly in size, it is not trivial to find enough space to accommodate them
- Ever-increasing need to access information
    - Documents are getting more varied, more and more professional roles need enterprise knowledge to perform their job
    - Need for more structured data, users often look for a piece of information rather than an entire document
    - Centralization of all knowledge in a single repository would facilitate both the management and the access to information

Despite the advent of ICT, documents are still at the heart of any IS. Of course, they are now in an *electronic* format, but this is often the only difference with their analog ancestors. Usually, electronic documents are a direct translation of their paper counterpart, with no additional metadata or elaboration whatsoever.

Information is scattered throughout their content and is not optimized for automated processing by software agents; in a word, they are an *unstructured* data source, as opposed to the structured information contained in conventional databases. The overwhelming amount of such documents frequently becomes a problem for companies, rather than an opportunity: a well-organized repository is thus a common need in the industry.

Document Management Systems (DMSs) are the standard solution to knowledge management requirements within enterprises. A typical DMS consists of a repository containing the actual documents and an engine working on top of it (Figure 1). The repository may be a simple as a disk partition, or as complex as a dedicated cluster of high-availability file servers equipped with load balancing and fail-over. It is never directly manipulated, the engine takes care of all interactions with external agents in order to maintain its internal coherence and integrity. The engine's core is a database, containing information about every document in the knowledge base: metadata, such as author, keywords and creation date, but also data for the management part, including access rights and physical location of the file.

Users normally invoke engine functionalities through a Web interface, or, less frequently, a stand-alone client; the engine may also offer public APIs to interact with custom software. At a bare minimum, a DMS offers functions for storing, searching and retrieving documents. Searching is the primary operation, and may involve metadata, content, or both; however, metadata are normally included in the search, as they are less tied to syntax and closer to semantics, and are thus more likely to capture the intent of the user. Advanced features include access control, versioning and tracking of document history.

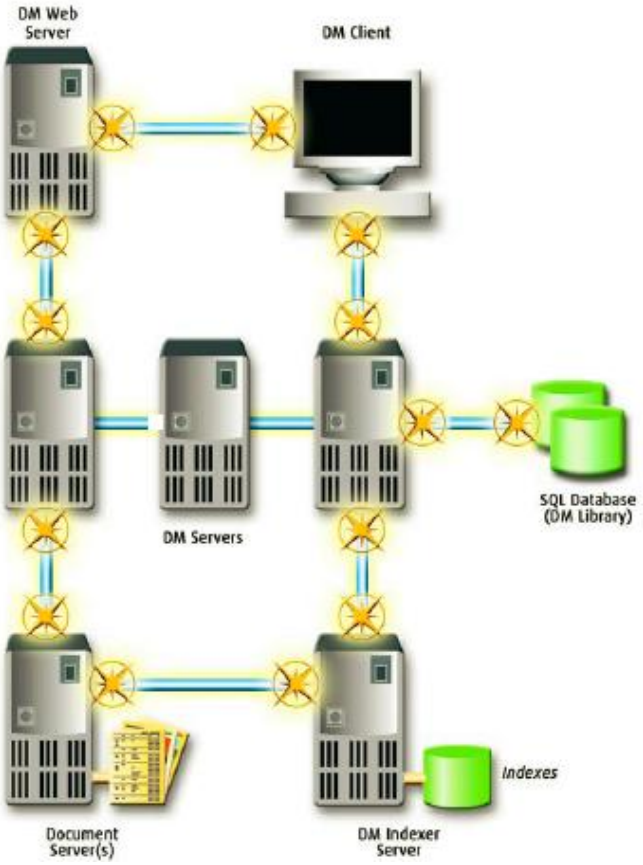

Figure 1: typical DMS architecture

1.2. Evolution of the IT department

1.2.1. The beginnings

The first general-purpose computer, ENIAC (Electronic Numerical Integrator And Computer), was developed during WWII as a top-secret project funded by the U.S. Army. Once completed, in 1946, it was mainly used to calculate artillery firing tables, and was also used in calculations for the H bomb. Military research has always benefited from the highest funds, so it is not surprising that a similar achievement was accomplished during a global conflict; new technologies may directly translate to an advantage over the enemy, and that is the reason why in war periods this kind of research is even more supported.

However, computers, like several other military technologies, quickly drew the attention of the industry: in 1947, only few months after ENIAC's completion, a leading food manufacturing company of the UK started to investigate the possible applications of computers to their every-day activity. In order to acquire expertise in this field, they sponsored the development of one of ENIAC's successors, ESDAC (Electronic Delay Storage Automatic Calculator), and they eventually managed to create the first commercially applied computer in 1951. Initially meant for internal use only, LEO (Lyons Electronic Office) models I, II and III met such a great success, that a separate firm was started to market the machine and manage the rising demand for computations by external companies; an early instance of what will later on be called *outsourcing*. From that moment, ICT began spreading in the industry and affecting every business operation, starting from the lowest layer of the activities' pyramid and slowly climbing up towards the strategic level.

1.2.2. Brief history of computers and enterprise software

Electronic Data Processing (EDP) systems initially dealt with the automation of back office applications, mainly financial ones; in fact, in most corporations the Data Processing department reported to the financial department. This operational layer includes systems like payroll calculation, general ledger, inventory management, and so forth. As we already observed, software systems at this level are useful to speed up manual processes and increase their accuracy. They boost productivity relieving humans from repetitive and error-prone tasks, and leaving them free to dedicate to more satisfactory activities. These operations were performed in batch mode, i.e. with no interaction, by mainframes, big and expensive computers particularly used by large institutions between the 1950s and the 1970s. By the early 1970s, many mainframes acquired interactive user interfaces and operated as timesharing computers, supporting hundreds of users simultaneously through "dumb" terminals.

Terminals were slowly replaced by personal computers: this migration caused a shakeout in the mainframe market around the1980s, due to the possibility to perform dedicated and decentralized computations. In this period "Office" systems and personal productivity software saw the light. Several forms of decision support systems appeared, including reporting tools, expert systems and business intelligence in general, although research in those fields dates back to the 1960s.

In the 1990s, the fall in the price of networking devices encouraged a re-centralization of computing resources and a renaissance of mainframes. Driving factors of this new market trend include the advent of e-business and the rapid expansion of emerging markets. As of today, the demand for services in the areas of banking, insurance or government are higher than ever. In this centralized environment, ERP (Enterprise Resource Planning) systems were born. An ERP is an integrated system that is meant to manage all corporate activities: accounting, logistics, inventory, production, human resources, customer relationships. Data is stored in a single repository and is independently accessed by the different modules.

It can be noted how enterprise software moved from operational to decisional concerns. Of course, this does not mean that operational ISs are not needed

anymore, but simply that “smarter” tools have slowly become available in addition to them. Moreover, the specialized, single-function (vertical) approach has been superseded by horizontal application mash-ups focused on the achievement of a goal rather than the execution of a procedure.

1.2.3. Present and future

“Enterprise 2.0” and “Enterprise 3.0” are among the most common buzzwords nowadays in the IT field, and represent the present and the future of Enterprise systems. They take inspiration from their Web counterparts: the Social Web, a.k.a. Web 2.0, and the Semantic Web, sometimes referred to as Web 3.0.

Enterprise 2.0 refers to the exploitation of typical Web 2.0 social computing tools (instant messaging, blogs, wikis, forums, folksonomies, social networks, mashups) within the business context, in order to improve productivity and efficiency. The term was introduced in 2006 by A.P. McAfee, a professor at Harvard Business School [44]. The key idea behind this new vision is the “prosumer” role (producer + consumer), borrowed from the Social Web. In contrast with the old approach of static contents decided by an author or an editorial committee, regular users of the knowledge base now have the possibility to produce information in a collaborative fashion. This is a small step for technologic infrastructure, but a giant leap for corporate attitude towards democracy: if this change of perspective was a revolution in the World Wide Web, it is even more significant in rigidly structured environments such as a corporations.

Actually, the implied transformation is so considerable that many companies hesitate to undergo the change. This is mostly for psychological reasons, even though some of the feared threats are real. After all, there are many open questions regarding these new systems:

- With such a large number of new applications at their disposal, employees may easily get distracted
- The impact on IT resources may be relevant (transfer of multimedia files)
- Confidential information may be exposed to the public
- Embarrassments may arise in managing interpersonal relationships

On the other hand, companies have many good reasons to move in the direction of social computing:

- If sharing documents and ideas is made easier and more pleasant, employees are more likely to actually do that. A highly accessible Information System filled with great volumes of shared data, is in turn an invaluable tool for knowledge reuse, reducing dead times and eventually improving efficiency
- Errors and gaps in the knowledge base get quickly fixed (self-healing content). Since this operation can be performed by anyone, the workload of administrators is much lower
- From a psychological point of view, “prosumers” feel more involved in enterprise activities, and achieve high levels of trust and sense of belonging.

Moreover, creating communities where *colleagues* become *friends* facilitates the flow of ideas and know-how.

Whether or not to invest in 2.0 solutions is a matter of trade-offs. However, it should be mentioned that several industry giants, including Shell Oil, Procter & Gamble and General Electric, are already hitting this road.

In the future, Enterprise systems are expected to absorb Semantic Web technologies within their knowledge management systems. Those new tools would serve a dual scope:

- Ease integration of heterogeneous data sources, so as to facilitate interoperation between systems and organizations, and increase the size of the searchable knowledge base
- Enhancement of information retrieval algorithms, in an effort to improve the quality of the results

This process has already started in some realities, and will probably continue in parallel to the adoption of Enterprise 2.0 solutions.

2. INTEROPERABILITY

The IT infrastructure in modern companies is a highly complex system with several heterogeneous components, including Document management systems, Workflow systems, Web portals and more. Coexistence of diverse sub-systems is a common scenario: integrated suites are too expensive for SMEs, and are frequently an overkill; even in bigger enterprises, the IT system is not always carefully planned from the beginning. Often, it undergoes a chaotic evolution, due to unforeseen changes in technology, in the market or in management directives.

2.1. Dealing with heterogeneous systems

Interoperability issues can be classified in two different categories: syntactic and semantic ones. By syntax, we mean data formats, interfaces and protocols, which may be (and usually are) incompatible with each other. At the lowest level, problems in the interpretation of exchanged messages may come from byte order or character set; however, potential troubles also lie in other aspects of communication, such as the representation of basic data types, the structure of the different messages or the expected message sequence within a protocol. Fortunately, there is a widespread solution: Web Services. Web Services are the de-facto standard for interoperability across heterogeneous software: entirely based on XML dialects, they expose system functionalities in a platform-independent manner. An XML descriptor, the WSDL file, accurately illustrates service location and available methods, including expected input and output parameters. Data types refer to the XML Schema specifications; they can also be composed to create custom complex types. Actual invocation is performed through standard SOAP messages (Simple Object Access Protocol), typically conveyed through HTTP.

Once a basic communication infrastructure is deployed, the next issue to be tackled naturally is how to perform a sensible information exchange, i.e. semantic interoperability. At a higher level of abstraction with respect to the previous task, we now focus on the ability to interpret data in a way that is meaningful for both parties: what is sent is the same as what is understood. This task is much more demanding than the other, and only partial solutions exist. One notable aspect of semantic interoperability is metadata integration of document management systems.

Metadata play a crucial role in the exploitation of functionalities exposed by a typical DMS. In order to be properly managed and to efficiently contribute to information delivery goals, each document must be properly characterized by specifying its coordinates within an adequately rich metadata space. In the archive management and digital libraries community, a standardization effort has led to the definition of a basic set of metadata to be dealt with for each stored/referenced document (Dublin Core); moreover, communication protocols for metadata harvesting have been built up within the Open Archive Initiative (OAI), taking into account Dublin Core metadata set. Despite this standardization attempt, Dublin Core has not been adopted by the large majority of DMSs, mainly due to its focus on informative documents, instead of business-related ones. It is worth noticing

that problems about semantic mapping among different metadata items (or towards a commonly established ontology) arise also in related application fields, e.g. cultural heritage digital libraries [19] and niches search engines [22]: in these contexts, it is often reasonable to employ a mediator scheme approach, possibly referring to Dublin Core as a common metadata set.

2.1.1. An industry standard: ebXML

ebXML (Electronic Business using eXtensible Markup Language) is a well-known family of XML-based industry standards for electronic business, including specifications for business processes, interoperable repositories and messaging. It is sponsored by OASIS (Organization for the Advancement of Structured Information Standards) and UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business), and has been ratified as the ISO-15000 standard in 2004. The development and the adoption of ebXML-based software is promoted by an open source initiative called freebXML. Within this initiative, the OMAR project is the reference implementation of the registry/repository specifications.

ebXML is emerging as the de-facto standard for e-business: for example, it has been proficiently used as the principal building block within information systems to support supply chain traceability [17, 18]. The specifications define, among other things, “an information system that securely manages any content type and the standardized metadata that describes it”: the ebXML Registry. Such registry thus enables the sharing of content and related metadata across organizational entities. Being the core structure for document exchange, the Registry is the key for interoperable business transactions. Unfortunately, only few companies adopted this relatively new standard, while the vast majority persists using traditional DMSs.

2.2. Interoperable DMSs

The race for e-business capability has hampered the adoption of one acknowledged standard solution for document management, thus yielding significant interoperability problems. Much (if not all) of the generated information is present inside one or more traditional DMSs. Each of them is implemented upon a different technology and follows a proprietary metadata model, leading to serious interoperability issues. Ideally, interoperability could be achieved moving all enterprise knowledge into an ebXML Registry. In practice, this is hardly ever feasible, due to the strong bindings between the DMS and the rest of the company information system.

2.2.1. An interface for interoperability

A common requirement for a DMS is the ability to easily integrate with external systems, in order to provide access to enterprise knowledge from a wide variety of platforms. In spite of this, DMSs typically support a small number of applications, and little or no effort is made towards generalized interoperability. As a partial solution, some systems provide APIs to enable administrators to code adapter applications.

Recently, the adoption of Service Oriented Architecture (SOA) contributed to modify this scenario: the latest versions of the most popular DMSs provide *support* for Web services to ease system integration and extension (FileNet, Documentum, Vignette). Web services run on the server side and wrap API calls to offer system access by means of standard Web technologies. Unfortunately, the claimed *support* often simply relates to a framework to help develop custom services: administrators still need to study the system's details and write the service code accordingly. This is a tedious and error-prone process, and it should be avoided as much as possible. Moreover, no standard way is defined for the implementation of such Web services, hence third-party software need to comply with different interfaces, depending on the actual DMS in use.

In order to achieve true independence from the actual DMS in use, we need to set up a general-purpose interface, able to accommodate typical needs for document management systems. Therefore, our prime concern is to outline a set of core operations that every DMS is required to support, leveraging the fact that typical operations performed over this kind of systems are fairly general. We restricted our choice to fundamental services, leaving out any platform-specific feature. This is an explicit design choice: most applications interface to a DMS in terms of simple queries (mainly document upload/download and metadata or content-based searches), whereas advanced features are rarely used and are not guaranteed to be available in all environments. We didn't take into account administration functions, such as user creation, role definition and so on, however this might be a significant functionality to add in future versions. We finally came out with the following list; since these operations are at the heart of document management itself, they are the most commonly used by end-users and the most widely supported by existing systems.

- Authentication - Obviously, some sort of authentication is needed to access the system. The simplest and most common way to identify a user is asking for a username/password pair. If successful, the login function returns a unique session identifier, which will be needed for every subsequent operation; this identifier encodes user rights and roles, thus allowing access control.
- Document/version creation - Creating a brand new document implies uploading the file and creating (and filling) a new entry inside the database to store its metadata. The creation of a new version for an existing document is almost the same process, but slightly more information is needed to identify parent document and version number.
- Document/version editing - Since file editing can't be performed in-place, it requires document download (check-out), local editing, and upload of the modified copy (check-in); this typically leads to the creation of a new version, but version replacement is possible as well. Metadata may be directly edited with ad-hoc functions, but it most often changes as part of a file updating process.
- Document/version deletion - Deleting one version determines the erasure of all its subversions; deleting an entire document results in the erasure of all its versions. In either case, the deletion includes both physical files and database entries.

- Metadata/fulltext search - Users rarely need a specific document, and hardly ever know the exact id; instead, they often look for documents talking about some topic, or containing a few given words. Hence, DMSs support searches over both metadata and file content.

Function	Description
`public string doLogin(string library, string user, string password)`	Login with the given user/password pair.
`public Profile[] search(string dst, string lib, string docnum, string author, string name, string type, string[] words)`	Perform full-text and metadata search. The *advanced* version allows a variable number of name/value pairs in input (the ProfileField structure array). Search criteria can be freely mixed.
`public Profile[] advancedSearch(string dst, string lib, string form, ProfileField[] sc, string[] words, string[] rp, ProfileField[] ord)`	
`public byte[] getDocument(string dst, string lib, string docnum, string ver, string subver)`	Document check-out.
`public string putDocument(string dst, string lib, string docname, string author, string typist, string type, string app, byte[] data)`	Document check-in. The first two functions create a new document. This implies the creation and filling of a new metadata entry, together with actual file transfer. The third function creates a new version, filling the comment and author fields in version metadata (almost a standard). The last one replaces an existing version.
`public string advancedPutDocument(string dst, string lib, string form, ProfileField[] fields, byte[] data)`	
`public void putVersion(string dst, string lib, string docnum, string ver, string author, string typist, string comment, byte[] data)`	
`public void replaceVersion(string dst, string lib, string docnum, string ver, string subver, byte[] data)`	
`public void deleteDocument(string dst, string lib, string docnum, bool delProf)`	Document deletion (including all versions) and version deletion (including all subversions).
`public void deleteVersion(string dst, string lib, string docnum, string ver, string subver)`	
`public void updateProfile(string dst, string lib, string docnum, string docname, string author, string typist, string type, string app)`	Metadata editing, and *advanced* version.
`public void advancedUpdateProfile(string dst, string lib, string form, string docnum, ProfileField[] fields)`	

Figure 2: Functions overview. The Profile structure represents a minimal subset of supported metadata (system specific). The ProfileField structure represents a name/value pair for arbitrary metadata fields.

2.2.2. Interface implementation

In order to make the DMS even more interoperable, the abstract interface can be implemented as a Web Service. The Web service extension integrates DMS interfaces and renders it platform- and technology independent. Figure 2 shows actual function signatures corresponding to interface methods. The overall system basically takes advantage of DMS-specific APIs, combines them into logically distinct functions, and exposes them as Web services. In this perspective, each specific DMS requires its own specific wrapper software. Figure 3 shows the architecture of a traditional DMS, composed with our extension to obtain an interoperable DMS. As it can be seen, our system acts as an *additional* access point to the DMS, leaving the original system intact. This kind of enrichment in the access point number let clients free to keep on working through the native interface, whenever it is either mandatory or convenient. Clients using the new interface, instead, will be able to communicate with the DMS via SOAP messages, regardless of technology issues and implementation details.

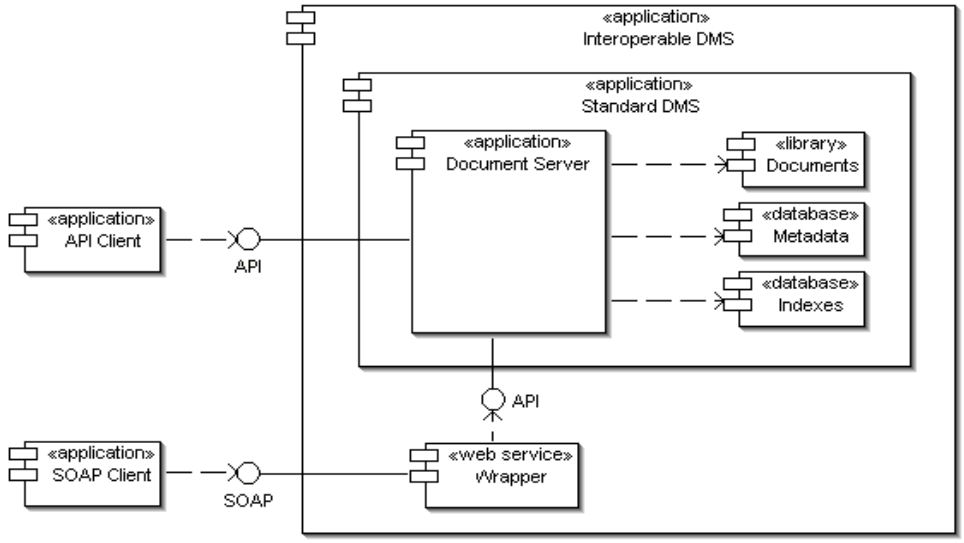

Figure 3: interoperable DMS

2.2.3. Metadata management

With no common set of metadata to be taken as reference, we can think of explicitly working with different metadata sets in a coordinated way. The coordination mechanism, according to the SOA approach, has to be implemented in the software layer that accesses the single services at the different DMS interfaces. This approach to metadata management deeply affects the interface structure of any service involved with metadata. In fact, the service signature must be general enough to allow passing a list of name/value pairs to describe multiple metadata items. The service has to be implemented so as to parse the list and to behave accordingly to what pairs are actually meaningful on the specific DMS

platform. In our implementation (Figure 2) we applied a slightly different variant of this approach, choosing to develop each service function in two different flavors: with a static signature, and with a variable set of parameters. The first one is useful to access system-specific metadata, whereas the other can manage any kind of metadata, passed in as name/value pairs.

2.2.4. A common architecture

Once all involved DM systems are empowered with an interoperable interface, they can be proficiently connected to a standard ebXML registry, so that ebXML-compliant clients could access the information within them. The main objective of this system is to promote a gradual adoption of the ebXML Registry specification; in fact, the proposed architecture takes advantage of the power and flexibility of ebXML while leaving in-place systems unchanged. The original DMSs are coupled with an ebXML Registry, used to mirror their metadata and manage all metadata-related operations, thus overcoming the typical restrictions of the back-end legacy module. An additional distinct component can coordinate the access to the underlying systems, and enforce metadata consistency. A direct access to the original DMS is performed only in case an actual document would be involved in the query.

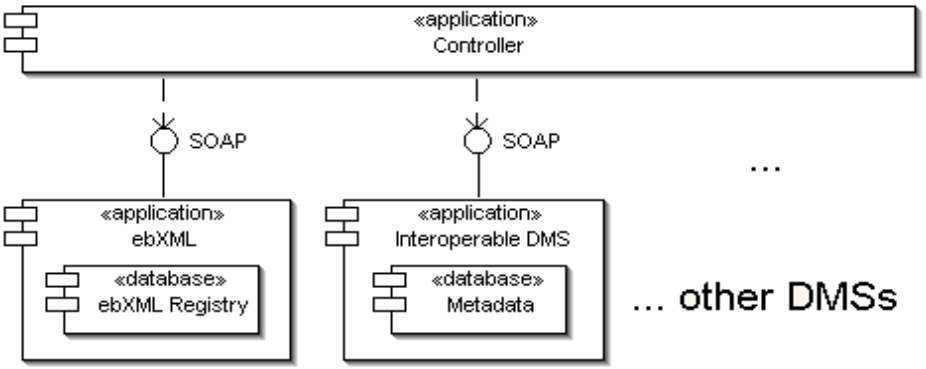

Figure 4: overall architecture

According to our architecture, newly installed and in-place components are arranged in three sub-systems (Figure 4):

- A legacy DMS, containing both documents and related metadata, with the added value of our interoperability component. In the general case, there could be many different systems.
- An ebXML Registry, used to store a copy of DMS metadata and provide advanced management features over legacy metadata.
- A controller application, intended to coordinate access to the above-mentioned systems.

In order to maintain the independence of each individual component, every interaction is mediated by the controller: as far as the single sub-system is concerned, no knowledge about the external world is required. It is up to the controller to compose the simple interaction functions provided by each interface into a globally meaningful and consistent operation. In particular, such software level should provide also a semantic mapping among different metadata items on distinct DMS, or maybe a mapping towards some kind of commonly established ontology.

2.2.5. Related work

SoDOCM [23] is an example of federated information system for document management. The declared goal of the project is to provide an application that can transparently manage multiple heterogeneous DMSs, while retaining their autonomy. It is based on the mediator architecture, a well-known concept developed for database interoperability, and follows the service-oriented computing paradigm.

With the AquaLogic Data Services Platform [24] the authors propose a declarative approach for accessing data, as opposed to the standard procedural way. The system is intended to integrate heterogeneous data sources in order to support composite applications. This is achieved using XML and XQuery technologies, and introducing the concept of data service. A core functionality is the automatic translation between XQuery and various SQL dialects.

LEBONED [25] is a metadata architecture that allows the integration of external knowledge sources into a Learning Management System. The example presented in the paper operates with the eVerlage digital library, which provides a Web service interface to import documents and related metadata.

OAI compliance and metadata re-modeling are a central issue of the eBizSearch engine [22]. In this paper, the authors describe how they managed to enable OAI access to the CiteSeer digital library; the proposed solution consists in mirroring the original library and extending this external database to meet OAI specifications. Synchronization between the two systems is performed periodically.

2.3. Other applications

As an evidence of the interest companies actually have in integration and interoperability, we now present two real-world examples of such systems. The first one is an integrated control panel for the management of access rights to diverse applications within the Enterprise Portal [20]. The second one is a metacrawler, i.e. a search engine that queries multiple search engines at a time [21]. Both were developed during the Ph.D. course and are currently in use, or will be in the near future, in a well-known multinational for their daily operations.

2.3.1. Managing access control

Being strictly hierarchical entities, corporations greatly stress the importance of security. It is critical to clearly state *Who knows what* and *Who does what*: administrators need to outline detailed access rights for each user and for each sub-system. Each employee has well-defined tasks, and his access rights vary from application to application. Of course, one must be able to get his job done; however, having too much power, i.e. benefiting from unnecessary permissions, may lead to serious security threats. Users must be able to reach the information they need, and be prevented from accessing restricted material This need for accurate tuning of access rights clashes with the presence of many diverse systems to be managed: many permission schemes, many possible configurations, many access points. There is no simple way to have a comprehensive view of a user's current permission set, and no standard approach to modify it.

Figure 5: managing permissions

Our solution to the problem is a Web-based control panel for permission management. Its main feature is user search: the administrator can filter employees down through several search parameters, including id, name and department. He can also search by access rights, i.e. search for employees having or not having a given permission. The resulting grid lists all personal details of retrieved users, which may turn useful to accurately spot the interesting ones. When one is selected, a panel containing current permissions is dynamically loaded (Figure 5). A row is shown for every available access right, and a checkbox quickly indicate whether the user has it or not. The administrator can now grant or revoke rights clicking on checkboxes: every change is automatically logged so as to trace modifications and roll them back when necessary. This tool also includes two additional utilities: reporting and graphs, and alerting. The former gives an immediate insight of accesses made by users during a given time span. Administrators can take advantage of such statistics to decide whether or not a user needs given permission for his work. The latter refers to the need to quickly act in response of a new hiring or firing. In the first case, an entirely new set of rights has to be inserted in the IS; in the other case, old permissions have to be cleared in order to prevent security leaks. In both situations, administrators get informed by alerts, which in turn redirect to the management of the concerned user; alerts are created by a dedicated job, running once a day, that queries the HR database and registers all changes in its records.

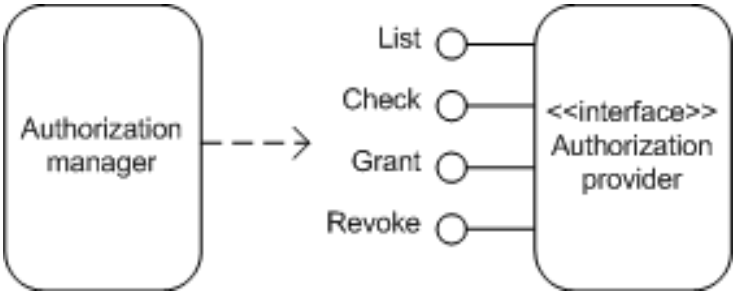

Figure 6: generic interface for authorization management

The core of this system is a pluggable architecture that supports generic authorization systems conforming to a given interface. We identified four basic operations supported by any system that offers access control: granting a right, revoking it, checking if a given user has the right and listing all users with the given right (Figure 6). The latter is not strictly necessary for permission management, but it is commonly available and turns out useful for filtering user searches. This interface may be variously implemented to wrap calls to APIs of legacy systems. In practice, we derived two different implementations. One is tied to groups in the DMS: each group of users has statically defined permissions with respect to each document type; controlling who belongs to which group thus translates to controlling who has access to which documents. Each group is treated as a single right and is managed through Web Service methods, which were developed as an add-on to the “interoperable DMS” interface described in previous paragraphs. The

other implementation is much more generic, and involves direct database queries. It requires the details of a database connection and the names of four stored procedures, one for each method. It is currently exploited for enterprise portal access, but it is clearly simple to reuse the same mechanism in other scenarios. It's worth noting that this architecture is extendable in a very straight-forward manner, only using the web application's configuration file: at start-up, the application automatically reads the list of authorization providers, their types and parameters, and makes use of them throughout its lifecycle without the need of writing a single line of code. Of course programming would be necessary when considering new kinds of providers, apart from the two we implemented, but even this task is made easy by the hierarchical structure of existing objects and the overall architecture of the system.

2.3.2. Metacrawling

This project is part of the Enterprise 2.0 framework which is under development for the evolution of the corporate IS. The framework as a whole is designed to support multiple forms of integration and collaboration: wikis, forums, VOIP, instant messaging and sharing of documents, calendar events and knowledge in general. In particular, it should work as a link among DMSs, the ERP system and final users. To better take advantage of the numerous benefits of this new approach, an adequate UI is required. All complexity should be hidden away from users, and the learning curve of the new tool should be as flat as possible. Moreover, it should gracefully handle the conflicting requirements of being both ubiquitous and unobtrusive: ubiquity, together with ease of use, enables users to get quickly accustomed to the new system, and lets them become active part of the "prosumer" community; on the other hand, users should be granted the freedom to ignore the component and perform their work as they used to. This last constraint is particularly important during the first period after system introduction, when employees have no knowledge of the added value it provides, and see the novelty with suspicion, or worse, antagonism. The more it gets in their way when they try to carry out an operation without its support, the less likely they are to use it when they would actually need it. Negative feelings such as mistrust and hostility are obviously the worst possible result when talking about a social framework intended to ease interpersonal communication and sharing of ideas.

Our solution consists of an unobtrusive widget that gets loaded at system start-up and runs in the background only showing a minimized taskbar icon (we are talking of Windows platforms). When invoked by the user, a small input mask is shown, with two simple fields: the query and the context. For context, we mean the set of systems which will be queried with the given search string: possible choices naturally include DMSs, but also other kinds of systems. For each of them, a plug-in is needed as a middle layer between our component and the specific knowledge repository. Plug-ins handle all intricacies of legacy systems, and communicate with the widget through a clean and standardized interface. The first plug-in to be developed obviously was the one for querying the DMS through its Web Services. However, the second one has an interface towards SAP, a widespread commercial ERP system, and more are expected in the future; for instance, one explicit demand was to upgrade the human resources database to store the know-how of employees and thus be able to ask questions such as "who should I contact to

solve this problem?". Plug-ins may also support advanced search features, such as filetype filtering. Special tokens included in the query text may handle those special requests; less capable engines should be able to ignore advanced tokens and perform a classic search.

3. SEMANTIC KNOWLEDGE MANAGEMENT

The current research trend, when it comes to data quality, information retrieval and database integration, is the semantic one. Driven by the Semantic Web hype, investigation over this topic set off in 2001 [4] and is now being carried out in universities and research labs all over the world [6,7, 26,28,30].

3.1. Syntax and Semantics

At present, the approach to information processing is mostly based on syntactical rules. A search engine of this kind can determine, for instance, that a document containing the word “trees” is relevant to the user even if the original query mentioned the singular form “tree”, or vice-versa. It may also handle more complex forms of word declinations, such as verb tenses, provided that a suitable vocabulary is made available. While this may suffice for some applications, namely those specialized on a very limited domain, the same does not hold for more generic systems. Natural language is inherently various and ambiguous: straight-forward syntactical procedures are not flexible enough to cope with homonymy, synonymy, typos and all the sorts of “semantic noise” one may find in a real knowledge base. Take the example of homonymy: out of context, it is impossible to know whether the word “keyboard” refers to the musical instrument or to the input device for computers. From the perspective of information retrieval, this results in poor precision, since a search engine would return documents having to do with both senses of the word, regardless of what the user actually meant. Problems arise also in the field of automatic organization of data. When trying to infer the structure of the knowledge base, spurious connections would be found between unrelated documents; fully automatic organization is thus unfeasible. For the same reasons, integration of heterogeneous knowledge bases is hard, at best, and has to be performed with largely manual methodologies. Again, the merging agent has to compare objects only through syntax, and determine whether they are equivalent, or refer to the same topic, or not. “Semantic noise” can be found even in small, closed databases; not surprisingly, the situation gets worse when we extend our view to external sources.

All the above problems can be overcome exploiting semantics. The key idea is to associate information units to concepts, which uniquely identify a semantic area within the application domain. For instance, there is not a single “keyboard” concept, because the meaning of the word is ambiguous. There are, instead, two distinct concepts: “keyboard (music)” and “keyboard (informatics)”. However, ambiguity may exist in the general case but not necessarily in the specific application: if the knowledge base is known to contain information about computers, the right sense for “keyboard” is immediately clear. Synonyms, syntax errors, alternative spellings and even translations in different languages are handled gracefully by this approach: all the words sharing a common underlying idea will simply be associated to the same concept. A remarkable feature of concepts is that they are interlinked by a dense network of relations, such as “A is a subclass of B”, “A is an antonym of B” or “A lives in B”. Obviously, the complete graph is hard (when not impossible) to obtain, unless the focus is restricted to a narrow range of

interesting relation types; the most common one is subsumption, i.e. hyponym-hypernym or “is-a”, which allows the definition of hierarchies.

This web of relations can be exploited so as to reveal implicit connections between information units related to different concepts, and thus influence the outcome of search engines and data integration tools. Obviously, the structure of the knowledge base is directly inferred from the structure of related concepts in the ontology. Thanks to the semantic approach, information retrieval algorithms can experience a boost in both precision and recall. The search engine spider can easily determine whether a document is connected to the user query or not. Finally, database integration is made much easier, as it now implies the matching of ontology concepts instead of the matching of every possible couple of documents. Not to mention, the chance given by OWL (see next paragraph) to share a common ontology in an interoperable format; when both knowledge bases refer to the same classification model, integration is automatically achieved without further processing. The same holds when one ontology is asserted to be a specialization, or an addition, to the other: all is needed is the exact merging point, then reasoning over the compound model can happen just like it used to be with a single ontology. Performing this kind of reasoning is not trivial, though, and requires a different strategy for each supported relation.

3.2. Improving information repositories

In order to support semantic-aware tools, the knowledge base needs to be upgraded in two successive steps: first, create or obtain a formal model of the target domain; second, proceed with semantic tagging, i.e. associate concepts (entities belonging to the model) to objects in the knowledge base. Usually, the model is described in an ontology. Ontologies are a generalization of hierarchy trees: nodes can have multiple parents, and they can also be connected by relations other than subsumption. Nodes may have properties (name-value pairs), possibly with restrictions on the set of acceptable values, and they can in turn be arranged in hierarchies, too. Individuals, instances of a concept/node, can be included in the graph, and are identified by a specific combination of values for their properties.

Ontologies are expensive to create and maintain, and can quickly grow to an unmanageable size. For example, the bioinformatics community is particularly active in this aspect, as the extremely large knowledge bases in this research field impose strong requirements on information interchange capabilities [32]. Hence, semi-automatic and fully automatic approaches have been proposed for their creation out of a corpus of related resources; nevertheless, human intervention is usually required when graph quality is a primary concern. There are relevant efforts towards the (manual) definition of extensive dictionaries, thesauri and ontologies for general and domain-specific knowledge [36,31,35].

OWL (Web Ontology Language) currently is, by far, the most frequently used language for defining and instantiating ontologies [5]. As the name suggests, it was originally developed as a basic building block for the Semantic Web [4], but quickly became a de-facto standard in several other application areas. OWL is a markup

language based on XML: ontologies expressed in OWL are thus easily interchangeable text documents, intended to be both machine processable and (almost) human readable. As such, a small-sized ontology can be written by hand, although tools exist to facilitate the definition of more complex domains.

An ontology is just a static structure used to store information; software systems called *reasoners* are employed to fully exploit this knowledge [38]. A reasoner can infer facts not explicitly stated in the ontology, but entailed by its semantics. For example, if *A* is a subclass of *B*, and *B* is a subclass of *C* (and knowing that *subclass of* is a transitive relation), a reasoner can deduce that *A* is also a subclass of *C*, although it was not explicitly declared. Moreover, reasoners are commonly used to uncover potential contradictions among assertions and for property-based object classification.

Once the model is complete, enclosed concepts can be pointed to by elements in the knowledge base: these associations are created in the tagging process. Tags may be attached to a resource as a whole, or only to a part of it. For instance, it can be stated that a given textual document concerns a particular topic, but, at a finer granularity, only a couple of paragraphs actually do. The usefulness of such distinction totally depends on the application. Tags can be embedded directly in the knowledge base (RDFa and MicroFormats are examples regarding XHTML documents), or be stored in a separate database, normally as triples in the form subject-predicate-object.

Given the relatively recent appearance of semantic technologies, the vast majority of resources on the Internet and in closed information systems is not semantically tagged; even considering only newly created information, difficulties arise because of the huge numbers involved in the process. For these reasons automatic tagging has gained great popularity in the scientific community, and several algorithms have been developed to perform the task with little to none human support. They usually leverage natural language processing (NLP) and other techniques from the artificial intelligence research field, in order to understand the context in which words are used and consequently disambiguate unclear situations. However, complete automation is not available yet, as it requires absolute accuracy in identifying entities inside source documents. All existing annotation systems are semi-automatic, and rely on human intervention at some point. Input data is normally text: when dealing with non-textual resources (audio, video, images), the most frequent approach is to exploit attached metadata rather than the actual content.

For instance, Amilcare [34] is an adaptive information extraction tool. It uses supervised machine learning techniques and requires a corpus of manually annotated documents for training. It treats the semantic annotations as a flat set of labels, thus ignoring knowledge from the ontology. Amilcare is the basic component of several annotation platforms, including OntoMat [34], Armadillo [29] and MnM [39]. The KIM system [46] produces annotations linked both to the ontological class and to the exact individual in the instance base. This dual mapping allows the information IE process to be improved by proposing disambiguation clues based on attributes and relations. Ontology structure is

exploited during both pattern matching and instance disambiguation. SemTag [27] is the semantic annotation component of the Seeker platform, for performing large-scale annotation of web pages. It uses TAP, a shallow taxonomy covering popular items such as music, movies, sports and so forth. SemTag also has a disambiguation phase, where a vector-space model is used to assign the correct ontological class.

3.3. Bridging the gap

As the analyst Andrew White pointed out in his corporate blog (posted on April 30th 2009), the "semantic revolution" is yet to come in the industry: "For too long *semantic web* has focused on how data moves across the Internet. […] The level of investment and thinking applied to *inside enterprises* or *B2B* compared to B2C or the social side of the web is much, much lower". In the business context, semantic technologies are widely considered "still on the rise", highly immature and overestimated. Early adopters may well take advantage of some of their features, but full exploitation of their potential and mainstream adoption are still a few years away. Nonetheless, there is little doubt that Semantics will eventually be included in ordinary information systems and transform the way information is created, searched and consumed throughout companies. The trigger for large-scale adoption of these technologies may lie in the ability to seamlessly integrate them with existing applications, as opposed to forcing the migration to a brand new knowledge repository.

Attempts to introduce semantic-aware tools in an enterprise information system are subject to several technical and non-technical impediments. First of all, while the expense to purchase/develop such systems is definite and immediate, the return of this investment is not as evident to the average manager. The advantages of a semantic model are still misunderstood and underestimated. This is changing, however: at the New York Times, articles are regularly tagged with entities taken from complex thesauri; this knowledge base is now available to the public, too. Another example is the recent agreement between Oracle and Thomson Reuters, which states the introduction of OpenCalais support (an automatic, NLP-based, metadata generation service) into Oracle's 11g database. But even though the mentality is changing, another big issue with the adoption of most systems presented in academic papers remains: they assume full freedom in design and implementation, while in a real enterprise strong constraints are dictated by the presence of older systems. The top management is usually inertial to change, for both monetary and psychological reasons, and obviously prefers small adjustments over extensive transformations. Migration of all knowledge from one repository to another is rarely an option. Therefore, we studied two low-impact approaches to the introduction of semantic-aware computing into traditional Document/Content Management systems, the kind of systems that may be currently found in real-world companies.

3.4. From keywords to tags

With the advent of Web 2.0 technologies, tagging systems have become one of the most common forms of collaboration over the Internet: popular websites, mostly belonging to the "social" part of the Web, let users associate single words or short

sentences, i.e. tags, to pieces of information (photos, videos, URLs…), in order to describe them and consequently ease future searches. Depending on the specific system, users may be allowed to only tag their own resources, or their friends' resources, or anyone's. When users are allowed to tag other people's resources, spontaneous "bottom-up" classifications emerge, known as folksonomies (folk + taxonomy); this kind of classification supersedes the traditional "top-down" approach, where a designer defines a rigid hierarchy of categories once and for all. Tags are similar to keywords, but while keywords are static entities defined by the author(s) when creating the resource, tags change dynamically and may be attached to a resource anytime by anyone. Also, tags typically have a weight with respect to a given resource, expressing how many times that tag was selected to describe that resource; these weights can be graphically represented by means of "tag clouds", and are subject to change in time, too. Generally the system keeps track of who added a tag, which tag was that, and which resource was involved; as a consequence, a rich network of tags, users and resources exists, which highlights interesting relations such as people using similar tags, or words chosen to describe a particular resource.

In the original scheme, tags are unconstrained; the freedom accorded to users in choosing these words may or may not be a problem depending on the context of the specific application: on the one hand, it helps the emergence of a categorization which closely reflects the conceptual model of the users; on the other hand, it allows the introduction of the previously mentioned "semantic noise". Attempts to cope with this shortcoming usually move in the direction of mixing the top-down and the bottom-up methods, trying to make the most out of each paradigm. The general idea is to give structure to the tag model, while preserving some of its original flexibility. One possible solution is to restrict the set of available tags to those contained in a controlled vocabulary of some sort; when it stems from a collaboration of domain experts and end-users, it is referred to as a "collabulary". An enhancement to this approach is to substitute the vocabulary with a full-blown ontology, so that the additional information contained in the tag domain model could be exploited in the search phase. Sometimes called "folktologies" (folk + ontology), these classifications represent the highest peak in the integration of apparently conflicting philosophies.

In the industry, precision is obviously preferred over democracy. Companies are highly hierarchical entities with little interest in the distribution of decisional power and great interest in proper categorizations. Hence, in our study we disregarded unsupervised folksonomies and focused on the folktology approach. Although ontologies referred by folktologies are usually supposed to be editable by users (with some constraints; for instance, concepts can be added but never deleted, and statements cannot be included when they contradict existing axioms), we did not investigate a collaboration environment for doing so. Given its complexity, such a system can hardly be considered a low-impact modification. Therefore, new versions of the ontology cannot be created incrementally and require the upload of the entire model; nevertheless, a standard versioning mechanism can be exploited to trace the relations between the different files.

3.5. Introducing semantic tagging in enterprise systems

The adaptations we are about to introduce affect the heart of any company's information system, i.e. Content/Document Management Systems. Due to the amount and worth of informative assets in the current scenario, no modern enterprise can do without the features they offer: metadata support, versioning, access control, metadata- and full-text-based search, at the bare minimum. Some provide knowledge categorization through the use of tags, other less recent solutions work with folders. Semantic tagging can be unobtrusively introduced in both sorts of systems: the upgrade essentially revolves around constraining the classifications available to users to the set of entities enclosed in one or more ontologies. The main difference, when dealing with folders, is that multiple categorizations for a single object are not natively supported; still, they can be expressed by replicating the object (or references to the object) under each concerned folder. We can thus refer to tags only without loss of generality. Either way, the fundamental concern is the introduction of structure in the classification model so as to mirror that of a selected ontology. Tags in the original system may be totally free, or they may be bound to a tree hierarchy. In the following, we show how both situations can be managed: we considered Alfresco ECM as a representative of the first class, and the reference implementation of ebXML registries for the second one.

3.6. Working with unconstrained tags

Alfresco is a widespread open source solution for both Document Management and Web Content Management. It is based on Java technologies, and supports tagging, although only in its simplest form; ontology-related concepts, instead, are completely ignored. In Alfresco, documents are organized in "spaces". A space is similar to a folder, but with some "smart" facets: most notably, we can add rules to manage content being added to, edited in, or removed from the folder. One of the actions a rule may trigger is the activation of a functionality, or "aspect", for the selected documents, such as "versionable", "taggable" or "classifiable". These features can be exploited to set up a simple semantic-aware environment.

As a preliminary step, we prepared two spaces to contain the documents involved in the classification task. First of all, necessary ontologies, either in OWL or RDF format (Resource Description Format, the ancestor of OWL), must be uploaded to the repository. During this process, ontologies are treated just like any other file, and no particular action needs to be performed. This upload is not strictly necessary: in principle ontologies might be downloaded on-the-fly from a user-provided URL, or may even be remotely navigated. A second space is reserved for contents subject to classification. A rule is associated to it: every document uploaded into that space will be "taggable" (i.e., will be extended with the "taggable" aspect). Again, this space is not essential, and it is there only for a practical reason: it ensures us that tagging is active for those documents.

Being a web-based tool, the user interface heavily relies on Javascript; we adapted the behavior of the tagging module so as to accommodate ontology-aware tagging. In order to keep this a low-impact modification, we chose a Javascript-only

approach to both ontology processing and visualization. The starting point to add a semantic tag is the same as for traditional tagging: in the property view for a “taggable” document, an “Add a tag” link can be seen (Figure 7).

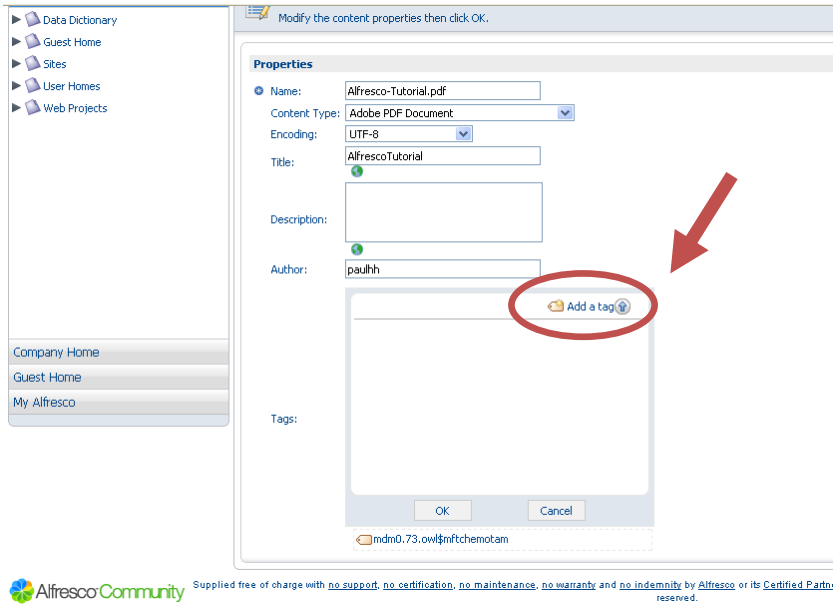

Figure 7: setting up document metadata

When clicked, the link reveals a new navigation interface, which allows the selection of the reference ontology by means of a combo-box, filled with the contents of the ontology folder (Figure 8). This interface may be changed so as to allow retrieval of remote files too.

Once loaded, a graphical representation of the ontology is shown, which can be interactively explored for an easy identification of interesting concepts. We made use of a force-directed graph layout for animated visualization, where nodes are ontology concepts and links between nodes represent a parent-child relation (Figure 9). Navigation is performed by clicking on nodes, which reveals direct children and at the same time selects the node as a candidate tag, changing the text on the “Add tag” button. This kind of representation lacks some expressivity when applied to ontologies; other graphic layouts may be tested for a better user experience: trees, hyperbolic trees, radial graphs and more [40,41]. Finally, when the user has found the entity/concept of interest, he can add the entity name to the document’s tag set. In order to avoid ambiguities, the resulting tag will include both the name of the entity and the name of the ontology.

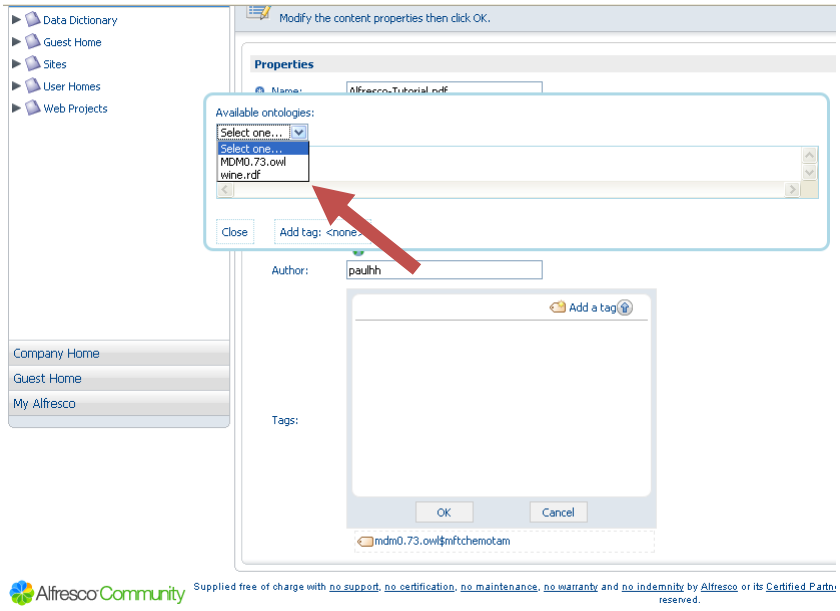

Figure 8: selection of the reference ontology

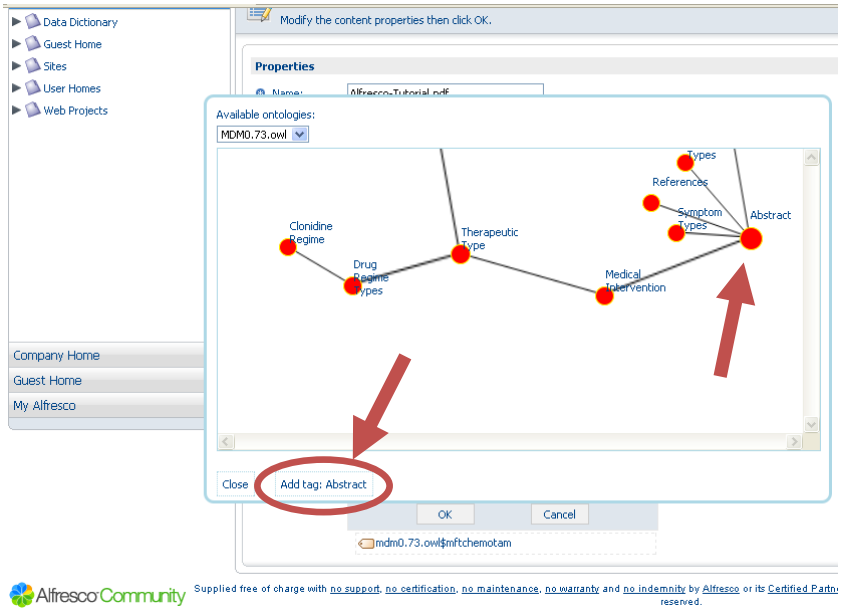

Figure 9: navigating the ontology and selecting the tag

3.6.1. A case study

The application of the main ideas underpinning our approach to semantic tagging in the ECM field has been exploited also in a specific case study. Diagnostic procedures in molecular biology tests are carried out by a number of different steps, using specific analysis tools in the subsequent phases. The final outcome of the test is usually sketched out in a standard, rigidly structured format. Conversely, the whole path to the test outcome throughout the different steps encompasses intermediate documents that describe results from the employed machinery. Such intermediate information, given as files in both textual and proprietary formats, is related to the test outcome plainly by file naming conventions stated by the laboratory staff. In order fully exploit the knowledge base in this large amount of data, it is crucial to properly apply a standard classification of all the related documents, as well as addressing lineage issues in a more systematic way.

We have thus proposed [43] to proceed with a semantic tagging of most of the intermediate files (to be kept in a document repository) making use of well-known ontologies in the life sciences field, such as GO (gene ontology), OBO or Biopax (see Appendix A for references). The semantic tagging procedures have been introduced according to the aforementioned minimum-impedance approach, hiding the burden of ontology navigation by means of simple, user-friendly interfaces (developed in Javascript) which do not severely interfere with the underlying ECM environment.

The results are encouraging, both in the common diagnostic procedures and in the research field. In particular, in this last case the semantic approach let us overcome the inherent unpredictable usage of each intermediate result within the whole framework of the experiment development.

3.7. Adapting ontologies to trees

Sometimes the CMS/DMS already supports structured categories, but their structure is limited to traditional trees. This can be a problem, because trees are not sufficiently expressive to represent ontologies in the general case: even focusing on parent-child relations only, entities can have multiple parents, while tree nodes do not. However, one can get around this limitation, while still leveraging the native support to tag structure, with the usual “cheat” of repeating concepts under each of their parents. Alternatively, trees can be ignored and a parallel tag model may be created to reflect ontologies more closely.

We developed this solution on OMAR, the open source reference implementation of the ebXML registry specifications. Sponsored by OASIS and UN/CEFACT, ebXML is a well-known family of industry standards for electronic business, including specifications for business processes, interoperable repositories and messaging. The most recent edition of the registry/repository standard is version 3.0. Since then, a new working group (the Semantic Content Management Sub-Committee - SCMSC) has been established under the Registry Technical Committee to investigate use cases and requirements for semantic content management, and produce specifications to be introduced in the next major

revision of the ebXML Registry. At the time of this writing, version 4.0 is still under development.

The current version allows hierarchical category trees and multiple classifications for a single object. It does not explicitly mention ontologies nor tagging, except for a deliverable from the SCMSC suggesting patterns and workarounds for the translation of OWL constructs into Registry objects. However, its features can be easily extended through slots and content management services. Slots are generic name-type-value triples which can be attached to objects in order to enhance their metadata. Content management services are custom services that can be used to perform content-based transformations on new documents before submission to the repository. They are associated to one (or more) object type(s), and are invoked upon a publishing operation that involves one of those associated objects. Users can provide their own services (either as an XSLT stylesheet or as a stand-alone process, accessible through the Web Service technology) and accommodate them in the Registry through its pluggable architecture. To set up this functionality, a reference to the service must be first created into the registry, and then associated to the classification node representing the target object type. Standard service types include content validators and content catalogers: validators check the conformance of documents with respect to a given rule-set, and prevent the publication of invalid ones; catalogers extract metadata from document content and create the related objects inside the registry. This latter mechanism can be proficiently used to automatically perform the translation from OWL to the Registry Information Model (RIM) recommended in the above-mentioned paper from the SCMSC.

3.7.1. Mapping OWL to RIM constructs

The first step for the semantic enhancement of OMAR was the creation of a few Registry objects to mirror the functionality of some OWL constructs that do not have an equivalent in the RIM. They include, among others, structures for the definition of properties, hierarchy of properties and hierarchy of classes. Although we will only be using class hierarchy information, we took into consideration the whole set of constructs for the sake of completeness; this way, we cleared the ground for any future work regarding ontology management on ebXML. We also created a new object type specifically for OWL ontologies, in order to distinguish them from other kinds of content; this will turn out useful when figuring out whether a document needs to be processed by the cataloger or not. Finally, we implemented the discovery facility, i.e. a set of queries for the navigation of the model: find all parents of a class, find all children, find all transitive properties, etcetera This involved some SQL to query the database underlying the Registry, and, since recursion was required, is implementation-dependent.

In the ebXML framework, documents are stored inside the repository, and represented in the registry by their associated metadata. A registry object holding metadata for a repository item is called *extrinsic object*. Moreover, other types of objects can be stored in the registry. The ebXML Registry Information Model (RIM) lists quite a few subclasses of the *registry object* class, such as *classification scheme*, *person,* and *notification*.

The specifications regarding OWL support [15] clearly define one possible mapping scheme to represent OWL concepts in terms of ebXML RIM constructs. An essential requirement for such an approach is the preliminary creation of *basic structures*, including *association types* (for hierarchy and property definition), *external links* (referencing XML Schema datatypes) and *ad-hoc queries* (for elementary ontology browsing). For instance, before we can say that *P* is a property of class *C*, we have to state the existence of the *hasProperty* association type.

Once the basic structures have been set, any OWL document can be represented inside the registry according to the following rules:

- An ontology is translated to a *classification scheme*. It represents a taxonomy tree, whose nodes are called *classification nodes*.
- Each entity corresponds to a classification node within the scheme. Since OWL allows multiple inheritance while ebXML does not, the *subclassOf* association type is introduced to override the built-in *parent* property, normally used to define hierarchies. Given that th ebXML registry does not natively support the "subclassOf" association, nodes are organized in a flat arrangement and are all direct children of the classification scheme.
- An individual is represented by an extrinsic object. Its type is specified by means of a *classification*, i.e. an association with a classification node. An extrinsic object can be classified according to several different schemes, and consequently may present many classifications.
- Properties become associations. In particular, property *characteristics* (such as transitivity, symmetry, etc.) are accounted for as *association types*.

Class properties are further examples of associations, linking classification nodes to whatever is in the property range: other classification nodes or any XML Schema datatype. Properties may be organized in hierarchies, too, connecting them with a "subpropertyOf" association. Moreover, OWL allows to state that a property is transitive, or symmetric, or that it is the inverse of another property: these aspects are accounted for by using different association types for each of them. Ontologies often comprise individuals, i.e. class instances: a class is an abstraction of a set of individuals sharing some common properties, thus an individual is characterized by the specific values assumed by those properties. An individual is represented in ebXML by an "extrinsic object", that is, a registry object containing metadata for a repository document. Both properties and individuals were included in the cataloging process in order to obtain a comprehensive representation of the ontology, although they are not of immediate use for the semantic tagging task. Once provided with the required classification nodes, semantic tagging is just a matter of associating documents to the correct categories. This can be done using either the standard UI or in an automated cataloger service.

Figure 10: an ontology related to the biological field loaded in OMAR

For the actual elaboration of OWL files we developed a dedicated software module: the Ontology Cataloger service. Its purpose is the creation of the registry objects required to mimic a given ontology, according to the directives in it. It consists of a Java program which leverages the Jena framework for ontology processing, in particular for the identification of classes and their relationships. Jena is a popular open source framework for semantic data management in Java, offering an environment for OWL, RDF, RDFS and SPARQL; it was born within the HP Labs Semantic Web programme, and is now an independent project. The objects resulting from the cataloging phase are stored in the registry, while the originating XML file is stored inside the repository like any other document. Figure 4 the outcome of the described loading activity in the case of a biological ontology. For the cataloger to be reachable by the Registry, we had to provide a Web Service interface, which we did using the Apache Axis engine, a well-known SOAP processor for the Apache Tomcat server. A reference to the actual service URL was then entered in the registry and associated to the OWL object type, thus activating the automatic invocation upon submission of OWL files.

3.7.2. Browsing the ontology

We already observed that entities apparently lose their hierarchical structure, since the “subclassOf” property is not natively supported. For this reason, we need an ad-hoc solution to query an ontology. One possibility is to use an external program, a management service, which would be able to access the registry/repository while leveraging dedicated libraries, such as Jena. However, such a loosely coupled architecture hampers the efficient implementation of ontology navigation. The service, while referenced inside the registry, is still an external entity that has no direct access to the registry. In order to perform any kind of query, it needs to

create a local copy of the ontology, downloading it from the ebXML repository. This approach could be feasible just with small-sized ontologies, and it requires a considerable communication overhead.

Due to these difficulties, we followed a different technique, which is offered by ebXML's standard *AdhocQuery* objects, explicitly meant to to wrap complex queries while offering a simple interface, similarly to what happens in databases when using stored procedures. Input parameters are usually entered through a graphical interface which completely hides the query logic. Behind the curtain, an AdhocQuery object ultimately contains a SQL query to be performed over the database underlying the Registry, and is thus implementation-dependent. The ebXML profile for OWL explicitly mentions several *canonical queries*, such as "find all superclasses of X" or "find all inherited properties of Y", that should be supported by a registry in order to explore OWL documents. Unfortunately, several of the most useful browsing queries imply recursion, which is only supported from the SQL-99 standard onwards. This means that the vast majority of DBMSs, that do not fully comply with SQL-99, will not be able to perform them. That was also the case for our environment (we used PostgreSQL 8.1); to overcome this limitation, in our implementation we replaced the flawed canonical queries with stored procedures written in PL/pgSQL, the PostgreSQL's procedural language.

3.8. Metadata organization using ontologies

Metadata are heavily used in document searches (much more frequently than content, because metadata are less tied to syntax and closer to semantics), and since searching is the most important function in DM systems, it becomes clear that efficient knowledge management can be enabled by a well-structured metadata model.

Unfortunately, no standard schema currently exists for satisfactory document characterization. The most notable effort, the well-known Dublin Core metadata set was explicitly designed to be minimal, in order to suit a wide range of applications. The Dublin Core schema does not accommodate the creation of complex user-defined structures, thus it lacks the flexibility often required in a number of application fields. This need for complexity arises when observing that different document types need different metadata sets to be fully qualified. This problem has grown even worse in recent years, due to the current trend of using DMSs like generic Content Management Systems (CMSs), containing not only textual documents but multimedia files as well.

3.8.1. Document-type ontologies

Since we cannot define a straight-forward "one-size-fits-all" metadata set, we need a more complex model to enable DMSs to manage metadata in a structured way, such as classifying similar documents using the same properties and taking advantage of hierarchies to express different levels of detail and abstraction. The description of entities, properties and their relations is exactly the focus of research about ontologies and related technologies: hence, we can leverage the infrastructure we discussed in the previous paragraphs to introduce metadata-related ontologies in existing DMSs.

In literature, ontology entities always describe documents' content, and are used as an enhancement of keywords. However, we spotted another possible utilization, limited to a specific class of ontologies, that can solve our document characterization problem. If the ontology describes *document types*, such as newspaper article, internal report and so on, it can be exploited when publishing new documents to the repository. In fact, each entity attribute is translated to a metadata field for that particular document category. Therefore, if we know that document *D* belongs to class *C*, then *D* must be qualified with all the attributes asserted (or inferred) for class *C*. Of course, only attribute names and domains are definitely set, while actual values differ among documents. These values are usually provided by the publisher, but might be extracted automatically, as seen in several recent papers, through machine learning techniques [33] or static rules [12].

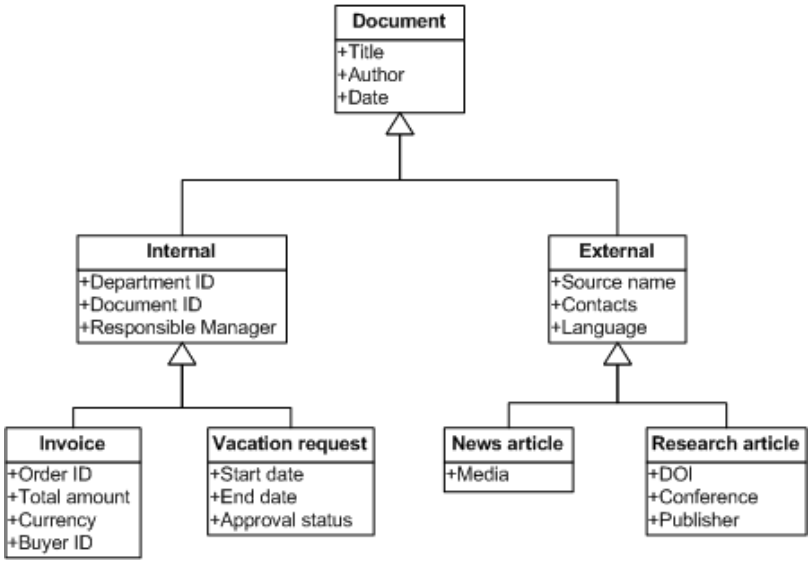

Figure 11: example of document-type ontology

Document-types ontologies can thus be a solution for the problem we initially stated (Figure 11). In order to support both traditional tagging and document-type tagging, we define two different annotation types, to specify whether either a *content*-related or a *document-type*-related ontology is referenced.

Assuming that at least one *document-type* ontology is present in the registry, our goal is to automatically annotate all the documents according to the type descriptions supported by the ontology. The annotation process consists of two steps: i) *metadata discovery*, i.e. detection of all metadata fields that make sense for that particular document type; ii) *metadata population*, i.e. fill in their values according to the actual document. In most of currently available, ordinary DMSs, the former phase is usually ignored, and metadata fields are either *statically defined* or *freely customizable*. Both approaches have limitations, one being

excessively generic (the same metadata set for all documents) and the other being excessively specific (potentially, a different set for each document). The set of interesting attributes to be extracted from documents is usually fixed, as Digital Libraries typically conform to the Dublin Core standard, or some variant of it. The Dublin Core metadata set consists of 15 elements, embracing data about the document itself (title, language, format) as well as people who worked on it (creator, publisher, editor). It frequently turns out to be excessively generic, thus different systems need to integrate the metadata set with different information. Obviously this limits the extent to which such systems can be labeled as "interoperable", although an extension mechanism is provided by the Dublin Core itself (Qualified Dublin Core) using element refinements. Basic compatibility should be insured by the "dumb down" principle, stating that a system that does not understand a refined attribute should be able to use the broader term.

Whatever the metadata set, once it has been chosen it is used uniformly over the entire corpus. This may be reasonable when documents share a common format and/or scope, such as research papers or case histories, and we have a priori knowledge of which metadata fields make sense in our context. However, when we face the need to manage an heterogeneous collection of resources, this approach is just not flexible enough. We can easily observe that different document types need different metadata sets to be properly characterized: a piece of information may be vital for one document and completely worthless (or not applicable) for another. For instance, a reference to the originating department may help the classification of internal reports produced within an enterprise, while it would be non-sense to attach that information to newspaper articles in a press review. At the other end of the spectrum, excessive flexibility, meaning completely unstructured metadata, is undesirable as well. Even though knowing that a document is related to the term "PDF" in some way could still be a valuable information, the nature of this relation makes a big difference, since "PDF" may be interpreted as the subject, the format, or even the author's initials. Of course, if documents are completely unrelated to each other and we have no prior knowledge about the metadata we can extract from their content, we cannot do much to obtain a structured characterization: therefore, we will make the assumption that our repository contains documents belonging to a definite, possibly hierarchical (or otherwise interrelated) set of document types, each qualified by a different metadata set.

In this setting, we proposed the introduction of *document type* ontologies as a reasonable trade-off, providing a flexible yet rigorous metadata model. A *document-type* ontology describes all the associations among document types and metadata fields, making use of hierarchies in a clean and extensible scheme. The document-type classification for one specific document implies the identification of all the related nodes in the document-type ontology. As a consequence, metadata discovery involves reasoning over the ontology, because fields for a given document type may not be explicitly asserted as properties of the related entity; instead, they may be inherited from super-types, or inferred in more complex ways. If a document references multiple metadata-related ontologies, it will be categorized by the union set of all the inferred fields.

Once metadata attributes have been discovered, their values for a given document can be figured out in several ways. Depending on the actual document format, it might be possible to automatically compute them, which is obviously the best solution. However, this is not always feasible, and human involvement is generally needed. In our implementation, we took the naive approach of leaving those fields empty: the publisher is supposed to take care of them right after submitting the document. Other more advanced solutions could assist the user by giving suggestions on most likely values for each metadata field.

3.8.2. Implementation details

We implemented this solution in an ebXML registry. Obviously, implementation of the OWL profile documented in previous paragraphs is a prerequisite. As for the OWL translation procedure, the whole annotation process is carried out by another software component, which we named *Document Cataloger*. Figure 12 shows the arrangement of the two Catalogers within the overall architecture. It is worth pinpointing the loose coupling of both the additional modules and the ebXML registry, leaving untouched the DMS core engine.

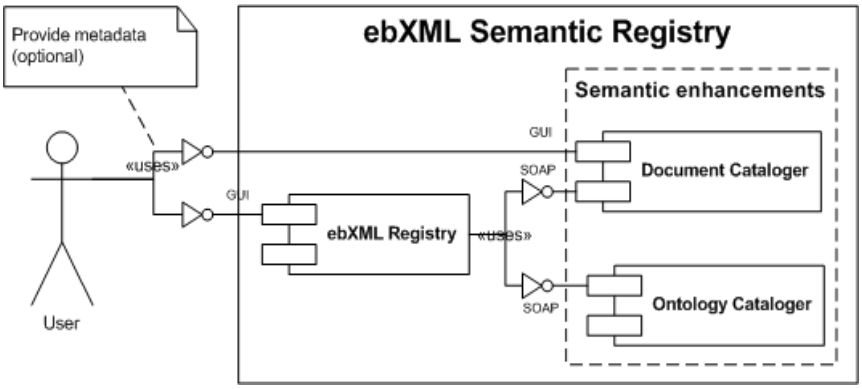

Figure 12: architecture of Semantic upgrades to an ebXML registry

Two practical issues arise from this arrangement: one is the need to accommodate semantic tags when a document is submitted to the registry; the other is the definition of the trigger which is supposed to wake the Document Cataloger. The first problem can easily be tackled using *slots*. A slot is a completely generic extension mechanism with no predefined meaning. It is composed by three fields (name, type and value) which can be freely filled: the only constraint about slots is the uniqueness of their name within the local registry object. Therefore, we chose to define a slot type for supporting semantic annotations (actually, two different types in order to discriminate between content-related and metadata-related ontologies), using the slot value to hold the unique identifier of the ontology's classification node.

The second problem is a bit more tricky. According to the standard specifications, catalogers have to be bound to a specific object type. However, in this case there

is no specific kind of objects that should be tied to the execution of the cataloger: the cataloger should simply be invoked anytime a semantically tagged document, whatever the type, is submitted for publication on the repository. There is no easy solution to the question, since semantic tags actually are generic slots which are not taken into consideration by the ebXML engine. Only two solutions apply:

- We could create a new document type (named *annotated document*), to bind the cataloger to. This solution, even if definitely feasible, is not satisfactory from the theoretical point of view because it defines a fictitious category embracing documents that may be completely unrelated to each other. Instead, the document type should be a more meaningful piece of information, regarding either content or format.
- We could associate the cataloger to *all* documents (i.e. the *extrinsic object* root node) so as to trigger its invocation from every publishing request. Thus the cataloger is in charge of detecting whether a document requires special processing, by looking for its possible semantic annotations.

For the sake of integrity and meaningfulness of the registry, we selected the second alternative and demanded all tests to the cataloger itself.

3.8.3. Sample publication

The usage of the ebXML registry as modified to support OWL ontology-based classification is briefly illustrated in this section, by describing the publishing mechanism first for a document type ontology, and then for an annotated document. In the following, we refer to the simple tree-shaped ontology shown in Figure 11.

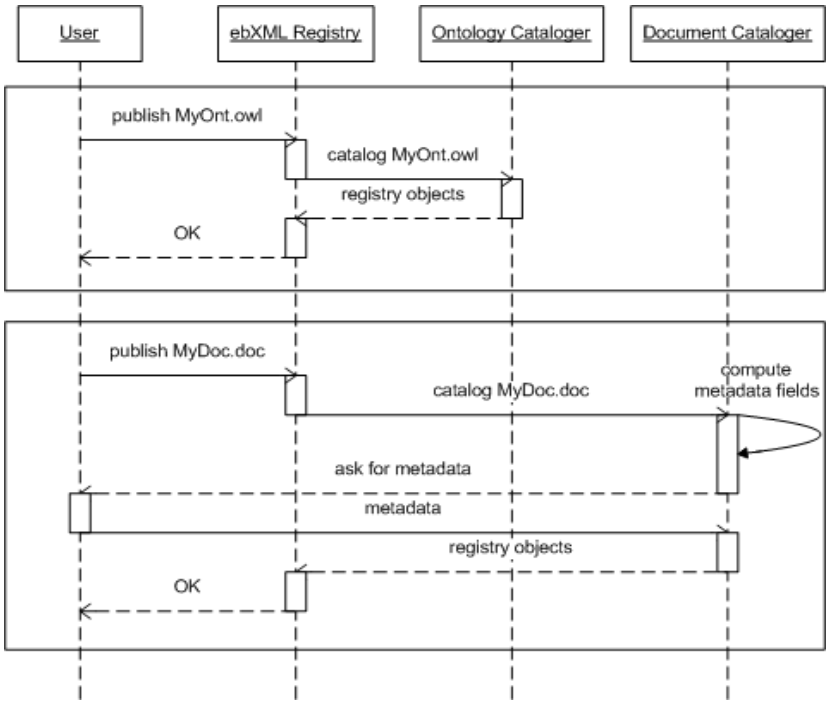

Figure 13: cataloging services in action for ontology and document publishing

The sequence diagram in Figure 13 shows how cataloging services are involved when publishing an ontology (top box) and an annotated document (bottom box); the latter use case assumes user interaction for metadata input. The first step is ontology publication. If the OWL cataloger is correctly set up, from the end-user standpoint this publication is identical to any other. No particular care needs to be taken, except for the object type which should be set to “OWL”. The cataloger will be automatically invoked, and the resulting objects will be inserted into the registry, according to the mapping defined in previous paragraphs.

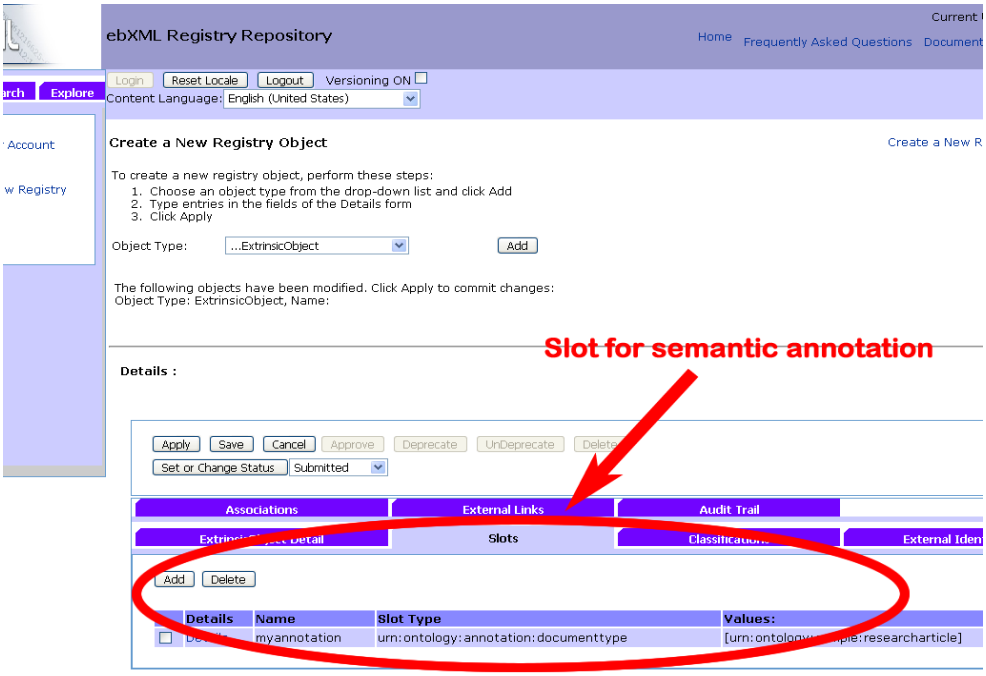

Figure 14: using slots via the repository GUI in publishing annotated documents

In order to illustrate the document categorization process (shown via the system GUI in Figure 14) , we consider the upload of a research article. The publishing request expected by the ebXML engine is an XML document formatted according to the ebXML Registry Services specification. This particular request will also include one or more slots representing our semantic annotations (Figure 15).

```
<?xml version="1.0" encoding="UTF-8" ?>
<lcm:SubmitObjectsRequest xmlns=… >
  <rim:RegistryObjectList>
    <rim:ExtrinsicObject id=… >
      <rim:Slot name="myannotation"
       type="urn:ontology:annotation:documenttype">
        <rim:ValueList>
          <rim:Value>
           urn:ontology:sample:researcharticle
          </rim:Value>
        </rim:ValueList>
      </rim:Slot>
      …
    </rim:ExtrinsicObject>
  </rim:RegistryObjectList>
</lcm:SubmitObjectsRequest>
```

Figure 15: XML content of a publishing request message

As the document cataloger has been bound to the *extrinsic object* type, this submission will automatically invoke the cataloger that, scanning the metadata of the received object, will find a slot whose type is "urn:ontology:annotation:documenttype". This is the type we defined for semantic annotations towards document type ontologies. The slot value is thus interpreted as the name of the entity (i.e. classification node) to be analyzed for metadata discovery: in our example, the "researcharticle" entity within the "sample" ontology.

The described analysis is actually performed by the registry, which can freely access the ontology without any communication overhead. The cataloger simply calls a couple of canonical queries provided by the registry, namely *FindAllInherited-ObjectProperties* and *FindAllInheritedDatatypeProperties*. Those queries exploit the ontology structure and eventually return a list of properties. Such values in this context define the metadata fields associated to that particular document type. In the case of a research article, those fields include DOI, Conference, Publisher (explicitly stated for the entity), Source name, Contacts, Language (inherited from External), Title, Author and Date (inherited from Document).

After the discovery phase, metadata fields can be filled in with values extracted from the document. As previously mentioned, and as shown in Figure 13, we chose to rely on a subsequent human intervention. Metadata are ultimately attached to the document as extrinsic object slots and returned to the registry for publication.

4. CONCLUSIONS

This work was intended to target a vacant niche in traditional academic studies: the relationship with the business world. Since the main problem with the realization of most systems in research papers sits in their requirement of a completely virgin and unconstrained environment, we decided to try the new direction of small-scale modifications to existing systems. In this perspective, we tackled several real problems related to the greatly felt topic of interoperability. Conforming with our minimum-impact philosophy, we came up with several small solutions, rather than a single comprehensive system. Notable results of our effort can be summarized as follows:

- We defined a generic interface to access the most commonly used DMS functions, independently from the actual system. We also provided an implementation based on the Web Service technology, so that an interoperable DMS might be queried in a technology-agnostic manner. The high relevancy of this achievement is due to the key role DMSs play in modern companies.
- We contributed to the cause of e-business standardization, designing an architecture to support the seamless introduction of ebXML registries in existing ISs. This architecture allows the exploitation of all ebXML functionalities over any kind of connected DMS, and thus eases the shift of applications' interactions from system-specific to standard-based.
- All the proposed solutions satisfy the common requirement of adding new possibilities without revolutionizing in-place systems, so as to avoid service discontinuities in pre-existing applications.

Following the thread of interoperability, we shifted our focus to the latest achievements in the field of data integration: semantic enhancements.

- We developed an accurate implementation of the semantic enhancements described in the OWL profile specifications for ebXML registries. To our knowledge, this is the first implementation of the standard.
- We showed the feasibility of a general ontology-based approach to semantic classification in state-of-the-art document management tools. Again, the objective was accomplished under the imperative of keeping the number of modifications as low as possible, leveraging existing features whenever appropriate, and ensuring consistency with external software.
- We delimited an often neglected sub-topic of semantic tagging, i.e. metadata discovery, and discussed its traits.
- We introduced the novel concept of *document-type* ontology and illustrated its usefulness with respect to the metadata discovery problem. Taking advantage of an OWL-aware ebXML registry, a sample architecture was devised and implemented to tackle the problem through this kind of ontologies.

It is worth noticing that the validity of our findings is endorsed by real world use cases and implementations. Some of the systems presented here are daily exploited in a multinational manufacturing industry. In the context of this work, this is a significant achievement, since it testifies that our hypothesis were correct and enhancement of knowledge management systems can actually be obtained through low-impact approaches.

REFERENCES

1. Anthony R.N., **Planning and Control Systems: A Framework for Analysis**, Harvard Business School Press, 1965.
2. Land, F.F., "The First Business Computer: A Case Study in User-Driven Automation", *IEEE Annals of the History of Computing*, Vol. 22, No. 3, 2000.
3. Simon H.A., **Administrative Behaviour 2nd ed**, The Free Press, 1976.
4. T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web", *Scientific American*, May 2001.
5. S. Bechhofer, F. van Harmelen, J. Hendler et al., **OWL Web Ontology Language Reference**, 2002. http://www.w3.org/TR/owl-ref
6. Ankolekar, M. Krötzsch, T. Tran, D. Vrandečić, "The two cultures: mashing up web 2.0 and the semantic web", 16th Int'l Conf. on World Wide Web, 2007, pp. 825-834.
7. T. Tran, P. Cimiano, S. Rudolph, R. Studer, "Ontology-Based Interpretation of Keywords for Semantic Search", ISWC/ASWC, 2007.
8. L. Specia, E. Motta, "Integrating Folksonomies with the Semantic Web" , **The Semantic Web: Research and Applications**, LNCS 4519, 2007, pp. 624–639.
9. Li Ding; Finin, T.; Joshi, A.; Yun Peng; Rong Pan; Reddivari, P., "Search on the Semantic Web", *IEEE Computer*, vol.38, no.10, Oct. 2005, pp. 62-69
10. Bojars, U.; Breslin, J.G.; Peristeras, V.; Tummarello, G.; Decker, S., "Interlinking the Social Web with Semantics," *Intelligent Systems*, IEEE , vol.23, no.3, May-June 2008, pp.29-40.
11. J. Mayfield, T. Finin. "Information retrieval on the Semantic Web: Integrating inference and retrieval", 26th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003.
12. M.Y. Day, R. Tzong-Han Tsai, C.L. Sung et al., "Reference metadata extraction using a hierarchical knowledge representation framework", *Decision Support Systems*, Vol. 43, n. 1, 2007.
13. Bechini, A. Tomasi, and J. Viotto, "Collaborative e-Business and Document Management: Integration of Legacy DMSs with the ebXML Environment", **Interdisciplinary Aspects of Information Systems Studies**, Physica Verlag, 2008
14. Bechini, A. Tomasi, and J. Viotto, "Enabling Ontology-Based Document Classification and Management in ebXML Registries", ACM SAC, 2008, pp. 1145-1150
15. Dogac, Y. Kabak, G. Laleci et al., "Enhancing ebXML Registries to Make them OWL Aware", *Distributed and Parallel Databases*, Vol. 18, n. 1, 2005, pp. 9-36.
16. T. Celik, K. Marks, "Real world semantics", O'Reilly Emerging Technology Conference (ETech), 2004.
17. Bechini, A., Cimino, M.G.C.A., Tomasi A., "Using ebXML for Supply Chain Traceability - Pitfalls, Solutions and Experiences", 5th IFIP I3E Conf., 2005, Springer, pp. 497-511
18. Bechini, A., Cimino, M.G.C.A., Marcelloni, F., Tomasi A., "Patterns and technologies for enabling supply chain traceability through collaborative e-business", *Information & Software Technology*, 2007

19. Bechini, A., Tomasi, A., and Ceccarelli, G., “The Ecumene Experience to Data Integration in Cultural Heritage Web Information Systems”, CAiSE Workshops, 2004, Vol. 1, pp. 49-59
20. J. Viotto, “Progetto UtenzePAT”, Technical report, 2009
21. J. Viotto, “Progetto Agorà”, Technical report, 2008
22. Petinot, Y., et al, “eBizSearch: an OAI-Compliant Digital Library for eBusiness”, JCDL, 2003, IEEE CS Press, pp. 199-209
23. Russo, L., and Chung, S., “A Service Mediator Based Information System: Service-Oriented Federated Multiple Document Management”, 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06), 2006
24. Carey, M., “ Data delivery in a service-oriented world: the BEA aquaLogic data services platform”, ACM SIGMOD international conference on Management of data, 2006
25. Oldenettel, F., Malachinski, M., and Reil, D., “Integrating Digital Libraries into Learning Environments: The LEBONED Approach”, IEEE Joint Conference on Digital Libraries, 2003
26. P. Castells, M. Fernandez, D. Vallet, “An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval”, *IEEE Trans. on Knowledge and Data Engineering*, Vol 19, No. 2, 2007.
27. S. Dill, J. A. Tomlin, J. Y. Zien et al., “SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation”, 12th In’l Con. on World Wide Web (WWW’03), 2003.
28. L. Ding, T. Finin, A. Joshi et al., “Swoogle: A Search and Metadata Engine for the Semantic Web”, 13th ACM Conf. on Information and Knowledge Management (CIKM’04), 2004.
29. A. Dingli, F. Ciravegna, Y. Wilks, “Automatic Semantic Annotation using Unsupervised Information Extraction and Integration”, K-CAP Workshop on Knowledge Markup and Semantic Annotation, 2003.
30. W. Fang, L. Zhang, Y. Wang et al., “Toward a semantic search engine based on ontologies”, 4th Int’l Conf. on Machine Learning and Cybernetics, 2005.
31. C. Fellbaum (editor), **WordNet: an electronic lexical database**, MIT Press, 1998.
32. M. Ashburner , C.A. Ball , J.A. Blake et al., “Gene Ontology: tool for the unification of biology”, *Nature Genetics*, Vol. 25, 2000, pp. 25–29.
33. H. Han, C.L. Giles, E. Manavoglu et al., “Automatic Document Metadata Extraction using Support Vector Machines”, Joint Conf. on Digital Libraries (JCDL’03), 2003.
34. S. Handschuh, S. Staab, F. Ciravegna, “S-CREAM - Semi-automatic CREAtion of Metadata”, Semantic Authoring, Annotation & Knowledge Markup (SAAKM 2002), 2002.
35. D. Lenat, R.V. Guha, **Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project**, Addison-Wesley, 1990.
36. I. Niles, A. Pease, “Towards a standard upper ontology”, International conference on Formal Ontology in Information Systems, 2001.
37. B. Popov, A. Kiryakov, A. Kirilov et al., “KIM - Semantic Annotation Platform”, 2nd International Semantic Web Conference (ISWC2003), 2003.
38. E. Sirin, B. Parsia, B. C. Grau et al., “Pellet: A practical OWL DL reasoned”, *Journal of Web Semantics*, Vol. 5, n. 2, 2007.

39. M. Vargas-Vera, E. Motta, J. Domingue et al., “MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup”, 13th In’l Conf. on Knowledge Engineering and Management (EKAW2002), 2002.
40. J. Lamping, R. Rao, P. Pirolli, “A focus+context technique based on hyperbolic geometry for visualizing large hierarchies”, SIGCHI conference on Human factors in computing systems, 1995.
41. K. Yee, D. Fisher, R. Dhamija, M. Hearst, “Animated Exploration of Dynamic Graphs with Radial Layout”, IEEE Symposium on Information Visualization, 2001.
42. A. Bechini, J. Viotto, “Enterprise Information Management towards the Semantic Era”, submitted for publication
43. A. Bechini, R. Giannini, J. Viotto, “Semantic support to document managent in bio-medical environment: a case study on molecular diagnostic procedures”, submitted for publication
44. A.P. McAfee, “Enterprise 2.0: The Dawn of Emergent Collaboration”, *MIT Sloan Management Review*, Vol.47, n. 3, 2006, pp. 21-28

APPENDIX A – INDUSTRY STANDARDS AND ORGANIZATIONS

Many of the topics covered in the presented work are concerned with applicative aspects of knowledge management and industrial solutions to known issues. In fact, the focal point of the whole thesis is to provide a bridge between the theoretical problems in academic work and actual systems currently employed in the business world. Due to their nature, these subjects are rarely treated in formal papers, or standardized in an official way. It is far more common to find online specifications or best practices, often recognized with a wide, but not necessarily global, consensus. This format allows a faster response to changes in market, technology and fashion; moreover, its informality eases comprehension and diffusion, and encourages implementation of the standard, even though this comes at a price: the risk of having a plethora of slightly different varieties of the specification. In this appendix, we list industry specifications, standards organizations and web references that play a major role in this field.

1. Document management

Documents are often cited in academic papers investigating metadata extraction, automatic classification, ontology inference or natural language processing in general. However, document management itself is more an applied science, and is highly relevant to companies. Latest developments in the field can be frequently found in fact sheets of real products rather than published in the scientific community.

Organizations and communities

- Dublin Core Metadata Initiative. http://dublincore.org
- Open Archive Initiative, http://www.openarchives.org/

Specifications:

- S. Weibel, J. Kunze, C. Lagoze et al., “RFC 2413 - Dublin Core Metadata for Resource Discovery, Technical report”, Internet Engineering Task Force (IETF), 1998.

Commercial products:

- FileNet Corporation. IBM Filenet P8 Platform, http://www.filenet.com/English/Products/Datasheets/p8brochure.pdf
- EMC Corporation. Developing Web Services with Documentum, http://www.software.emc.com/collateral/content_management/documentum_family/wp_tech_web_svcs.pdf
- Vignette Corporation. Vignette Portal and Vignette Builder, http://www.vignette.com/dafiles/docs/Downloads/WP0409_VRDCompliance.pdf

2. ebXML

The ebXML suite of e-business standards is sponsored by the OASIS consortium for open standards and the CEFACT center of United Nations. The freebXML intiative is in charge of promoting its adoption.

Organizations and communities:

- OASIS: Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org
- United Nations Centre for Trade Facilitation and Electronic Business. http://www.unece.org/cefact
- Electronic Business using eXtensible Markup Language. http://www.ebxml.org/
- FreebXML. http://www.freebxml.org/
- OASIS ebXML Registry Reference Implementation,. http://ebxmlrr.sourceforge.net

Specifications:

- S. Fuger, F. Najmi, N. Stojanovic et al., ebXML Registry Information Model specification version 3.0. http://docs.oasis-open.org/regrep/regrep-rim/v3.0/regrep-rim-3.0-os.pdf
- S. Fuger, F. Najmi, N. Stojanovic et al., ebXML Registry Services specification version 3.0. http://docs.oasis-open.org/regrep/regrep-rs/v3.0/regrep-rs-3.0-os.pdf
- Dogac, Y. Kabak, G. B. Laleci et al. ebXML Registry Profile for Web Ontology Language (OWL) Version 1.5. http://docs.oasis-open.org/regrep/v3.0/profiles/owl/regrep-owl-profile-v1.5.pdf

3. Semantic resources

Most standards and recommendations related to the Semantic Web are governed by the World Wide Web Consortium (W3C), and are available online throughout their standardization process. The biomedical field is particularly active in putting those specifications in practice, defining several huge domain ontologies and using them for data integration.

Specifications:

- D. Beckett (editor). RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/REC-rdf-syntax/
- D. Brickley, R.V. Guha (editors). RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/
- S. Bechhofer, F. van Harmelen, J. Hendler et al. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref
- Adida, M. Birbeck, S. McCarron et al., RDFa in XHTML: Syntax and Processing. http://www.w3.org/TR/rdfa-syntax/
- Microformats. http://microformats.org/wiki/Main_Page

Tools:

- Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net
- Protégé ontology editor. http://protege.stanford.edu
- RacerPro OWL reasoner. http://www.racer-systems.com

Ontologies:

- The Gene Ontology. http://www.geneontology.org
- OBO Foundry. http://obofoundry.org
- BioPAX: Biological Pathway Exchange. http://www.biopax.org
- Cyc and OpenCyc. http://cyc.com/cyc, http://www.opencyc.org/