A Semantic Backend for Content Management Systems

G.B. Laleci, G. Aluc, A. Dogac, A. Sinaci, O. Kilic, F. Tuncer

Software Research and Development Ltd.
Middle East Technical University (METU) Technoparc
06531 Ankara Türkiye
email: asuman@srdc.metu.edu.tr

Abstract

The users of a content repository express the semantics they have in mind while defining the content items and their properties, and forming them into a particular hierarchy. However, this valuable semantics is not formally expressed, and hence cannot be used to discover meaningful relationships among the content items in an automated way. Although the need is apparent, there are several challenges in explicating this semantics in a fully automated way: first, it is difficult to distinguish between data and the metadata in the repository and secondly, not all the metadata defined, such as the file size or encoding type, contribute to the meaning. More importantly, for the developed solution to have practical value, it must address the constraints of the Content Management System (CMS) industry: CMS industry cannot change their repositories in production use and they need a generic solution not limited to a specific repository architecture. In this article, we address all these challenges through a set of tools developed which first semi-automatically explicate the content repository semantics to a knowledge-base and establish semantic bridges between this backend knowledge-base and the content repository. The repository content is dynamic; to be able to maintain the content repository semantics while new content is created, the changes in the repository semantics are reflected onto the knowledge-base through the semantic bridges. The tool set is complemented with a search engine that make use of the explicated semantics.

1. Introduction

Content Management Systems (CMSs) are software applications for creating, publishing, editing and managing content. They are widely used by the news and media organizations, e-commerce websites, libraries, broadcasting and film industry, and educational institutions to handle the content efficiently. The

✩ The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no 231527

Preprint submitted to Elsevier
content used by a CMS is stored mostly in a content repository which is a hierarchical content store with support for structured and unstructured data. As the primary role of CMSs is to organize content items to make them accessible through intuitive queries; metadata assignment mechanisms are an important feature of CMSs [17, 9]. Several different means are available for adding structure and metadata to the content:

- Organizing content items as hierarchies allows users to relate content items and facilitate their discovery through the navigation of the hierarchies.

- Assigning properties to content items: The values of such properties can be set as free-format values, or they can be selected from constrained vocabularies. Different content items can also be related with each other through these properties. Finally, properties can be used to categorize content items.

- Constrained vocabularies such as taxonomies can themselves be represented as content item hierarchies, to categorize other content items by means of properties.

- Usually content repositories allow to define content types, which can be used to impose some structural restrictions on content item descriptions, such as the set of properties a content item may have, their possible ranges or cardinality restrictions. However the content administrators usually prefer to have “unrestricted” content types, so that the users can add arbitrary properties according to their need while creating content items. Hence content types are not always utilized by content administrators to add structure to content item definitions.

The content repositories enable structured queries to access content items as well as full-text search engines such as Lucene [3] are supported.

The major limitation of this content modeling approach and the associated search mechanisms is that it is only possible to query what is explicitly modeled by the content administrator. In other words, although the state-of-the-art
Figure 1: A Sample Content Repository Content

content management systems support metadata extraction to a degree that facilitates keyword-based search and content categorization; yet, enhancements are necessary for full alignment with the domain knowledge [2] by also considering the constraints of the CMS industry.

In this paper, we describe how to enhance the semantic capabilities of CMSs by lifting the already available semantics in content models as ontologies to be able to exploit implicit relationships among the content items for sophisticated content search and navigation mechanisms.

As a motivating example, consider a CMS used in a publishing house where articles and writers are represented as content items. Each article has a property, author, linking the articles with writers (Figure 1). Assume that this simple structure is lifted as a domain model and represented as an ontology, and a rule is defined to state that two writers are co-authors if they are the authors of the same article. With this rule, the query system may infer the co-author relationships between the writers, even if there is no explicitly defined co-author property. Assume further that, there is a news article taxonomy derived from IPTC News Subject Codes [23] that is maintained in the content repository, where the nodes in the taxonomy are created as content items in a hierarchy. The articles have a property category to relate them with these taxonomy nodes. When such a CMS is queried to find all “Health” related articles, only the content items that are explicitly annotated through the category
property that leads to this “Health” taxonomy node will be retrieved. When
the content model of this CMS is lifted as a domain model and represented as an
ontology, it is clear that through subsumption the articles that have not been
explicitly related with the taxonomy node “Health”, but related with the tax-
onomy nodes that are children of “Health” node in the content item hierarchy,
such as “Virus Diseases” and “Cancer” are also presented in the result set.

Through the semantic lifting mechanisms we are proposing, we explicate
an ontological representation of the content model of the content repository
that can be reasoned on, and that can be used for semantically enhanced value
added services such as search. However, the explicated semantics reflects what
is available in the content repository, the real potential of this semantic lifting
facility can be exploited when this extracted ontology is merged with external
domain and horizontal ontologies available in the Semantic Web. For example,
as a part of Linked Data Vision\cite{28}, there are 4.2 billion RDF triples, interlinked
by around 142 million RDF links, from various different domain and horizontal
ontologies, that are ready to be used for this merging facility. As a result of
such merging, we have much richer semantics to reason on and hence to present
more expressive and comprehensive searching capability to CMS users.

Consider the publishing house CMS example where the user queries the con-
tent repository to find the articles related with “Psychiatry” keyword. There is
an article about blumia in the content repository, which has been related with
the Eating Disorder taxonomy node from IPTC classification through the cat-
egory property. In IPTC, Eating Disorder taxonomy node is a subclass of the
Illness node, and there is no node “Psychiatry”. Assume this content model
is lifted as a domain model represented as an ontology, and merged with the
“MesH” biomedical ontology\cite{29}, where Eating Disorder class is a subclass of
Mental Disorder class which in-turn is a subclass of Psychiatry class. While
these ontologies are merged, the two Eating Disorder classes are set to be equivalentClasses. If we maintain this lifted ontology model in a triple store support-
ing inferencing, it is clear that the article related with blumia to be in our result
set through reasoning.
Meeting the Constraints of the CMS Industry

The value of lifting the already available semantics in the content models as an ontology is apparent, however our challenge is to achieve this within the constraints of the CMS industry. Some of the prominent CMS developers [1, 13, 32, 33, 35, 42] and their users do not want drastic changes in the repository that is already in production use. Our solution to this challenge is to have a backend knowledge-base and then develop semantic bridges between content repositories and the knowledge-base where the content repository semantics is maintained.

Equally important is to support a generic architecture that is not limited to a specific content repository. Several different content repositories have been developed with different APIs which do not interoperate. This problem is addressed through the Content Repository API for Java (JCR) [17] and Content Management Interoperability Services (CMIS) [9] which provide access to otherwise incompatible content repositories in a standard way; JCR through a java API and CMIS through Web Services and REST services. When a semi au-
tomated semantic lifting mechanism is built on JCR and CMIS Interfaces, a good percentage of the CMS market is covered. In addition to this, we have provided RESTful services that can be used by content repositories that do not support JCR or CMIS. Through RESTful services, the available content model and the updates can be sent to the semantic lifting mechanism so that an ontological representation of the content repository semantics can be created. This semi-automatic semantic lifting mechanism can be thought as a Semantic Bridge between a content repository and a knowledge-base storing the extracted semantics.

The proposed framework is implemented as a part of the IKS Project [22]. The extracted ontology is stored and maintained in the backend knowledge-base that supports both Jena Triple Store [25] and the Virtuoso Triple Store [43] as persistence mechanisms. Pellet [34] is used as the DL reasoner and Rule engine. A Search Engine is developed to exploit the extracted semantics.

The paper is organized as follows: Section 2 describes how to explicate the content repository semantics and the semantic persistence store implementation which is used to maintain the lifted semantics and to reason on it. In Section 3, a search mechanism that makes use of the lifted semantics is briefly described. In Section 4, related work is presented in comparison to our approach. Finally, Section 5 concludes the paper.

2. Semantic Back-end for Content Repositories

2.1. Semi-Automatic Explication of Content Repository Semantics

As already mentioned, the content management industry are eager to improve their business by using semantics, yet they do not want to modify their already in use content repositories drastically. Therefore the semantic annotation and reasoning must take place separately from their repository. Our solution to this challenge is to have a backend knowledge-base and then develop semantic bridges between content repositories and a knowledge-base to maintain the content repository semantics.
For semantic backend system to be generic enough to serve a wide range of content repositories, a common content repository model is needed. Fortunately, the repository models defined by JCR and CMIS provide a good basis to build upon as follows: Both JCR and CMIS define a hierarchical repository model. JCR calls the building blocks as nodes while CMIS calls them as objects. In each of them, repository items may have several properties, which may be assigned data type values, or other repository items. A repository item may include other repository items as child objects. Compared to JCR, CMIS has a more specialized model where objects may be a folder object, a document object, a relationship object or a policy object.

Apart from the repository items, both JCR and CMIS have specific node type or object type definitions through which the repository item definitions can be restricted, where the properties and child items they may have, can be restricted further. These types can be thought as template repository item definitions. Since our aim is to have a generic model that can be used by any content repository, we keep the common model as abstract as possible as depicted in Figure 3.

In this model, the first intuitive correspondence between the common repository model and an ontology is between object types and ontology classes. If the
content repository provides access through a common interface as in the case of JCR or CMIS, then our semantic lifting mechanism can easily create class definitions from object types automatically using the mapping relationships as presented in Figure 4. Once such class definitions are created, then the objects of these object types in the content repository can readily be represented as individuals of these classes.

However because of the real-life usage practices in content repositories, transforming object types into ontology classes is not sufficient to explicate the content repository semantics. Content repositories usually do not differentiate among data and metadata: both repository items that represent the actual content and the repository items created to classify other content items are all created as objects of a certain object type. When the data, metadata differentiation does not exist in the repository, exposing this as semantics is not very helpful.

Consider for instance the previous publishing house content repository example (Figure 1). In this repository, a news article taxonomy derived from IPTC News Subject Codes [23] is maintained in the repository by representing the nodes in the taxonomy as objects in a hierarchy. The article objects of type articleType have a property called category, to relate them with the nodes of
this taxonomy represented as objects. Assume an article object which is related with the “Health” object. From an ontological perspective, this model is best represented if a set of classification classes are created from the object hierarchy representing the IPTC taxonomy, i.e. as a result there will be a class named as Health. Then the specific article related with the Health object which is created as an individual of its object type’s class in the ontology (articleType Class), is assigned a second rdf:type as the Health class.

However such an ontological representation cannot be created in a purely automated way without a priori configuration: The content administrator should specify that the object tree in the repository created to represent IPTC codes is in fact a classification object tree, and while the articleType object type is created, it should be specified that the category property has a specific semantics,
i.e. *classification*. To be able to specify such semantics, we need a configuration to be defined so that semantic lifting can be achieved.

In our architecture we provide two options for this:

- **Configuration for Content Repositories supporting JCR or CMIS:** Both JCR and CMIS enable accessing the content repository model through standard interfaces, and accessing to the selected repository items through a query language. We have developed a graphical tool, through which the content administrator can visualize the content model, and create queries for selecting *classification objects*, *content objects*, *properties* and mapping these selected repository items to ontological constructs by creating semantic bridges among them. This graphical mapping process produces *bridge definitions* and the *semantic lifting mechanism* processes these bridge definitions to create the corresponding ontological representation by querying the content repository when necessary. The bridge definition is also used to keep the content model and the extracted ontological representation synchronized: whenever a change in the content model is reported to the *semantic lifting mechanism*, it in-turn checks the bridge definitions to update the knowledge-base if necessary. In this way, the content administrator needs to configure the *semantic lifting mechanism* once graphically, then the *semantic lifting mechanism* automatically keeps the ontological representation up-to-date. This lifting mechanism is presented in Section 2.1.1 in more detail.

- **Configuration for Content Repositories that do not support a standard API:** Although JCR and CMIS provide enabling services to facilitate interoperability of content repositories, still there are many content repositories which use proprietary interfaces. In order to support these repositories through our *semantic lifting mechanism*, we provide a number of RESTful services so that such repositories can declaratively feed semantics to our backend knowledge-base. This alternative lifting mechanism is presented in Section 2.1.2 in more detail.
2.1.1. Semantic Lifting in Content Repositories supporting JCR or CMIS

In order to explicate the semantics of a content repository supporting standard interfaces semi-automatically we developed a GUI as shown in Figure 5. The user is provided with four graphical semantic bridge constructs as well as a graphical query mechanism. As the user drags and drops, indicating the correspondences between the selected repository item or item collections and the semantic bridge constructs; the native content repository queries are automatically generated by the system along with the bridge definitions in XML. After retrieving the object types and processing them according to the mapping rules specified in Figure 4, the semantic lifting mechanism processes these bridge definitions, and the ontological representation of the repository content model is enriched with the new axioms.

There are four semantic bridge constructs defined in XML as follows:

• ConceptBridge: This bridge allows the content administrator to annotate the graphically selected objects in the content repository as classification objects. As a result of this bridge execution, an ontology class or an ontology class hierarchy is created corresponding the selected objects. For example, in Figure 1, the tree rooted at the “NewsSubjectCodes” object represents metadata and therefore its child objects can be selected through a ConceptBridge to create a class hierarchy in the ontological representation of the content model. A complete hierarchy in the repository can be graphically selected to correspond to a class hierarchy (Figure 5) or any object in the hierarchy can be selected to correspond to a class in the ontology. Once the objects are selected, the corresponding native JCR and CMIS queries are automatically generated by the tool and saved in the bridge definition. Additionally, if the user wishes, s/he can write any query on the repository and may request a corresponding ontology class to be created.

• SubsumptionBridge: This bridge allows users to specify the properties in a content repository that correspond to subsumption relationships in the
ontology. For example, for the “Tags” tree shown in Figure 1, the property called “broader” represents such a class/subclass hierarchy in the content repository, or the “parent/child” relationship between objects in the “NewsSubjectCodes” tree may be chosen to represent a class/subclass hierarchy. A Subsumption Bridge has two elements: objectQuery element is used to specify the objects as “SuperClasses”, and the predicateName specifies the property name that will be used to select the “SubClasses” of these objects selected as “SuperClasses”. If a SubsumptionBridge is used within a ConceptBridge, the subsumption relationship is established automatically among the objects qualified by the ConceptBridge and hence there is no need to define an objectQuery.

- PropertyBridge: In order to selectively “lift” some of the content repository properties to the ontological representation being created, PropertyBridge construct is used. This is necessary, because not all of the properties defined for an object in the content repository may have a semantic value; some of them can be syntactic attributes such as the file size. Property Bridge helps to selectively lift the semantics of properties. Similar to Subsumption Bridge, the Property Bridge has two elements: the objectQuery element is used to specify the set of objects to which this PropertyBridge is applied to. The selected objects are set as the “domain” of the ontology property that is to be generated. The predicateName element gives the name of the property in the content repository to be used to select the range of the ontology property. Note that a property may refer to another object, or to a data value. If the property is referring to another object, then an OWL “ObjectProperty”, else a “DataTypeProperty” is created in the ontology. If the PropertyBridge is specified within a ConceptBridge, then there is no need for an objectQuery; the selected ontological properties of all the objects qualified by the ConceptBridge are established automatically.

Finally, for each PropertyBridge, the user can define an annotation to
Figure 6: A sample Content Repository and a part of the Extracted Ontology

specify additional semantics: this is used by the bridge processor to set the type of the OWL property such as the functional, symmetric, transitive or inverseFunctional OWL property. Another point to note is that, the repository users may specify object equivalences using the object properties which correspond to class expressions in OWL such as “equivalentTo” or “disjointWith”. This semantics is also explicated through property annotations in the PropertyBridge.

• InstanceBridge: This bridge is used to select the objects in the repository that are actually used for storing content items, i.e. to select content objects. For example, in Figure 1, the objects in the “Article” tree can be graphically selected using an InstanceBridge as content objects since they represent actual data items rather than metadata. Each selected object is created as an individual of the class created to represent its object type. For example, the type of the objects in the “Article” tree is “articleType” and the selected objects are made instances of the ontology
The selected objects may be related to other objects that are specified as classification objects. For example, the article objects under “Article” tree are related with objects under the “NewsSubjectCodes” tree through “category” property. From an ontological perspective, this kind of annotation is best represented by making the selected article object also an individual of the classes created to represent the classification objects. To be able to express such relationships, in each InstanceBridge, it is possible to add a PropertyBridge through which a content repository property name can be specified. This property is annotated as “classification”, to indicate that it in fact represents an “rdf:type” relationship in the ontology to be created.

In Figure 5, the left bottom pane of the interface allows the user to browse the content repository. Semantic bridges can be selected on the top pane of the interface, and the repository items and the properties can be dragged and dropped from the repository browser to the “Bridge Definition” Windows. For example, in the figure, the user has selected the “Concept Bridge Definition Window” and has dragged and dropped the “NewsSubjectCodes” object, and indicated that this node is a “tree root”. This causes the semantic lifting mechanism to automatically generate the corresponding bridge definition and the involved query on the repository. The bridge definitions are created as XML files and are processed to extract the ontology definition. The extracted ontology is stored to the backend knowledge-base.

The result of this process is shown through an example in Figure 6. The NewsSubjectCodes taxonomy of Figure 1 are automatically converted to ontology classes, and the Articles that are linked to the taxonomy objects with the “category” property became the instances of these Ontology classes.

A content repository has dynamic content; the users continue to create/update/delete the object types, objects and the properties. Unless these changes are reflected to the lifted ontology, the tool set developed will be of limited value. In JCR
there is an event notification mechanism which informs the subscribers whenever a node or property is added/updated/deleted. However there is no such mechanism in CMIS. To facilitate this kind of event notification, our semantic lifting mechanism opens RESTful interfaces, through which the content repositories just report the unique references of the object types/objects/properties added/updated or deleted. The bridge processor in turn queries the content repository when necessary to retrieve further details; checks the bridge definitions whether these recently reported repository items are covered in them and updates the knowledge-base as necessary.

2.1.2. Semantic Lifting in Content Repositories without Standard Interfaces

There are many content repositories which do not support standardized interfaces such as JCR or CMIS. These repositories provide access to repository content model and repository items through proprietary interfaces. It is apparent that our semantic bridging methodology presented in Section 2.1.1 will not work for these repositories: first of all it is not possible to use a standard query language to select the classification or content objects, and it is not adequate to have such queries in bridge definitions since there is no standard interface to access the content repository. We address this challenge by developing a mechanism to give a dump of repository items to the semantic lifting mechanism, along with the categorization of the repository items as classification and content objects. Apart from this, when a repository item is added/deleted or updated, for content repositories supporting JCR or CMIS interfaces, it was enough to inform the semantic lifting mechanism only about the identification of the repository item affected. The semantic lifting mechanism would check the bridge definitions, and if the added/deleted/updated repository item satisfies any of the queries in the bridge definitions, then the ontology is updated accordingly. However for content repositories that do not support standard interfaces, there should be interfaces to report the repository item definition that is added/deleted/updated along with its semantics i.e. whether it is a classification or a content object.
In order to address these requirements, we provide a number of RESTful services so that such repositories can declaratively feed semantics of repository items to the *backend knowledge-base*:

- Add/Delete/Update Object Type Definition: This interface allows content repositories to inform the *semantic lifting mechanism* about added/deleted/updated *object types* together with the complete definition of the *object type*. In response, the *semantic lifting mechanism* creates/removes/updates the corresponding ontology classes in the *backend knowledge-base*.

- Add/Delete/Update Property Definition: This interface allows content repositories to inform the *semantic lifting mechanism* about added/deleted/updated *property definition* for a specific *object type*. Together with *property definition* information, the semantic annotations of the property such as transitive, symmetric, functional, inverseFunctional, equivalentClass, disjointClass, subsumption and classification are also specified. In response, the *semantic lifting mechanism* creates/removes/updates the corresponding ontology properties of the selected classes in the *backend knowledge-base*.

- Add/Delete/Update Classification Object: This interface allows content repositories to inform the *semantic lifting mechanism* about added/deleted/updated *object* that are used for *classification* of other *objects*. In response, the *semantic lifting mechanism* creates/removes/updates the corresponding ontology classes. The object definitions can be nested within each other, in this case the nesting implies subsumption relationship between the ontology classes to be created. Then the properties of the object are checked to see whether an equivalentClass or a disjointWith semantic annotation has been defined for them while the related property definition has been registered. If there are such annotations, then the related equivalentClass or a disjointWith class expressions are added or deleted between the class of this reported *object* and the corresponding class of the *object* specified as the range of the property.
• Add/Delete/Update Content Object: This interface allows content repositories to inform the semantic lifting mechanism about added/deleted/updated object that are used for storing actual content. In response, the semantic lifting mechanism creates/removes/updates the corresponding individuals. Their rdf:type is set as the ontology class created for their object type definitions. The object definitions can be nested within each other, in which case the contains property is set between the corresponding individuals. Then the properties of this object are checked to see whether a classification semantic annotation has been defined for any of them while the related property definition has been registered. If there are such annotations, then rdf:type expressions are added or deleted between the class of this reported object and the corresponding class of the object specified as the range of the property.

• Add/Delete/Update Property: This interface allows content repositories to inform the semantic lifting mechanism about added/deleted/updated properties for a specific object. The semantic lifting mechanism checks whether this is a classification or a content object. If it is a content object, then the semantic lifting mechanism sets/removes/updates the corresponding ontology properties of the selected individuals in the backend knowledge-base. If the reported property has a classification semantics, then the rdf:type, assertion is set or deleted for this individual with the class corresponding to the specified classification object. If the reported property belongs to a classification object, and if this property has an equivalentClass or a disjointWith semantics, then the owl:equivalentClass or owl:disjointWith assertion is set or deleted between the classes created for the corresponding classification objects.

• Add/Delete/Update Property Semantics: This interface allows the content repository to add/delete/update the semantic annotations of a selected property definition such as transitive, symmetric, functional, inverseFunctional, equivalentClass, disjointClass, subsumption and classifi-
cation. In this case, the *semantic lifting mechanism* checks the ontology and the necessary updates in the ontology are achieved such as changing the property types, adding/deleting `rdf:type`, `owl:equivalentClass`, `owl:disjointWith`, `rdfs:subClassOf` assertions.

Using these interfaces it is possible to keep the semantics of the content repository synchronized with its ontological representation maintained in back-end knowledge-base.

The “Add/Delete/Update Object Type Definition”, “Add/Delete/Update Object”, and “Add/Delete/Update Property” interfaces can also be used by content repositories that support standard interfaces just to inform the *semantic lifting mechanism* about such events as presented in Section 2.1.1.

2.2. Backend Knowledge-base for Content Repositories

As a result of semantic lifting mechanism, we have the ontological representation of the content repository semantics, which will be kept synchronized with the changes in the repository. This ontology is maintained in a knowledge-base that supports a triple store for persisting the ontology. We have investigated
the open source triple stores Jena [25], Sesame [6] and Virtuoso (Open Source version) [43]. Both Jena and Sesame provide for scalable storage and query of RDF datasets using conventional SQL databases. Sesame supports built-in RDFS reasoner, while Jena supports reasoning for RDFS and OWL Lite. However it has been reported that both of these have poor performance [20]. Yet Jena supports DIG interface [15] to communicate with external reasoners and Pellet [34] can easily be used in integration with Jena. Pellet is a reasoner based on well-known tableau algorithms for TBox and ABox reasoning, supporting SHOIQ [21] (the expressiveness of OWL-DL plus qualified cardinality restrictions in DL terminology). Pellet reasoning capability has been reported as resilient to nontrivial, expressive ontologies, however when data sets especially the number of ABoxes increases, Pellet experiences a performance problem and time-outs [7].

Virtuoso, on the other hand, is a native triple store available in both open source and commercial licenses. In the evaluations it is found out to be scalable to the region of 1B+ triples [38]. Virtuoso has been used for hosting many of the data sets in the Linking Open Data Project [28], including Dbpedia [14]. However Virtuoso has a less expressive reasoner, only supporting a limited set of expressions including: rdfs:subClassOf and rdfs:subPropertyOf, owl:sameAs, owl:equivalentClass and owl:equivalentProperty. It does not materialize all implied triples but rather looks for the basic facts implying these triples at query evaluation time.

Considering the pros and cons of different triple stores, we have decided to provide a generic interface to communicate with triple stores, so that independent of the triple store hosted at the backend knowledge-base, the semantic lifting mechanism can feed and query ontologies hosted. We have implemented RESTFul interfaces to create/update/delete ontologies, classes, properties, individuals. Currently these RESTful services have been bound to Jena and Virtuoso triple stores. Depending on the complexity and size of the data set, the triple store to be used can be configured. This generic triple store interface also enables us to integrate other more scalable triple stores that supports rea-
soning for more expressive ontologies without changing the implementation of the semantic lifting mechanism.

The semantic lifting mechanism stores the extracted ontological representation of the content repository to the backend knowledge-base. This ontology is an isolated representation of the content repository itself, and if it can be aligned/merged with well established external domain ontologies, then the semantic capabilities of the CMSs will be enhanced. To support this, we have implemented a number extensions to the Protege Tool [36]. Through these extensions, it is possible to select our knowledge-base as an ontology store, load the ontologies maintained in the persistence store, edit, update them, merge with external ontologies, and save them back to the knowledge-base.

3. Using the Extracted Semantics in Content Discovery

After having these semantic enhancement mechanisms in place for content repositories, the next step is to exploit them for content discovery. Again, the aim is to be the least disruptive to the current practices in the CMS industry. Examining the current search techniques used in the CMSs, we build a search interface similar to the available faceted search mechanism. The idea is to present different views on the results such as possible similar semantic content categories, similar keywords, and semantic links between the documents. In
this way, the search engine allows the search scope to be iteratively changed or narrowed until the user is satisfied with the results. The user is never expected to write a “SPARQL” query; he just navigates through the faceted search options, the system queries the knowledge-base when necessary and presents more comprehensive results than that could be possible through a full text search engine.

The steps taken by the search engine can be summarized as follows:

- The user specifies a free text keyword to the search engine. First, a keyword based search is done to collect the content items indexed by a full-text search engine, in our case Lucene.
- In the meantime, the search engine queries the knowledge-base and for each of the content item returned in the result set of Lucene, retrieves the
corresponding individuals. Then the ontology classes related with these individuals through rdf:type property are retrieved from the knowledge-base. While presenting the resulting content items to the users, these retrieved ontology classes, together with their superclasses/subclasses are also presented as related semantic categories in a navigable form as illustrated in Figure 8. A screenshot from the actual search GUI is presented in Figure 9.

• The user is free to navigate the explicated ontology presented together with the resulting document set, and the search scope can be restricted accordingly. Assume, for example that the user searches with the terms “drug” and “collapse” with the “addiction” context in mind. As a result of the full-text search, a number of results are returned, as shown in Figure 8, including an unrelated resource such as “China’s anti-malaria medicine producers face market collapse” along with the related resources such as “arrests after boys’ drug collapse”. However, now the user can see not only the result set but also a navigable ontology that is composed of the semantic categories of the resulting documents and their vicinity classes. If the user is interested in “drug collapse” incidents related with “drug addiction”, he can select the “addiction” semantic category and discard the “pharmaceutical” category. In this way the “context” of the search can be easily specified. As a result the search focus is narrowed to this context and hence the unrelated content items are eliminated.

• Based on user’s preferences, the extracted ontological representation of the content repository can be used more effectively to enhance the initial result set. After the keyword based search is finalized, the semantic categories of the resulting documents and their subclasses/superclasses, or the ontology resources related with these semantic categories through object properties, can be used to query the knowledge-base, to retrieve “semantically similar” content items. Assume the user searches with “bone marrow” and “cancer” keywords. As a result of the full-text search an article
is found to be related since these keywords are available in the article in a statistically significant manner. This article is tagged with “leukemia” in the content repository, hence is created as an individual of the “leukemia” class in the knowledge-base. As a result of this, through a query to the knowledge-base, all the other articles in the content repository tagged with “leukemia” can also be included in the result set. On top of this, consider the case where the extracted semantics of the content repository is merged with a biomedical ontology, where “leukemia” class is related with “antileukemic drugs” class with an object property. The articles annotated with “antileukemic drugs” may also be presented to the user, due to the semantic relation between the “leukemia” and “antileukemic drugs” in the ontology. In other words, the ontology classes in a vicinity of one another may provide useful semantic information to the user. The challenge here is to be able to provide related ontological sources without overwhelming the user. Our solution is to set configurable thresholds for calculating the vicinity set of an ontology resource, i.e., how many superclass/subclass, objectProperty links should be followed to enrich the ontology resource set that will be used to query the knowledge-base to retrieve the relevant contents. Based on the vicinity of the ontology resource to the semantic category, the resulting documents are ranked to give the user a semantic similarity measure.

- Additionally, the user defined keywords are queried from the extracted ontology using string similarity metrics [41], and the search results are extended with the content items that have been annotated with lexically similar ontological resources (classification objects in the content repository). Hence it becomes possible to retrieve related content that the full-text based search engines have not indexed.

Our ultimate aim is to improve the discovery mechanisms of content repositories by practical semantic means that they can use in production. Another such mechanism is to enrich the search keywords. A horizontal ontology such as
WordNet [45] or DBpedia [14] can be used to discover the resources semantically related to the keywords provided by the users. The original set of keywords is then expanded with those retrieved from the ontology look-up. The search process proceeds as purely full-text querying of the indexed documents with the expanded keyword set as the new input. In Figure 9, the user gives “drug” and “factory” as keywords and a set of synonymous and related terms (such as hyponyms, hypernyms) are retrieved from the WordNet Ontology. Once the keyword set is expanded with these terms, further meaningful results can be obtained. For example, the document entitled: “Cannabis factory found by police” is also retrieved since “cannabis” is a hyponym of the “drug” term in WordNet Ontology. Similar to the WordNet, the DBpedia ontology can be queried to enhance the keyword set. DBPedia is an initiative to realize the Linked Data vision [28] and it is possible to locate resources linked to a given keyword, by making use of Wikipedia Infoboxes. For example, given a keyword “German Chancellor”; the “Angela Merkel” resource can be easily located from the DBPedia, since these two resources are semantically associated with each other in the DBPedia ontology and this new keyword can be used in the faceted search. The user is also allowed to decide whether to include such additional keywords directly to the search process, or they can graphically select the ones to be included in the next step of the search as shown in Figure 9.

4. Related Work

There is a growing amount of research on semantically enhanced discovery mechanisms although only a few of them address the CMSs. [18, 30] describe mechanisms for building semantically enhanced CMSs from scratch. Our approach complements this by focusing on the commercially available CMSs in production use and their constraints. We provide mechanisms to explicate the already available semantics in “content repositories” used by the CMS industry through a semantic backend, and exploit this semantics in the search engine developed. Our main concern while developing these mechanisms
was their practical set. Therefore, we have concentrated on the constraints set
by the industry in improving their current usage patterns rather than starting
from fresh.

Although not directly addressing CMSs, in [26, 40], semantic, ontology-based
“Web site” and “Web portal” management systems are proposed, where the Web
sites and portals run on native ontology management systems with RDF stores
as backends.

[4] and [27] propose mechanisms to automatically categorize content items
within a CMS to build a semantic content management system. Our focus is not
processing different types of content such as documents or videos in order to ex-
tract semantics from these content items to categorize them; rather, we address
lifting the semantics in the content model (the content organization, content
metadata given by content administrators, and the content types). These two
approaches are complementary and in fact the semantic lifting process we pro-
pose in this paper will be integrated with a semantic document analysis tool
[27], within the scope of the IKS project [22].

The closest study to our work is [10], where semantic enhancements to Dru-
pal CMS [16] are addressed. This work proposes to automatically create on-
tologies from Drupal Content Construction Kit (which can be thought as JCR
node types) and provides a SPARQL end point for the Drupal. In [11], a similar
approach to [10] is proposed for supporting SPARQL queries on a Drupal CMS
for a biomedical application. Our work complements these work as follows: first,
to be able to serve to a wider CMS community, we base our enhancements on
JCR and CMIS that can be used as common interfaces for accessing different
CMS implementations. We also provide RESTful services for content reposito-
ries that do not support such standard interfaces. Secondly, we do not restrict
semantic extraction only to predefined object/node types because in the CMS
industry the use of “node/object types” are not very common; content admin-
istrators usually prefer to create “unrestricted” object/node types, where CMS
users can add arbitrary properties to the objects at run time. Finally, through
the semantic bridge facility, we were able to cover the mechanisms already used
by the industry for adding semantics to the content, such as content hierarchies, 
tag hierarchies, and links between content items. For this purpose, we have de-
vised semi-automatic mapping facilities so that the user can selectively “lift”
the semantics of a content repository to an ontological representation.

Although not directly related with CMSs, there are important research on
enriching Wiki’s capabilities for semantic search and navigation by extending
Wikipedia’s existing category system, typed links among articles and assigning
attributes for some properties in an RDF-based format [8, 31, 44, 37]. ES-
TER [5] combines full-text search with ontology search for resource discovery
in Wikipedia using entity recognition algorithms and SPARQL queries over the
YAGO ontology [46] integrated with Wikipedia. Wikis can be thought of as
content management systems in a broader sense, however the underlying infra-
structure of Wikis and CMSs are completely different, hence although the aim
is similar, the required mechanisms for semantically enriching Wikis and CMSs
are different.

Finally, there has been quite a lot of research on integrating keyword based
full-text search mechanisms with ontological search methods to enable semantic
search of Web content. QuizRDF [12] proposes a search technique for Semi-
tatic Web, where the content resources are ontologically indexed through the use
of full text analysis and domain ontologies. It also proposes a faceted search
method on top of this index. In TAP [19], a semantic search system is presented
for Semantic Web, where the traditional search results are augmented with fur-
ther semantics. SCORE [39] also proposes a semantic search engine for the
Web content, where semantic metadata is extracted as ontologies from struc-
tured and semi-structured text by software agents and queried with a semantic
ingine where it is also possible to employ full-text indexes optionally. All these
efforts address the semantic search of Web resources, and do not directly address
semantic management of content items in contemporary “Content Management
Systems”.
5. Conclusions

In this work, we describe how a content management systems can be supported through a backend knowledge-base to semantically enhance the content annotation and discovery by also addressing the following challenges:

- The users of a content repository express the semantics they have in mind while defining the nodes and properties among the content items and forming them into a particular hierarchy. However, unless this valuable semantics is not formally expressed, it cannot be used in an automated way to discover meaningful relationships among the content items. We describe a semantic lifting mechanism for CMSs to express the already available semantics in their content models as ontologies and make use of reasoners and rule engines on top of this ontology. This provides the shared understanding of semantics of the content model as a formal and machine manipulable vocabulary. Furthermore, it becomes possible to model the constraints capturing additional knowledge about the content model through rules. In this way, the content model becomes not only machine processable but also \textit{machine understandable}, i.e., the implicit facts are deduced from explicitly stated facts through reasoners.

- The developed mechanisms must meet the constraints of the CMS industry to have practical value. Although the content management industry wants to improve their business by using semantics, understandably so they do not want to modify their content repositories in use drastically. Therefore, the semantic annotation and reasoning must take place separately from their repository. Our solution to this challenge is to have a backend knowledge-base and then develop semantic bridges between content repositories and the knowledge-base where the content repository semantics is maintained.

- There is a wide range of content repository systems and a proprietary solution to serve only a few them will not be of significant value. Our solution
to this challenge is to develop a generic content repository model based on the Content Repository API for Java (JCR) [17] and Content Management Interoperability Services (CMIS) [9]. When a semi automated semantic lifting mechanism is built on JCR and CMIS Interfaces, a good percentage of the CMS market is covered. In addition to this, we have provided RESTful Services that can be used by content repositories that do not support JCR or CMIS.

The semantic backend described in this article is being used within the scope of the Interactive Knowledge Stack [22] which is an Integrating Project funded by the European Commission where the end users are some of the prominent CMS providers [1, 13, 32, 33, 35, 42].

Within the scope of the project, another semantic lifting component [27] is being implemented to extract semantics from text-based content items based on the Named-Entity Recognition methods. As a future work, this component will be incorporated to the semantic back end presented. In this way, it will be possible to exploit the semantics available in the documents themselves as ontologies.

References


[22] Interactive Knowledge Stack (IKS) Project, http://www.iks-project.eu/


[31] Muljadi H. and Hideaki, T. “Semantic wiki as an integrated content and metadata management system”. In Poster Session at the ISWC 2005, Nov 2005


[34] Pellet, http://clarkparsia.com/pellet/


