An Infrastructure for Managing Semantic Documents

Conference Paper · October 2010

DOI: 10.1109/EDOCW.2010.17 · Source: DBLP

2 authors:

Lucas de Oliveira Arantes
Universidade Federal do Espírito Santo
5 PUBLICATIONS **15** CITATIONS

SEE PROFILE

Ricardo de Almeida Falbo
Universidade Federal do Espírito Santo
172 PUBLICATIONS **1,661** CITATIONS

SEE PROFILE

2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops

An Infrastructure for Managing Semantic Documents

Lucas de Oliveira Arantes
UFES – Univesidade Federal do Espírito Santo
Vitória, Brazil
lucasdeoliveira@gmail.com

Ricardo de Almeida Falbo
UFES – Univesidade Federal do Espírito Santo
Vitória, Brazil
falbo@inf.ufes.br

Abstract—Software Documentation is an important mean for stakeholders to collaborate in the software development context. However, several works point out that gathering relevant information from different documents can be so wearing that involved people may tend not to do it. Combining ontologies and documents by adding semantic annotations to documents can help diminish the burden of gathering information later on. However, this approach also adds an overhead in documentation, concerning the time spent on document annotation. In order to overcome some of these obstacles, we developed an infrastructure for managing semantic documents, combining semantic annotation on document templates, versioning data extracted from semantic documents and notifying interested people when extracted data has changed.

Keywords: Semantic Web, Semantic Documentation, Ontology-based Semantic Annotation, Knowledge Management

I. INTRODUCTION

Despite the current advances in electronic documentation along with the boom of collaborative text edition tools (such as wiki engines found nowadays), desktop text editors are still the most pitched solution used by software organizations when it comes to electronic documentation [1,2]. Whether on the modern approach (the use of wiki engines) or the classical approach (the use of desktop text editors), documents produced by these tools are still the main vehicle for knowledge dissemination [1, 3, 4].

In the context of a software project these documents holds a considerable amount of information (such as use case descriptions, requirements, human resource allocations, etc) that are mainly interpreted by human readers. Besides the fact that these documents are, normally, directed for human reading only, the process for producing and maintaining them can generate untrustworthy information [1]. Moreover, tracking down the evolution of the data contained in these documents is only achieved by reading each version of the document. Obviously this activity is often dull and could lead to misinterpretation. Additionally, gathering relevant information contained in different documents spread through the organization repositories demands a considerable effort and, because of that, this activity is often skipped [1].

Communication between involved parties can be eased if the data embedded in the documents are reachable in a way that does not require a thorough reading through the whole document in order to achieve good understanding. In other words, the semantic content of the document could be exposed in order to allow visibility of the data and the relationships embedded in the document. Tracking the changes between document versions could be also facilitated if the semantic content of each document version is extracted and registered into a version control system. In order to make these scenarios possible and achieve semantic richness, it is essential to allow metadata annotation into documents. Additionally, strategies for metadata annotation, extraction and searches based on metadata are equally important.

Taking into account this scenario, we developed an Infrastructure for Semantic Document Management (ISDM), covering the following features: semi-automated semantic annotation of documents through the use of semantic annotations embedded in document templates; data evolution traceability; advanced data searching based on extracted semantic content; and change notification subscription.

This paper aims to present this infrastructure and discuss how it can be used to support the integration of information spread in several documents. The paper is organized as follows: Section II regards the theoretical background of the paper, discussing briefly the Semantic Web wave and its reflexes in documentation, giving rise to a new area of study, the Semantic Documentation; Section III briefly discusses related work and points out the existing gaps that motivated us to develop our work; Section IV presents the ISDM and its main components; Section V discusses an example of the use of ISDM in a requirements management context, showing how it can be used to support the integration of information spread in several requirements related documents; Section VI compares our work with the related works presented in section III; finally, Section VII presents our conclusions and future work.

II. SEMANTIC WEB AND SEMANTIC DOCUMENTATION

Since the conception of the World Wide Web, web pages were mainly written in formatting languages (such as HTML) in order to allow browsers to present information to human readers. Web pages were not initially meant to hold machine interpretation [5, 6].

The increasing amount of web content started to show some serious problems. First, searching for particular information can be a burden, since the results generally include a considerable amount of material that is either irrelevant or does not contain the same semantics of what we are looking for. In other words, searches do not hold the desired precision, presenting many items that are not really relevant [6]. Another recurring problem is that the web pages

978-0-7695-4164-8/10 $26.00 © 2010 IEEE
DOI 10.1109/EDOCW.2010.17

that allow machine-readable solutions are not standardized and most of the time semantic information is not embedded into the page content, but as separate files. This scenario enhances the efforts applied on page maintenance and integration with other systems [6].

The semantic web aims to tackle the lack of semantics by providing a way that both humans and computers can interpret the content of web pages [5]. In order to reach this, generally web pages are annotated with metadata that describe small fragments of the page content in a way that machines could read, interpret and use them for search disambiguation and reasoning.

In order to achieve this purpose, metadata cannot be created in an ad-hoc way. It is essential to establish a common vocabulary that guides people and machines that are working with a page. In this context ontologies are of great value to establish concepts and relations that are used as metadata and to maintain relationships between these and external ontologies [7].

Even though there is an increasing movement towards the semantic web, relying on web pages in order to reach semantic interpretation by both human and machine is not enough. There is a considerable amount of work done in desktop electronic document tools, such as traditional text editors, that are very important for organizations. Some companies still rely on this kind of document for many activities such as: registering and controlling their knowledge, invoicing employees, managing projects and so on. Thus, semantic web alone does not solve the issues that these sectors encounter themselves in. In this context, semantic documentation is the key for tackling these scenarios. They combine documents and ontologies in the same way that the semantic web proceeds: ontology-based metadata is created and then attached to the document content. Since the annotations are based on a common ontology (or a set of ontologies) and all the data could be spread through different data sources (documents, legacy databases, etc), ontologies provides a framework for information integration [8]. Also, relating documents and ontologies through metadata annotation allows users and machines to navigate from the document to the concept that generated an annotation contained in the document, providing the necessary semantics to perform tasks such as search disambiguation [7,9]. Figure 1 presents the idea behind semantic documentation: annotation tools make use of ontologies to map text in documents to concepts, relationships and instances.

Allowing users to add metadata annotations to documents can improve understanding and accessibility of the data contained on it. However the task of adding metadata manually can consume a considerable amount of time [7] and is susceptible to errors [8]. Therefore, using semi-automated techniques for annotating is necessary. Mature companies normally maintain a set of document templates that guides creating and maintaining specific document types. Adding semantic annotation to these templates can be a simple approach that will diminish the overhead users may found when it comes to document annotation.

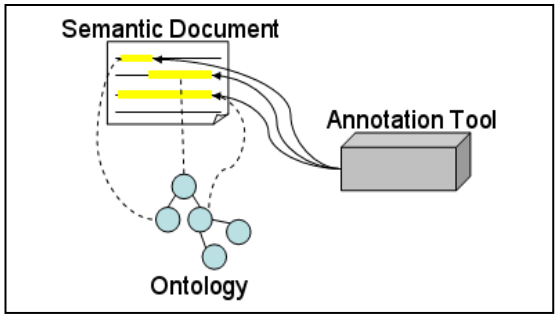

Figure 1. Semantic Document and its relation with Annotation Tool and Ontologies

Another important issue concerning software documentation is change control. Tracking the evolution of documents during the life cycle of a project can be vital for software organizations. A very common problem in software projects is the mismatch between what a specification document estates and what is implemented [10]. This happens as the software evolves. For instance, a change made on a requirement specification document can encompass a considerable amount of information. Be the following example where the scope of the project has changed and some requirements (namely 1, 2, 3 and 6) had their descriptions updated. In this example, developers working on these requirements should be aware of the changes, since they may critically affect their work. Without the use of tools for managing specific artifacts of the software project, in this case, it is very difficult to make sure that involved people receive the information they need in order to correctly perform their task, and ultimately change their code to make avoid mismatch between the specification agreed by clients and the program code. Reading the whole requirement specification document every time it changes is obviously a dull and error prone activity. In [11] this scenario is observed in order to enhance merging operations between groups of developers. In our work, the vision is extended to other software artifacts in order to provide data visibility (awareness) and to enhance the understanding of the data contained on the software artifacts.

III. Related Works And Motivation

There are several works that aims to deal with the problems related in the previous section.

In [12], a framework for structuring and extracting information is proposed. This framework uses an extension of an existing ontology (OSM), including extra regular expressions, to extract fundamental keywords found at the ontology and ontology relationships connectors from a given web page. In a nutshell, the authors implemented a solution based on domain ontologies that extracts information from HTML pages, populating specific relational database that is created according to the domain ontologies. The entire process consists in: (1) obtain an HTML web page that is related to the given domain; (2) remove unwanted HTML tags from the document, maintaining only the textual interesting part; (3) apply a parser generated from an ontology of the same domain of the document and the

regular expressions; (4) include extracted data into the relational database. Since there is extracted semantic data persisted on the database, it is possible to execute SQL queries that are specific to the document's domain. This way, this work provides semantic search and data extraction.

In [7], Eriksson and Bang discuss the importance of managing organization's documents inside a repository and how these documents could be incremented in order to become Semantic Documents. Basically the authors propose an approach to annotate electronic documents that will ultimately be published on a repository (Semantic Document Repository). The use of ontologies is a key in order to guide the metadata creation within the documents. Also, they propose an infrastructure for managing semantic documents, including services for advanced searches and reasoning using the documents' metadata, among others.

Concerning storage services, Eriksson and Bang examine three types of services that support semantic-document repositories: manual annotation, automated analysis and annotation, and document generation. In the first, users interact with annotation tools, relating concepts and individuals from an ontology to document text fragments. Despite the utility of this approach, it is clear that this is a time-consuming activity. An automated analysis and annotation approach can minimize manual annotation work by taking advantage of the document structure to add annotations manually. Finally, document generation aims to automating as much of the annotation task as possible by taking advantage of information available at document-authoring time. It can use specific data sources and ontologies that describe the data in these data sources to generate annotated documents based on existing document source structures.

For annotating documents, Eriksson and Bang proposed an infrastructure that involves three types of ontologies: (i) a domain ontology defining the vocabulary used for annotation; (ii) a document ontology defining the elements found on documents (such as text fragments, paragraphs, sections, tables, images, and so on); and (iii) an annotation ontology that links the document elements (found in the document ontology) and the elements in the domain ontology.

Kim et al. [13] developed an architecture for managing electronic documents that use wrappers created based on domain ontologies to extract metadata from resources coming from the web or social desktops (collaborative desktops). Their main focus is to add annotation into PDF (Portable Document Format) documents. This is done by the use of XMP (eXtensible Metadata Platform), which allows users to embed data about a file in itself. To do that, a set of ontologies is used to create annotations: a document schema ontology, a document type ontology and domain ontologies. The first two ontologies are part of their platform and are used to guide the annotation. The document schema ontology defines the elements of a document. The document type ontology describes publication's type of research communities and relevant concepts, such as proceedings, thesis, article, technical reports etc. Domain ontologies describe a certain subject which is closely related to a content of document. Domain ontologies can be added or edited by end users in order to fulfill their intention while annotating a document.

In the proposed architecture, key functions or processes are extraction, generation, indexing, and search. Concerning data extraction, data embedded in documents are gathered using Jena RDF framework to generate RDF graphs. The resulting data contained in the extracted graphs are properly indexed and then persisted in a data repository for later advanced searches. Two main components compose the proposed architecture: (i) a data repository, which is responsible for indexing and storing the extracted data; and (ii) the main module, which is responsible for domain ontology edition, and metadata extraction and generation. This module also provides an advanced ontology-based search interface for searching the data repository.

Semantic Word [14] provides utilities to annotate documents edited in Microsoft Word and to publish them on the web. Basically, Semantic Word extends MS Word allowing users to select resources from the web, such as ontologies and semantic web pages, and provides ways to use these resources to compose annotations that will be embedded in the document content. All annotations are written in the knowledge representation language DAML+OIL (DAML – Darpa Agent Markup Language, OIL – Ontology Inference Layer).

When a user selects web resources, instances, concepts and relations are listed to her in order to create annotations in text fragments. Two types of semantic annotation are provided: instance references and triple bags. An instance reference links a text fragment selected by the user to an existing instance of a concept of the ontology. Triple bags describe the content of a text region with a collection of triples that follow DAML+OIL's subject-predicate-object model. The subject is an instance, the predicate is a property defined in an ontology, and the object can either be an instance or a value.

Besides the semantic annotation feature, Semantic Word provides an integration with AeroDAML, an information extraction system. AeroDAML processes text and produces DAML markup that relates instances and values to the web resources retrieved. This can enhance significantly the manual annotation procedure, thus reducing the overhead of annotating an entire document from scratch.

Another important feature of Semantic Word is the ability to create semantic templates. This feature allows a user to annotate a document template, providing semantic annotation reuse and therefore improving the creation of semantic documents.

Besides the problem of creating semantic documents, another problem to be considered is to control changes on them. In this context, the Molhado framework [10] goes a step ahead. This framework aims to manage software of any domain using a logic approach. Based on version control systems and the software configuration management (SCM) process, Nguyen developed this framework in a way that developers must execute their software configuration management activities centered on the objects generated by them without worrying about the object's representation in a

file, like the standard version control systems do. In order to use the framework properly, developers must model their domain in terms of Java classes, extending the abstract classes provided by the Molhado framework. There also must be specialized tools that can generate instances of the model constructed by the developer that are connected to the SCM modules of the Molhado framework. As a result, data generated by these tools will be versioned as it evolves and possible mismatch that may appear between files and objects can be diminished. An alternative for existing tools is to build converters that can use services provided by the framework in order to convert data or use SCM functions.

The work carried out by Ognyanov and Kiryakov [15] also highlights the importance of tracking data evolution. In this work the authors focused on knowledge bases, proposing data versioning for RDF(S) repositories. Knowledge base repositories are materialized as RDF(S) graphs and a repository change history is treated as a sequence of RDF(S) graph *states* over time. Therefore a *state* is a snapshot of the actual situation of the knowledge base

As we can see by the analysis of the works briefly discussed before, features such as document annotation, automated document annotation, data extraction from metadata, data indexing and searching is somehow the base for semantic documentation. Moreover, we need to concern about changes on these documents. Looking at the analyzed approaches it is possible to observe some common aspects between them and therefore there is a chance to advance by tying important features together and adding some other ones. In resume, the works analyzed before contribute to the development of our Infrastructure for Semantic Document Management in the following ways:

- Embley et al. [12] described a way to extract data from web documents, persist them on a relational database and allow data search later on. However, the authors do not provide a way to annotate web pages, and so the content of the pages does not carry any metadata. Other important point is that this work does not address any kind of architecture for managing semantic documents.
- Tallis [14] provided semantic annotation to MS Word documents, semantic templating, semantic web document publishing and semi-automated annotation with an information extraction system (AeroDAML). Nevertheless, Tallis did not provide a way to manage the document itself and relied on the web as the document repository. Although this is useful, in most cases, the reality within a project is that the document will evolve and tracking its evolution can be crucial.
- Nguyen [10] constructed a flexible framework that provides an abstract model for data and service versioning. In order to get domain-specific data versioning, it is necessary to: (i) model a target domain (by extending an abstract data versioning model); (ii) create specialized tools (or converters) to generate instances of that model; (iii) use the SCM functions provided by the framework. Although the contribution is clear (diminishing the mismatch between versioned files and objects by providing a way to version the objects themselves), creating a domain model, or modifying an existing one, and providing a tool (or converter) to generate instances of that model can be a time-consuming effort.
- Eriksson and Bang [7] and Kim et al. [13] described not only ways to enrich the content of a document with semantic metadata, but they also proposed an architecture that allow the storage of semantic documents along with indexing and searching over the extracted semantic data. Despite the considerable contribution, these authors did not mention the use of semantic templates, like Tallis [14] suggested. Using templates has shown to a good practice for software documentation and we believe that annotating document templates can considerably decrease the time spent on document annotation. Moreover, annotating document templates can also make the document annotation procedure transparent for the end user.
- Ognyanov and Kiryakov [15] raised the importance of tracking changes on knowledge bases. Their focus is on providing strategies in order to maintain traceability between updates made upon a knowledge base. Although their work is somewhat in line with the discussion done in this section, they are not concerned with semantic documentation and how to manage semantic documents as a whole.

Summing up some of the features found at the works discussed in this section, it is clear that managing semantic documents as a whole is necessary. During the creation of a semantic document, the annotation procedure must be as transparent to the end-user as possible. During its maintenance, the changes applied to it must be tracked in order to provide visibility of its evolution, thus providing some level of understanding of exactly what has changed in the document. The common aspects treated in the proposals discussed before, if summed, can lead to a robust model. Therefore there is no need to supply a new model but an enhanced architecture that are able to fulfill possible existing gaps. This is the purpose of our Infrastructure for Semantic Document Management, which is presented in the next section.

IV. AN INFRASTRUCTURE FOR SEMANTIC DOCUMENT MANAGEMENT

In essence, the Infrastructure for Semantic Document Management (ISDM) we developed provides:

1. a way to semantically annotate document templates;
2. a mechanism for control version of the semantic content extracted from each semantic document version, therefore providing a way to track the evolution of the data embedded inside a semantic document;
3. data visibility to end-users, allowing searches and data-change notification subscription, to aid a developer to get a refined up-to-date information about something he/she is interested in.

In this paper, we use the term "semantic content of a document" to refer to the set of individuals (instances of the concepts of the ontology) and their relationships that can be extracted from the semantic document. We also use the term "graph" as a technical synonym for this expression.

Figure 2 shows an overview of the ISDM architecture, which comprises two main elements: (i) the Semantic Document Repository (SDR) that is responsible for storing semantic documents; and (ii) the Main Module, which is composed by the following sub-modules:

- Semantic Annotation Module (SAM): responsible for allowing users to semantically enrich a document template;
- Data Extraction and Versioning Module (DEVM): responsible for extracting the semantic content from an annotated document whenever a new version of that document is checked into the Semantic Document Repository. After extraction, the semantic content is stored, along with version information in another repository, called Data Repository, that is also part of this module;
- Search and Traceability Interface Module (STIM): responsible for providing an API (Application Programming Interface) that allows users and other systems to perform ontology-based searches and data traceability towards the Data Repository.

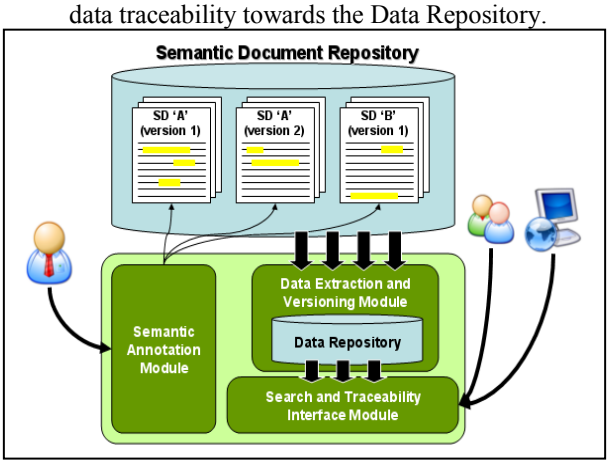

Figure 2. Overview of the ISDM Architecture.

In a nutshell, document engineers (human icon with a tie on the left side of Figure 2) annotate organization's document templates using the Semantic Annotation Module (SAM). Later on, developers and analysts may instantiate that template, generating a semantic document. This semantic document can be checked into the Semantic Document Repository (SDR). When a new version of a semantic document is available at the SDR, the Data Extraction and Versioning Module (DEVM) unwraps the semantic content of that version and store it into the Data Repository, making the information of that semantic content version available. At this point, it is possible to further enhance the integration of information scattered throughout various semantic documents contained in the Semantic Data Repository. This can be done by merging the graphs corresponding to the last versions of the documents contained in this repository. Finally, users and other systems (on the right side of Figure 2) may interact with the Search and Traceability Interface Module (STIM) in order to perform searches and queries about the evolution of a particular semantic content stored in the Data Repository.

The Semantic Document Repository is, in fact, a repository in a version control system such as Subversion [16]. In the sequence, the components of the Main Module are presented and discussed in more details.

A. Semantic Annotation Module – SAM

As discussed in Section III, there are several ways of annotating documents described in the literature. Notably, almost all of them make use of ontologies in some extent. Vocabulary standardization and metadata generation are the most pitched scenarios where it comes to ontology use in document annotation [4, 13]. Our approach, such as the one proposed in [7], uses domain ontologies, an annotation model and a document model.

The simpler way to semantically annotate a document is to create annotations manually. This scenario is known to be problematic, because the annotator may make mistakes and the annotation procedure is notably time consuming [4, 8]. As pointed by Eriksson and Bang [7], it is necessary to automate as much as possible the annotation procedure. Additionally, it is interesting to give some level of transparency to the end user, making her mostly unaware that she is actually producing a semantic document.

Tallis [14] proposed an interesting way of annotating desktop Microsoft Word documents. In this approach, document templates were annotated in order to reuse annotations made priori. This way both automation and transparency are achieved. Summing that to the fact that most mature software organizations use templates to produce documents, this approach can help companies organize and control their data retained in their document repositories. Thus, our approach consists in making domain ontology-based annotations in document templates.

The Semantic Annotation Module should provide the tooling for document engineers annotate document templates. For this purpose, we selected a standard document format and a tool to be the basis for template annotations. As the standard format for annotation, we chose the Open Document Format (ODF) [17], since it is an open format, with great span. Open Office [18] was chosen as the base office document editor for composing annotations in document templates.

In order to allow annotation in ODF document templates and make sure that these annotations are properly spread into an instance of that template, a mid level structure was developed. It is worthwhile to point out that we are working on document template annotation. Therefore, annotations must be specialized for the document element kinds that may exist in a document. Also an annotation must give directives to perform actions when the document is instantiated from the template, such as to create an instance, or to create a relation between instances and values.

Specialized annotations for annotating text fragments and tables were produced using Open Document Text (ODT) files. These kinds of annotations (template annotations) encapsulate annotations directed for document instances, allowing document engineers to add a set of instructions directly in these document elements (tables and text fragments). These instructions will be performed when the document instance is analyzed by the Data Extraction and Versioning Module that will ultimately generate instances and relations accordingly. An instruction is specialized in a way that it is the annotation itself and it is also an interface with the data extraction module, allowing it to create variables during extraction time and, therefore, granting ways of relating instances inside a document instance.

Instructions can be used for creating instances and relations. The syntax for the instance creation instruction is:

instance(arg,concept, accessVariable)

This instruction creates an instance of the *concept* (a concept of an ontology that is available in a given URL), using as identifier the value of *arg*. The result is a reference to the created instance and it is set on the *accessVarible* for later use.

The syntax for the relation creation instruction is:

property(arg1,prop, arg2)

This instruction establishes a relation *prop* between the instance referenced in *arg1* and an instance or a value given by *arg2*.

The first thing to do when creating a semantic template is to set a hidden input field in the corresponding ODT file with name = "SemanticDocument" and value = "true". This way the platform will be aware that this document is a semantic document and will start searching for semantic annotations. Also it is important for the extraction mechanisms to know what are the ontologies treated in the semantic document. To inform that, the user must create another hidden text field on the document with name = "Ontologies" and value equals a comma-separated list of complete URLs containing the path to the ontologies. So far, all the ontologies must be implemented in OWL (Ontology Web Language) [19].

As mentioned before, it is possible to annotate a text fragment with a set of instructions. In order to accomplish this appropriately, the user must define a text formatting style and give it a name. The style name will be related with a hidden field that will contain all the instructions defined for that particular text fragment. Following there is an example of a text fragment annotation:

[[textspan]]instance({content},http://localhost/ontologies/SE/onto.owl#Project,$project);

The tag *[[textspan]]* indicates that this annotation refers to a text fragment. Ultimately this annotation points out that the content of the text fragment (formatted using the style created before) will be used as the identifier of the created individual of the concept *Project* of the ontology available at *http://localhost/ontologies/SE/onto.owl*. The resulting individual will be recorded in the variable *$project* in the data extraction map and can be accessed and used later during the semantic document extraction procedure.

Annotations on tables are usually more complex, because each column might have a different set of instructions. To annotate a table in a template, it is necessary to first set a name to the table. This name will also be related to a hidden field with instructions for each column as value, as illustrated in the following example.

[[at0]]instance({content},http://localhost/ontologies/SE/onto.owl#FunctionalRequirement,$req);
property($req,http://localhost/ontologies/SE/onto.owl#artifactProducedIn,$project);
[[at1]]property($req,http://localhost/ontologies/SE/onto.owl#description,{content});

In this example the first instruction says that, for each line of the annotated table, from column 0 (the first column, indicated by the tag *[[at0]]*), the extractor will create an individual of the concept *Functional Requirement* of the ontology available at *http://localhost/ontologies/SE/onto.owl*, using as its identifier the content of this cell. The second instruction says that there is a relation (*artifactProducedIn*) between the requirement *$req* created by the first instruction and the individual *$project*. Finally, the last instruction says that the content of column 1 will be set as the property *description* of the requirement *$req*.

So far, all the annotations in the template are done without using a specialized Open Office plug in.

B. Data Extraction and Versioning Module - DEVM

The Data Extraction and Versioning Module is responsible for extracting semantic content out of the semantic document versions and store them appropriately in the Data Repository. All the semantic documents must reside in the Semantic Document Repository (SDR), which is, as said before, simply a repository in a version control system. In the current version of the ISDM, we are using a Subversion [16] repository as SDR.

Most version control systems allows the use of hooks, which are programs triggered by some repository event, like a file being checked into the repository. Taking advantage of this feature, when a semantic document is checked into the SDR (added for the first time or updated), a hook installed there calls the semantic content manager application, which starts the extraction procedure. First, semantic content manager checks whether the file being stored in the repository is a version of a semantic document, i.e., if it has a hidden input field with name = "SemanticDocument" and value = "true". Second, the semantic content manager extracts data from this file and creates a graph of the extracted data. Finally, it stores the derived graph in the Data Repository (DR). Figure 3 shows the behavior of the DEVM. It is worthwhile to highlight that each version of a semantic document (for instance SD 'A') located at the SDR have a corresponding graph on the Data Repository.

Basically the extraction process transforms the annotations in the semantic document into an OWL graph using the Jena OWL framework [20]. All the information extracted about the document version is registered in the Data Repository. The relation between the version of the

semantic document and the registry created in the Data Repository is also maintained.

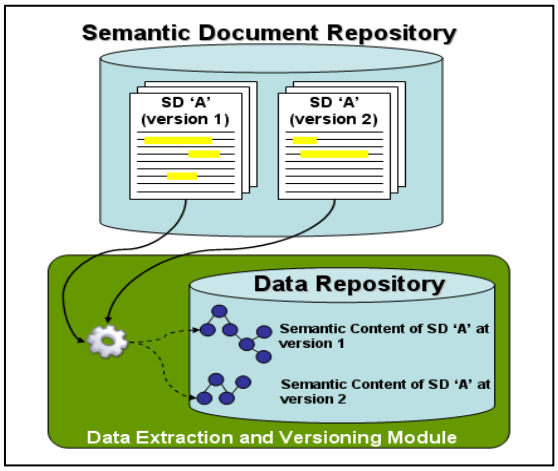

Figure 3. Overview of the Data Extraction and Versioning Module

The SDR is filled not only with different versions of a single semantic document, but also with different semantic documents that may have many purposes. Thus, there is a considerable amount of extracted and versioned data coming out from different versions of different semantic documents. In the context of software development, organizations generally have a version control repository per project. Thus, the SDR and the corresponding Data Repository contain valuable information about a given project. Requirements, use cases, personal allocations, stakeholders, and so on are all of them located somewhere in a graph of the Data Repository. Since these graphs are treated as sets (as in Set Theory) it is possible to combine them through set union operations, thus integrating different information contained in the Data Repository. Taking advantage of this feature, the platform performs a union operation, combining all graphs corresponding to the latest versions of each semantic document contained in the SDR, generating an overall integrated graph. This graph is also persisted in the Data Repository, in order to allow reasoning and advanced searches over the data. A fragment of the DEVM class diagram that encompasses the extraction and versioning is presented in Figure 4.

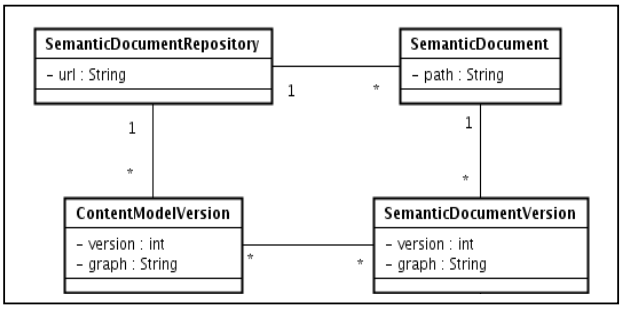

Figure 4. A partial class diagram of the DEVM regarding the structure of data versioning.

The class *SemanticDocumentRepository* represents the SDR and is located in a URL. A SDR has several semantic documents (represented by the class *SemanticDocument*), each one located in a specific path within the SDR. Each semantic document version (*SemanticDocumentVersion* class) maps the evolution of a single semantic document by registering the extracted graph and its version number. The class *ContentModelVersion* represents the union graph putting together all the graphs referring to the latest versions of the existing semantic documents in the SDR.

The technologies used for implementing the Data Repository, the office document reading, the persistence operations and the graph operations are, respectively, the PostgreSQL relational database, ODFDOM [21], Java with Hibernate Object Relational Mapping framework, and Jena OWL framework [20].

C. *Search and Traceability Interface Module – STIM*

The STIM focus on providing a common interface for external use. Applications may interact with this module to accomplish tasks such as:

- Searching for data in the Data Repository;
- Tracking the evolution history of a given individual;
- Subscribing change notification.

Searching interface is allowed by the use of SPARQL [22] queries over the union graph and over versions of it.

The term "evolution" used here is concerned to the different relations that a given individual may have throughout different graphs. There are two ways of checking the evolution of an individual: document level check and union graph (or content model) level check. The first allows applications to check how an individual evolved in a set of versions of a semantic document, such as checking how a given requirement evolved along the versions of a requirement specification document. The union graph level check allows checking an individual evolution throughout different versions of the union graph. Since the union graph gathers data from different documents, the amount of information is usually greater than the document level check. Both approaches use the following procedure:

1. Retrieve all the desired version graphs that contains the individual (whether it is at document-level or union graph level);
2. For *n* retrieved graphs, apply graph difference operation between graph *n* and graph *n-1*, including the resulting graph in a list;
3. For each difference graph in the list, query statements about the individual and return them.

Providing change notification subscription can improve greatly the understanding of a given SDR. In the context of a software project, it may help subscribers to take necessary actions, such as change the code to comply with a new requirement description that has just been changed. A user subscribed for change notification of a given individual will receive messages (email, so far) triggered when the Data Repository received a new version of that individual. Relationships to other individuals and values of properties are the main information presented to the subscriber.

V. USING THE ISDM FOR MANAGING SEMANTIC REQUIREMENTS DOCUMENTS

To illustrate the use of ISDM, a simple case in the requirements engineering domain is presented in this section. It encompasses template annotation, and data extraction inside a software project repository. The template we used is the requirements document template shown in Figure 5. It basically indicates the project name and lists its requirements. Figure 5 shows this template along with an exemplar instance of it.

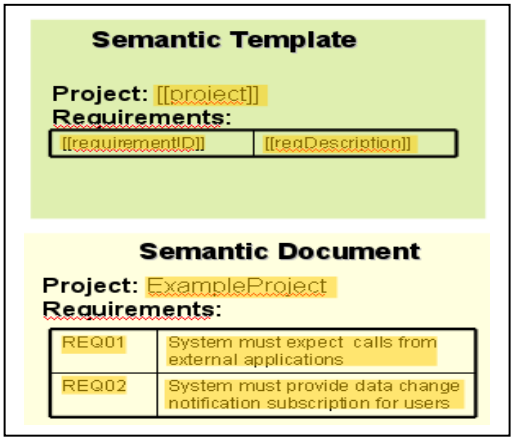

Figure 5. A Requirements Document template and an instance of it.

Figure 6 presents a fragment of the conceptual model of the software requirements ontology used as the domain ontology for creating the annotations. This ontology is presented in [23] and, for the purposes of this work, it was partially implemented in OWL and published locally in the following URL: http://localhost/ontologies/SE/onto.owl.

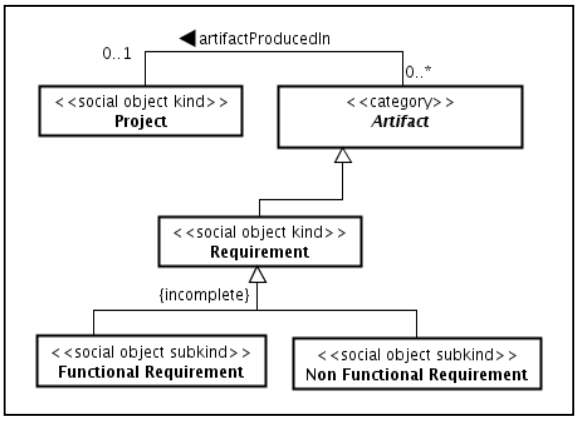

Figure 6. A fragment of the conceptual model of the Requirements Ontology used as the domain ontology for template annotation.

Defined the template structure and domain ontology to be used to annotate it, annotations were done in the template. The following text fragment annotation was defined to capture the project name in the first line of the document:

[[textspan]]instance({content},http://localhost/ontologies/SE/onto.owl#Project,$project);

When an analyst instantiates the template, she must overwrite the text into double brackets (*[[project]]*) by the corresponding content (in the case, the name of the project).

The following table annotations were defined to deal with the requirements table in the semantic template:

[[at0]]instance({content},http://localhost/ontologies/SE/onto.owl#FunctionalRequirement,$req);
property($req,http://localhost/ontologies/SE/onto.owl#artifactProducedIn,$project);
[[at1]]property($req,http://localhost/ontologies/SE/onto.owl#description,{content});

Following we present a resumed OWL representation of the first check in of the requirements document of the project *ExampleProject*. In this excerpt of the resulting graph, for sake of simplicity, we omitted several elements, such as class definition, property definition, disjoint class statements and data types.

```
<rdf:RDF …
   xmlns="http://localhost/ontologies/SE/onto.owl#" >
  <rdf:Description
rdf:about="http://localhost/ontologies/SE/onto.owl#Example
Project">
    <rdf:type
rdf:resource="http://localhost/ontologies/SE/onto.owl#Projec
t"/>
  </rdf:Description>…
  <rdf:Description
rdf:about="http://localhost/ontologies/SE/onto.owl#REQ001
">
    <description ...>System must expect calls from external
applications</description>
    <artifactProducedIn
rdf:resource="http://localhost/ontologies/SE/onto.owl#Exam
pleProject"/>
    <rdf:type
rdf:resource="http://localhost/ontologies/SE/onto.owl#Functi
onalRequirement"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://localhost/ontologies/SE/onto.owl#REQ002
">
    <description ...>System must provide data change
notification subscription for users </description>
    <artifactProducedIn
rdf:resource="http://localhost/ontologies/SE/onto.owl#Exam
pleProject"/>
    <rdf:type
rdf:resource="http://localhost/ontologies/SE/onto.owl#Functi
onalRequirement"/>
  </rdf:Description> …
```

Looking to this excerpt of the OWL file, it is possible to see the results of the extraction process. Two requirements, REQ001 and REQ002, were created along with their descriptions and were related to the newly created project, named ExampleProject.

In the sequel, we needed to update this document by adding a new row in the requirements table. The description of the new requirement, REQ003, was "System must provide version registration for document check in operations". We also have to change the description of requirement REQ001 to "System must provide a common API, allowing access through external systems". When these changed were checked in, a new version of this semantic document was generated. The resulting OWL file is presented in the next frame. Since the graph difference removes every node that exists on both graphs, things such as namespaces and class definitions are also cut off by the Jena framework. Thus, the difference graph displays only the new requirement (REQ003) and the description change on REQ001.

```
<rdf:RDF …
  <rdf:Description
rdf:about="http://localhost/ontologies/SE/onto.owl#REQ001">
    <j.0:description ...>System must provide a common API
allowing access through external systems </j.0:description>
  </rdf:Description>
  <rdf:Description
rdf:about="http://localhost/ontologies/SE/onto.owl#REQ003">
    <rdf:type
rdf:resource="http://localhost/ontologies/SE/onto.owl#Functio
nalRequirement"/>
    <j.0:artifactProducedIn
rdf:resource="http://localhost/ontologies/SE/onto.owl#Exampl
eProject"/>
    <j.0:description ...>System must provide version
registration for document check in operations
</j.0:description>
  </rdf:Description>
</rdf:RDF>
```

VI. Comparison with Related Works

As discussed in section III, during the development of this work, many related approaches have influenced our job. In this section we compare our work with some of the ones cited in section III.

Embley, Campbell and Smith [12] proposed a framework for performing data extraction from HTML pages and store them in a relational database, thus allowing further searches around the extracted information. One key aspect of their work is that it relied on text parsers in order to extract data. Although their work bring the idea of extracting semantic content from documents and filling up a relational database with it, somehow like our proposal does, they rely, basically, in parsing the document by using regular expressions and does not consider the idea of using semantic annotations directly on the document to support the extraction. Also their focus is on already existing documents, while we focus on providing ways to annotate template documents, making the annotation procedure more transparent for the end user.

Tallis [14] contributed with the construction of semantic document templates using a semantic annotation tool. One difference between his work and our work is that Tallis relied on a proprietary document format in order to produce semantic documents. He also relied on the web as the main document repository. Moreover, his work does not cover semantic document management as a whole, treating basically semantic annotation and automated information extraction. Our approach embraces the idea of semantic document templates and additionally focuses on keeping documents on a version control repository, which is inline with the reality of many software projects nowadays. Also our proposal intends to use an open document format (ODF) that is gradually been used in many government applications throughout the world [24].

The *Molhado* framework [10] is specialized in diminishing the impedance mismatch between versioned files contained in version control systems and the actual objects that they intend to represent. The main difference between *Molhado* and our approach is that the first is focused on specializing a version control system in order to provide semantic capability to it. Our work, on the other hand, aims to provide ways to enrich documents through the use of semantic document templates, extract their data and control their evolution. Additionally, we do not promote any changes to the file-oriented aspect of version control systems, since they are a de-facto standard.

Eriksson and Bang [7] and Kim et al. [13] proposed architectures to manage semantic documents that provide data indexing and searching capabilities. Similarly, ISDM allows searching. Moreover, ISDM combines the concept of semantic document template, initially proposed in [14], and the idea of tracking the evolution of extracted data, as proposed in [15]. Besides, a change notification subscription is proposed in order to provide change awareness.

Ognyanov and Kiryakov [15] proposed strategies for maintaining traceability in knowledge bases materialized as RDF(S) graphs. The approach used by them was considered very interesting and thus it was combined into ISDM. However, these authors focused mainly on providing ways to track changes in knowledge based and do not embrace the idea of managing semantic documents, which is the focus of our work.

The features proposed by our approach, such as annotating document templates, graph versioning and integration of semantic data, seems to be promising, but ISDM has also limitations and it needs to be tested in a production environment. The Semantic Annotation Module (SAM) is still in its early days and in the current version it does not provide ways to annotate several elements commonly used in documents, such as sections, images, lists, etc. Additionally SAM does not provide an easy way to annotate document templates. Thus, major work on usability must be carried on, in order to allow document engineers to better use this module. Finally, since ISDM has not yet been used in large scale, it is not possible to say if it does scale on large demanding environments.

VII. Conclusion and Future Works

This paper presented an Infrastructure for Semantic Document Management (ISDM). ISDM encompasses the following features: semi-automated and transparent annotations using the idea of semantic template documents (template documents with specific annotations); data

extraction and versioning, making use of a version control system and graph operations; change notification subscription for people that is interested on a particular content residing in the Data Repository.

Despite of the considerable number of opportunities within the work done so far, much work has to be done in the future.

The Data Extraction and Versioning Module (DEVM) only considers the main development line, known as 'trunk', of the Semantic Document Repository (so far a Subversion repository) for semantic information gathering. The model for registering version information must be extended to attend to any different development lines. These different lines of development are called branches in version control systems.

Although our annotation procedure is functional, it is better to use W3C standards, as Uren et al. [8] pointed out. The latest versions of the ODF specification (version 1.2) define ways to annotate office documents with RDF. This is particularly desirable, in order to provide some level of standardization. By the time the implementation of this work was being executed, the first version of an open source tool for editing office documents was being released. Open Office version 3.2 claimed to be fully compliant with ODF 1.2, but at the same time there was no Standard Development Kit (SDK) that allowed annotations. Later on, a library for RDF manipulation was added to the latest Open Office SDK [25]. With this new library, it is possible to be fully compliant to ODF 1.2 and use W3C standards as a whole. However, since the examples shown in [25] are entirely displaying situations where a document instance is being annotated, a challenge must be explored: how to maintain annotations at a document template level. In order to do that, our strategy of the semantic templating must be reviewed.

Other important point is to improve the change notification subscription. This feature can be enhanced to notify users not only when a given individual has changed, but also when a given situation occurs. For instance, a user may want to know when any activity is delayed. One way to address this scenario is to produce business rules and run them together with a reasoner in order to catch situations like that. Additionally a small front-end should be constructed in order to display user friendly notifications, searches and traceability results.

ACKNOWLEDGMENT

This research is funded by the Brazilian Research Funding Agencies FAPES (Process Number 45444080/09) and CNPq (Process Number 481906/2009-6).

REFERENCES

[1] Lethbridge, T. C., Singer, J., Forward, A., "How Software Engineers Use Documentation: The State of the Practice", IEEE Software, vol. 20, no. 6, pp. 35-39, Nov./Dec. 2003.

[2] Forward, A., Lethbridge, T.C., The relevance of software documentation, tools and technologies: a survey. Document Engineering. DocEng ’02, 2002.

[3] Bruggemann, B. M.; Holz K.P.; Molkenthin F. Semantic Documentation in Engineering, Proceedings of the Eighth International Conference held in Stanford, California, August 2000.

[4] Eriksson, H., The semantic-document approach to combining documents and ontologies, International Journal of Human-Computer Studies Volume 65 , Issue 7. 2007.

[5] Berners-Lee, T., Hendler, J., Lassila, O., The semantic web. Scientific American 284 (5), 2001, 34–43.

[6] Fensel, D., Hendler, J.A., Lieberman, H., Wahlster W. Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential. Mit Press. 2003

[7] Eriksson H., Bang M., Towards document repositories based on semantic documents, Proceedings of I-KNOW ’06. 2006.

[8] Uren, C. , Cimiano, P. , Iria, J. , Handschuh, S. , Vargas-Vera, M. , Motta, E. , Ciravegna, F. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. Web Semantic: Science, Services and Agents on the World Wide Web 4. 2005.

[9] Happel, H.; Seedorf, S.; Applications of Ontologies in Software Engineering, In 2nd International Workshop on Semantic Web Enabled Software Engineering . 2006.

[10] Nguyen, T. N. Object-Oriented Software Configuration Management. 22nd IEEE International Conference on Software Maintenance (ICSM'06). 2006.

[11] Estublier, J. , Garcia, S. Process Model and Awareness in SCM, in Proceedings of the 12th international workshop on Software configuration management, Lisbon, Portugal. 2005.

[12] Embley, D.W., Campbell, D.M, Smith, R.D. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents, at Conference on Information and Knowledge Management (CIKM 98), 1998.

[13] Kim H.L., Kim H.G., Decker S. Semantic Documentation using Semantic Web Technologies and Social Web Services, Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP'06), 2006.

[14] Tallis, M., Semantic Word Processing for Content Authors. Second International Conference on Knowledge Capture, Sanibel, Florida, October 2003.

[15] Ognyanov, D., Kiryakov, A., Tracking changes in RDF(S) Repositories, In Proceedins of EKAW’02. Spain. 2002.

[16] Collins-Sussman, B., Fitzpatrick, B.W., Pilato, C.M, Version Control with Subversion (For Subversion 1.5). Available at http://svnbook.red-bean.com/en/1.5/svn-book.pdf

[17] OASIS Open Document Format for Office Applications. Visited in April, 30th 2010. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office

[18] OpenOffice.org – The Free and Open Productivity Suite. Visited in April, 30th 2010. http://www.openoffice.org/

[19] McGuinness, D.L., Harmelen, F.V., OWL Web Ontology Language, W3C recommendation February 2004. Available at http://www.w3.org/TR/owl-features/

[20] Jena Framework website. Visited in April, 30th 2010. http://jena.sourceforge.net/

[21] The ODF Toolkit Project website. ODFDOM - the OpenDocument API. Visited in April 30th 2010. http://odftoolkit.org/projects/odfdom/pages/Home

[22] Prud’Hommeaux, E., Seaborne, A., SPARQL Query Language for RDF, W3C recommendation, January 2008. http://www.w3.org/TR/rdf-sparql-query/

[23] Falbo, R. A., Nardi, J., C., Evolving a Software Requirements Ontology. In: Proceedings of the XXXIV Latin-american Conference on Informatics - CLEI´2008, Santa Fé, Argentina, 2008. p. 300-309.

[24] ODF Alliance, ODF Annual Report 2008. Available at http://www.odfalliance.org/resources/Annual-Report-ODF-2008.pdf

[25] RDF metadata website. Visited in April, 30th 2010. http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/OfficeDev/RDF_metadata