ENHANCING CONTENT MANAGEMENT SYSTEMS WITH SEMANTIC CAPABILITIES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SUAT GÖNÜL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2012

Approval of the thesis:

ENHANCING CONTENT MANAGEMENT SYSTEMS WITH SEMANTIC CAPABILITIES

submitted by **SUAT GÖNÜL** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Nihan Kesim Çiçekli
Supervisor, **Department of Computer Engineering, METU** _____

Prof. Dr. Asuman Doğaç
Co-supervisor, **SRDC Ltd.** _____

Examining Committee Members:

Prof. Dr. Fazlı Can
Computer Engineering Department, Bilkent University _____

Prof. Dr. Nihan Kesim Çiçekli
Computer Engineering Department, METU _____

Prof. Dr. Özgür Ulusoy
Computer Engineering Department, Bilkent University _____

Assoc. Prof. Pınar Şenkul
Computer Engineering Department, METU _____

Assoc Prof. Ahmet Coşar
Computer Engineering Department, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SUAT GÖNÜL

Signature :

ABSTRACT

ENHANCING CONTENT MANAGEMENT SYSTEMS WITH SEMANTIC CAPABILITIES

Gönül, Suat

M.S, Department of Computer Engineering

Supervisor : Prof. Dr. Nihan Kesim Çiçekli

Co-Supervisor : Prof. Dr. Asuman Doğaç

July 2012, 77 pages

Content Management Systems (CMS) generally store data in a way that the content is distributed among several relational database tables or stored in files as a whole without any distinctive characteristics. These storage mechanisms cannot provide the management of semantic information about the data. They lack semantic retrieval, search and browsing of the stored content. To enhance non-semantic CMSes with advanced semantic features, the semantics within the CMS itself and additional semantic information related with the actual managed content should also be taken into account. However, extracting implicit knowledge from the legacy CMSes, lifting to a semantic content management system environment and providing semantic operations on the content is a challenging task which includes adoption of several latest advancements in information extraction (IE), information retrieval (IR) and Semantic Web areas. In this study, we propose an integrative approach including automatic lifting of content from legacy systems, automatic annotation of data with the information retrieved from the Linked Open Data (LOD) cloud and several semantic operations on the content in terms of storage and search. We use a simple RDF path language to create custom, semantic indexes and filter annotations obtained from LOD cloud in a way that is eligible for specific use cases. Filtered annotations are materialized along with the actual content of

document in dedicated indexes. This semantix indexing infrastructure allows semantically meaningful search facilities on top of it. We realize our approach in the scope of Apache Stanbol project, which is a subproject developed in the scope of IKS project, by focusing on document storage and retrival parts of it. We evaluate our approach in healthcare domain with different domain ontologies (SNOMED/CT, ART, RXNORM) in addition to DBpedia as parts of LOD cloud which are used annotate documents and content obtained from different health portals.

Keywords: Semantic content management, Semantic storage, Semantic search, Semantic Web

ÖZ

DÖKÜMAN YÖNETİM SİSTEMLERİNİ ANLAMSAL YETENEKLERLE GELİŞTİRME

Gönül, Suat
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Nihan Kesim Çiçekli
Ortak Tez Yöneticisi : Prof. Dr. Asuman Doğaç
July 2012, 77 sayfa

Döküman yönetim sistemleri içeriği genel olarak birçok veritabanı tablosuna dagıtılmış olarak veya dosyalarda bir bütün olarak, herhangi bir ayırt edici özellikleri olmadan saklarlar. Bu depolama mekanizmaları içerikle ilgili anlamsal bilginin yönetilmesini ve böylece depolanmış dökümanların üzerinde anlamsal bilgi çekme, arama gibi işlevleri sağlayamazlar. Anlamsal açıdan yetersiz döküman yönetim sistemlerine gelişmiş anlamsal kabiliyetler kazandırmak için bu sistemlerde depolanan içeriğe ek olarak, sistemin içindeki üstü kapalı anlamsal bilginin ortaya çıkarılması ve zaten var olan içeriğin harici kaynaklardan toplanan, asıl içeriğin kendisiyle alakalı ek bilgilerin dikkate alınması gerekir. Ancak, hali hazırda var olan döküman yönetim sistemlerinden açıkça ifade edilmeyen bilgiyi çıkarmak, bu bilgiyi bir anlamsal döküman yönetim sistemi ortamına aktarmak ve içeriğin üzerinde anlamsal operasyonlar sağlamak; bilgi çıkarma, bilgi çekme ve anlamsal ağ alanlarından bir çok yeniliklerin benimsenmesini gerektiren zorlu bir iştir. Bu çalısmada, içeriğin var olan sistemlerden otomatik olarak çekilmesini, çekilen içeriğin acık bağlantılı veri üzerinden çekilen bilgilerle zenginleştirilmesini; ve depolama ve arama bakımınından birçok anlamsal işlevsellik sağlayan metodolojiler geliştirilmektedir. Önerilen yaklaşımda isteğe uyarlanmış, anlamsal indeks oluşturmak için basit bir RDF yol dili kullanılmaktadır. Bu dil aynı zamanda acık

bağlantılı veri üzerinden elde edilen ek bilgilerin özel amaçlı kullanım durumlarına göre filtrelenmesi için de kullanılmaktadır. Filtrelenmiş ek bilgiler dökümanların asıl içerikleriyle beraber özel kullanım için hazırlanmış indekslerde somutlaştırılır. Bu anlamsal indeks alt yapısı anlamsal olarak değeri olan arama işlevlerinin sağlanmasına olanak verir. Önerilen metodolojiyi IKS projesinin yazilim ürünlerinden biri olan Apache Stanbol projesi kapsamında hayata geçirilmektedir. Bu esnada Apache Stanbol'un depolama ve arama kısımlarına ağırlık verilmektedir. Çalışmanın son aşamada önerilen metodoloji sağlık alanında çeşitli sağlık portallarından alınan dökümanlarla değerlendirilmektedir. Bu işlem sırasında bağlantılı veri bulutunun bir parçası olan DBPedia ve sağlıkla ilgili üç ontoloji kullanılmaktadır. Bunlar sırasıyla SNOMED/CT, ART ve RXNORM'dur ve bu üç ontoloji dökümanlara sağlıkla ilgili ek bilgi iliştirmek için kullanılmaktadır.

Anahtar Kelimeler: Anlamsal döküman yönetimi, anlamsal depolama, anlamsal arama, anlamsal ağ

To my dearest brother Samet...

ACKNOWLEDGMENTS

I would like to express my candid gratitude and appreciation to my supervisor, Prof. Dr. Nihan Kesim Çiçekli, for her encouragement, guidance and support all throughout my graduate studies as well as during the preparation of this thesis. I would like to express my gratitude to my co-supervisor, Prof. Dr. Asuman Doğaç, for her guidance and support.

I am deeply grateful to Dr. Gökçe Banu Laleci Ertürkmen, without whose guidance and invaluable contribution, this work could not have been accomplished. I am deeply thankful to Ali Anıl Sınacı for his suggestions and continuous support during the implementation of our approach.

I am deeply grateful to my wife for her love and support, also to my family albeit the very far distance. Without them, this work could not have been completed.

I would like to thank the "Interactive Knowledge Stack for small to medium CMS/KMS providers (IKS)" project for providing the necessary motivation.

I would also thank the Scientific and Technological Research Council of Turkey (TÜBITAK) for providing the financial means throughout this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Majority of today's Content Management Systems (CMSes) lack powerful semantic functionalities on the managed documents [1, 2]. These legacy systems mostly use relational databases to store the data through several relational tables. Some of the CMS providers try to include built-in semantic functionalities, however these cannot go beyond simple and domain specific operations on the stored documents. Considering the diversity and size of available Linked Open Data cloud [3] on the web, more powerful as well as customizable storage and indexing mechanisms are required rather than the standard relational database management systems.

While communicating with other people, content authors put additional, distintictive knowledge into the documents they publish [4]. This annotation process is important considering the activities that could be carried on on top of the additional annotations. Indeed, most of the content management activities such as categorization, retrieval, etc are done according to the semantics of the annotations. Therefore, associating documents with relevant and meaningful annotations and management of these annotations along with actual documents is very critical to provide competent content management activities.

As the number of CMS implementations increase, CMS providers have started to employ some API specifications and standards to increase interoperability between different implementations. JCR [5] and CMIS [6] are two such efforts. In content repositories which are compatible with these standards, documents are represented with nodes. Pre-defined properties are associated to nodes to be populated manually [7]. Although, such properties may give some explicit information about the content, without analyzing the content itself, it is not possible to provide sophisticated semantic features. Also manually annotation of documents is an error prune and time consuming task [8]. Please note that throughout this thesis, we will

use the *content repository* term interchangibly with the *content management system* term. So, they refer to same concept.

In the annotation process of the documents managed in the CMSes the Linked Open Data cloud should also be took into consideration as it grows day by day thanks to its open nature [9], yet it has 52 billion+ triples! Considering the existence of such a huge collection of data, enhancement of documents with semantically related knowledge would be pretty reasonable and such semantically enhanced content would enable building semantic features on top of them, which cannot be obtained via straightforward methods.

In this study, we address the semantic incompetence of existing CMSes. We propose a unified approach which brings several latest advancements in different areas of information extraction (IE), information retrieval (IR) and semantic web technologies all together. In the open source community, there are a remarkable number of mature and stable software tools providing solutions for the advancements in question.

Our study introduces two different software components: the *CMS Adapter* for the semantic lifting of JCR/CMIS compliant content repositories, and the *Contenthub* for powerful semantic indexing and search of the documents considering their metadata and enhancements coming from several different external RDF [10] datasets such as DBpedia [11]. We provide a mechanism for automatic semantic lifting of JCR and CMIS compliant content repositories. We directly communicate with the underlying data model and extract semantic relations, and generate RDF based semantic data to be processed in the semantic search process. The communication is bidirectional, thus conveniently structured RDF data can be pushed back to the content repositories.

Apart from the semantic lifting, one of the salient benefits offered by the features developed in our approach is the opportunity for creating custom, semantically meaningful indexes tuned with specific use cases, needs. By exploiting such a semantically enhanced indexing mechanism, we build a search machinery providing various ways to search for documents by keyword or structured queries or navigate on them using faceted search or related query terms.

Moreover, the aforementioned semantic indexing machinery enables users to use of the linked data according to their specific needs. Linked Open Data cloud already includes lots of

datasets for many different domains. Using such datasets which have reached a substantial maturity provide high-quality annotations, storage and search on documents.

This work has been conducted in the scope of "Interactive Knowledge Stack for small to medium CMS/KMS providers (IKS) project funded by the EC (FP7-ICT-2007-3). Apache Stanbol project [12] is one of the software products developed in the scope of IKS project. The CMS Adapter and Contenthub components introduced earlier are two of the components of the Stanbol, which will be explained in detail in the next sections of this study.

The rest of this thesis is organized as follows: We will give the overall definition of our approach in Section 2, then in Section 3, we elaborate the Contenthub component by explaining the semantic indexing and search functionalities in detail. Then in Section 4, functionalities provided by the CMS Adapter component will be presented. A case study in healthcare domain of our study will be given in Section 5. Before concluding, we will mention about the related works regarding our study Section 6 and finally the thesis will be concluded in Section 7.

CHAPTER 2

APPROACH

The ultimate aim of this study is to provide reusable services and components to enhance CMSes with semantic capabilities. As stated earlier most of the CMSes do not have the capability of mananagement of semantic information related with the documents contained in the system, therefore they cannot provide semantic content management activities such as retrival, navigation, etc. In our approach, we explicate two main components, which tries to overcome the semantic incompetence of existing CMSes. The proposed components are Contenthub [13] and CMS Adapter [14], respectively. Contenthub and CMS Adapter are two of the main components of Apache Stanbol. Although the focus is on those two, it is important to say that they make use of other components of Stanbol. Therefore, the other components that take place in this approach will also be mentioned in a high level manner. By utilising several components of Stanbol, we propose an approach through the integration of several techniques and tools from information extraction, semantic web mining, information storage and information retrieval areas.

In this chapter, the overall approach of our study is given by going through the several parts of the Stanbol framework. Some of the technologies and techniques related with the general approach are given in a moderate detail so that the study would be understood clearly. Also, not to leave the approach in a very abstract state, we mention about some other technologies in a very superficial way. Detailed background for these technologies and techniques are given in Chapter 3 and Chapter 4.

As the base framework for the proposed approach, we start with the Stanbol:

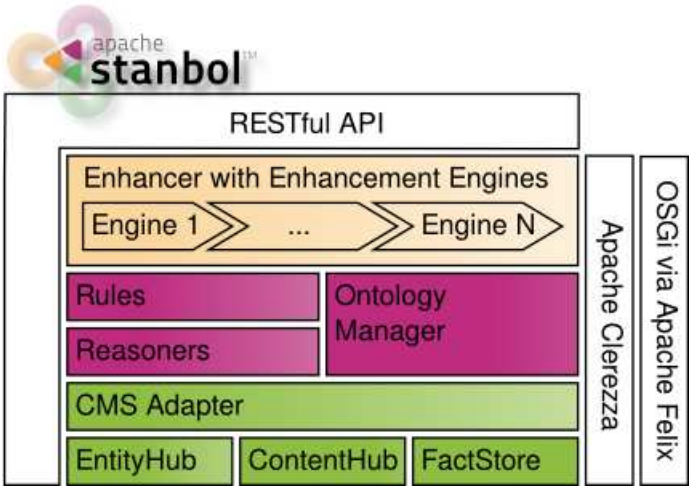

Figure 2.1: Stanbol components [15]

2.1 Apache Stanbol

Apache Stanbol is an open source modular software stack and reusable set of components for semantic content management as presented in Figure 2.1. As its name indicates, for the time being, it is being developed in the scope Apache Software Foundtion (ASF)[1]. In addition to the fact that each component of Stanbol provides both independent functionalities to be used by the CMS providers/developers, integrated services can also be procured by composing of several components of Stanbol.

2.1.1 Design Decisions in Stanbol

Stanbol has been being developed according to a number of design decisions. First of all, it is mainly implemented with Java [16] programming language, although some parts of it is implemented with other programming languages. For example, HTML interfaces of Stanbol are drew using the Java Script and Freemarker [17] templating language.

The second design decision is that Stanbol components should be implemented compliant with Open Services Gateway initiative (OSGi) [19] component framework. Units of resources which can be installed to an OSGi environment are entitled as *bundles*. Once an OSGi environment e.g Equinox runs, any code piece wrapped as an OSGi bundle can be installed to

[1] http://www.apache.org/

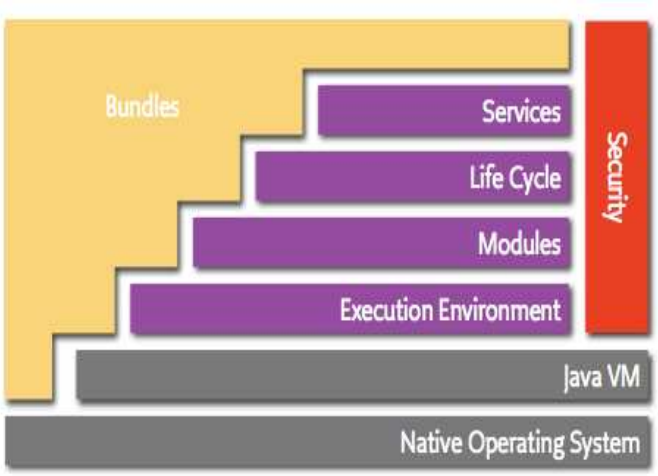

Figure 2.2: Layered OSGi architecture [18]

that environment as well as any bundle already installed to the environment can be updated, stopped or removed from the environment at the application runtime without rebooting the whole system. Figure 2.2 depicts the layered architecture of the OSGi and how bundles operate in several layers. One of the most important features of the OSGi is the service registry concept. In reference to service registry mechanism, each bundle can register one or more services to the OSGi environment and each bundle can detect addition, removal of services and continue to execution according the new state of services. Following this design decision, every component of Stanbol is being developed in a modular manner and they register their own services so that the other components and external clients can make use of them. Also, most of the components require other components to be able to execute. Thanks to the Apache Felix [20] which is the OSGi container implementation used in Stanbol, components are able to listen other components that they need to execute in a proper way. Once all of the listened components are registered to the environment, listener component starts to run and its services become available to be exploited.

The third design decision is the Representational state transfer (REST) [21] architecture. According to this, functionalities provided by components of Stanbol would be accessible as RESTful services. Stanbol uses the JAX-RS [22] which is a Java programming language API that provides support in creating web services in line with the REST architecture. More specifically, Jersey [23] implementation of JAX-RS API is used. In this context, Stanbol becomes a server offering various services and clients can make use of the provided services

through Hyper-text Transfer Protocol (HTTP) [24] methods such as *GET*, *POST*, *DELETE*, etc.

For the compilation step of the Stanbol, the Maven framework [25] is being used. It makes possible to compile several Java projects with a command. Maven als offers purpose specific plugins and allows compilation specific adjustments. For instance, thanks to the Sling Launchpad Plugin [26], the executable programs of all compiled Java projects are aggregated within a single executable Jar file according to their OSGi specific start levels. Afterwards, running the Jar file with the single "java -jar" command is sufficient to launch all OSGi environment.

The Stanbol project is built by several committers. So, it is managed with an version control framework. The Subversion (SVN) framework is being used to keep track of the source code of Stanbol. The online repository is accessible at [2].

2.1.2 Stanbol functionalities

Functionalities provided by Stanbol can be categorized under four main groups:

- **Content Enhancement:**
  Enhancer component [27] of Stanbol enhances a given content by gathering additional semantic metadata. It takes the content and relays it to a set of enhancement engines. Each enhancement engine has a specific purpose. For instance, one of the engines converts the initial content into a processable format, another one extracts the language of the given text. Other engines are able to extract named entities such as organizations and places from the given textual content. Further semantic information about the recognized entities is retrieved from Linked Open Data cloud by some other engines.

- **Reasoning:**
  Reasoning services offered by Reasoners component [28] of Stanbol provides extracting implicit knowledge from the semantic information obtained for the documents via the content enhancement process. Reasoning functionalities are composed of a set of services which make use of automatic inference engines such as Hermit [29], Jena RDFS reasoner [30], etc.

[2] `http://svn.apache.org/repos/asf/incubator/stanbol/`

- **Ontology Management:**

  Ontology Manager component [31] of Stanbol offers a controlled environment where users can access, modify the ontologies managed in the scope of Stanbol. It provides ontology networks and sessions to work on certain parts of complex models which are comprised of several ontologies.

- **Persistence:**

  The Contenthub component of Stanbol is a document repository which provides semantic storage for the content items and semantic search services on top of them. Text-based documents can be submitted, semantically indexed and searched through the services of Contenthub. CMS Adapter is another component which aims to ease integration of semantic functionalities provided by Contenthub with legacy content repositories compliant with JCR/CMIS specifications.

The work carried out in this thesis corresponds with the functionalities grouped under the *Persistence* layer of Stanbol. However, without using the enhancement capabilities of Stanbol on the documents, Persistence layer does not make sense per se. In the following section, how different techniques are combined to provide semantic indexing and search will be given in general terms.

2.2 Semantic Content Management with Stanbol

Initially there are the CMS and Stanbol on the scene. To benefit from the semantic functionalities provided by Stanbol, the CMS should submit its content to the Stanbol. It is possible to get only enhancements as the result of content enhancement process, however through this interaction pattern all semantic data flows in the runtime and the CMS would have nothing stored persistently at the end. However, in this study we focus on the interaction pattern in which the submitted content is store persistently so that sophisticated search functionalities would be provided on the persisted content.

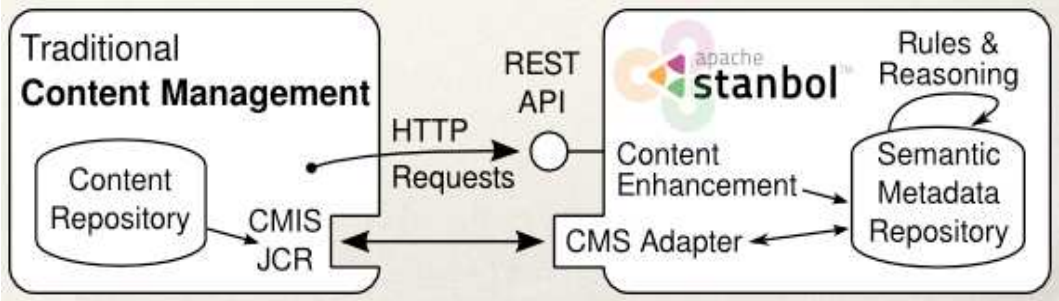

Figure 2.3: CMS - Stanbol interaction

2.2.1 Accessing to Stanbol

To persist submitted content within the Stanbol framework, any CMS can use the RESTful services of Contenthub as the easiest way. Thanks to the REST approach, a CMS can make use of Contenthub services with minimum change to CMS code. Serving the internal capabilities in a RESTful manner provides covering a broad target audience to Stanbol, because REST is a technology independent approach which requires nothing more than accessing to the services through HTTP methods.

The second way to use Contenthub services is the Java interfaces. However, this requires deployment of Stanbol instance into the same OSGi environment in which the CMS itself resides. That is to say, the CMS should have already been running in an OSGi environment and Stanbol modules should be located within the same environment. Once these conditions are satisfied, the CMS can retrieve services using the OSGi platform features e.g it can register for specific services or traverse the existing ones, even it can adapt its own modules such that those modules would be active if and only if the referenced Stanbol modules are already active and ready to use. Sling-stanbol [32] is an integrated framework which can be given as an example to CMSes which benefit from Stanbol services through Java API. Sling-stanbol is composed of union of Stanbol and Apache Sling [33] project which is a JCR based CMS. In the scope of Sling-stanbol the original Sling framework is enhanced with the semantic functionalities provided by Stanbol. For instance, Sling-stanbol provides faceted search over the documents which cannot provided by the Sling itself. Figure 2.3 shows the interaction between a traditional CMS and the Stanbol through RESTful services and CMS Adapter.

[2] http://incubator.apache.org/stanbol/images/stanbol-cms-scenario.png

An alternative way to submit documents from the CMS into the Contenthub could be the usage of CMS Adapter component. However, only JCR or CMIS compliant repositories can make use of CMS Adapter. In this interaction pattern, the CMSes manage their documents within the Contenthub by allowing CMS Adapter to pull the documents from the CMSes themselves and submit to the Contenthub using the standard accessing ways defined in JCR and CMIS specifications.

2.2.2 Content Enhancement

Once the content arrives in the Contenthub, it is directly delegated to the Enhancer component of Stanbol to obtain semantic enhancements belonging to the content itself before any storage and indexing operation performed. Depending on the configuration of Stanbol components i.e the Entityhub and Enhancer, enhancements might include various types of additional knowledge relevant with the content. As introduced earlier, different kinds of enhancement engines contribute different kinds of enhancements. The content enhancement step is the first step where sophisticated techniques are used and in the next paragraphs, the enhancement engines that take part in our approach will be mentioned.

Two of the enhancement engines that are used in our approach are related with Natural Language Processing (NLP). One of them, namely the Named Entity Recognition Engine [34], uses the Apache OpenNLP [35] to extract named entities from the the textual content which is passed as the input. This engine is able to extract *Person*, *Place* and *Organization* typed entities. Another NLP-related enhancement engine is the Keyword Linking Engine [36]. This engines aims to extract occurences of entities which are already a part of a controlled vocabulary. By controlled vocabulary we mean a dataset which is a collection of terms or phrases that are related with a specific domain or subject [37]. An example can be the terms regarding the animal kingdom e.g kinds or other distinctive characteristics of animals, etc. The terms or phrases defined in the controlled vocabularies are used to tag the documents and then the tags are used during the retrieval process of documents. Considering the definition of controlled vocabularies, it can be said that Keyword Linking Engine provides recognition of named entities related with a specific concept or domain, which is represented with a controlled vocabulary.

The last enhancement engine which we are interested in is the Named Entity Tagging Engine

[38]. This engine makes use of the Entityhub component to get detailed semantic information for the named entities recognized within the given text content. Any data source can be bundled in the scope of Entityhub component to retrieve details regarding an entity.

2.2.3 Content Storage

Contenthub is the responsible component from content storage, indexing and retrieval. It manages several semantic indexes which are configured according to specific use cases. Managed indexes differs from each other in terms of the index fields and index field configurations. Index fields and their configurations are specified by means of a simple RDF path language, LDPath, which is described in detail in Section 3.

Once the content enhancement process is completed, Contenthub realizes one last additional semantic knowledge gathering. In this activity, Contenthub uses the named entities recognized during the content enhancement process. It requests additional knowledge for each named entity by querying the Entityhub. However, the most important point of this query operation is that Contenthub queries the Entityhub using the same LDPath instance which was used to create the target index into which the documents are submitted. As a result, only relevant information for specific use cases is obtained. As a result, the acquired information is fully compatible with the target index which is dedicated to the use case.

In addition to the storage of actual content and additional knowledge, enhancements of documents are also stored as a whole. Since the enhancements are in RDF format, triple stores, which are specialized storage frameworks, are used to store the RDF data.

2.2.4 Content Retrieval

On top of the stored content and knowledge i.e the additional, semantic information about the content itself, various types of search operations are provided in the approach carried out in the scope of this thesis. Recent, advanced search techniques such as semantic search and faceted search are combined and a sophisticated content retrieval feature is offered.

The basic search functionility provided is the keyword/full-text search. The full-text search is not only done on the actual content but also considering the additional knowledge about

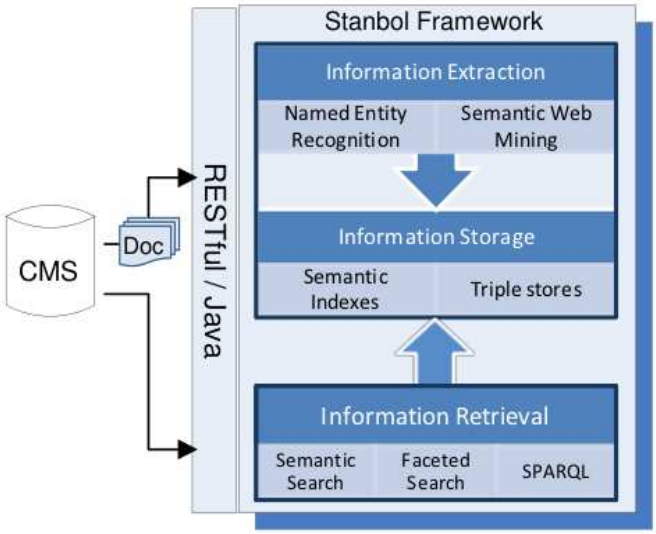

Figure 2.4: Semantic content management overview

the content itself. So this makes possible to get documents in the search results even if the original query term would not be contained in the actual content of the documents.

Another option to retrieve documents is via structured queries. Both the indexes keeping the documents and their purpose specific properties can be queried with structured queries and also the triple stores can be queried with the SPARQL [39] language. Ability to query the stored documents with already available query languages provide users with the opportunity of benefiting from the advanced features of those languages, which have been invented by the immense know-how.

Figure 2.4 shows the workflow of semantic content management in our approach. The flow starts with the submission of documents to the Contenthub through the RESTful API or Java API provided by the Stanbol framework. Using several information retrieval techniques, extra information about the content (e.g language, contained named entities etc...) are extracted. Later on, the extracted information is expanded with the related semantic knowledge retrieved from Linked Open Data cloud. All additional semantic knowledge related with the initial content (retrieved as enhancements) is stored along with the initial content in custom, specialized indexes and it is used during the search process afterwards.

2.3 Integration with Content Management Systems

Apart from the RESTful and Java access to the Stanbol, CMS Adapter is the bridge between the CMSes and Contenthub as stated in Section 2.2.1. While interacting with the CMS Adapter, the CMSes provide the CMS Adapter with some of the distinctive properties of the documents to be submitted to or deleted from the Contenthub e.g paths or identifiers. Thanks to the provided distinctive properties of documents, CMS Adapter would be able to access the CMS and retrieves the specified documents. By doing so, the CMS Adapter component aims to ease integration of semantic storage functionalities provided by Contenthub with CMSes considering the structure of CMS itself. During the submission process, together with the actual content of the documents, their additional properties are also fetched from the CMS submitted to the Contenthub along with the actual content to be used as additional knowledge in the subsequent operations such as search.

In addition to the Contenthub feed facility, CMS Adapter provides transforming the structure of the CMS into RDF format, which can be processed by the Contenthub component during the storage and retrieval activities. Therefore, not only the additional knowledge through the content enhancement process but also the semantics concealed within the structure of the CMS itself. With the same mapping feature, CMS Adapter is also able to update the CMS itself with a given external RDF data. This makes it possible to exploit Linked Open Data which is already available on the web, within the CMSes. By mapping external RDF data, existing documents in the CMS can be updated or new ones can be created. Thanks to this feature, already existing RDF datasets from various domains in the Linked Open Data cloud can be exploited to provide qualified classification/categorization for documents.

CHAPTER 3

SEMANTIC INDEXING AND SEARCH

In this chapter, the Contenthub, which is the core component providing the semantic, customizable storage and search functionality, is being described. To provide better understanding of the study, we first provide a background on the enabling tools, technologies and standards in Section 3.1. After giving the background information, semantix indexing and search approach will be explicated.

3.1 Background

3.1.1 Semantic Web

According to its creator Tim Berners Lee, semantic web is an extension of the original web with better methodologies to express the meaning of things, representing knowledge in standardized ways i.e by defining ontologies [40]. The entities represented as a knowledge piece within the semantic web and relationships between the entities are represented with URIs [41]. Since, the properties of entities and relationship are designed in a way that they would be recognized by machines instead of human beings, semantic web focuses on single entities rather the web pages. The common practice for representing the knowledge in the semantic web is to use well-known vocabularies as much as possible. Vocabularies are dedicated collections including the concept and possible relationship definitions between the concepts related with a specific domain. For instance FOAF [42] is a popular vocabulary which mostly contains concepts, terms related with people. Terms from various vocabularies can also be mixed to make the knowledge representation more understandable by machines [43].

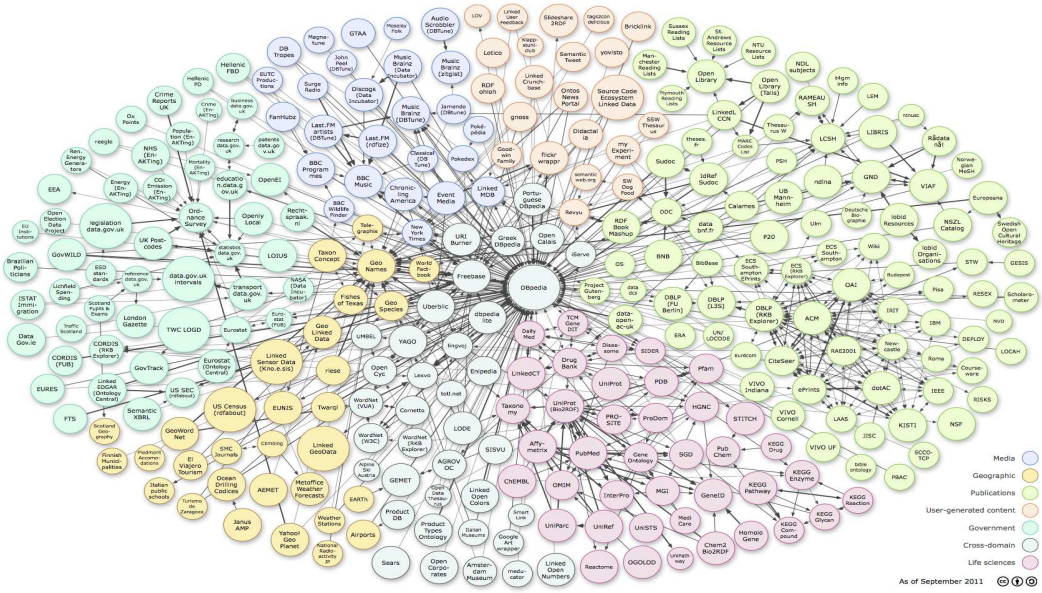

Figure 3.1: Interlinked LOD Datasets[1]

3.1.1.1 Linked Data

From the concept definition perpective, the standardization effort while defining the data within the semantic web necessitates vocabularies to be linked with other vocabularies to able to represent the terms corresponding to the same concept with common, well-known terms. Apart from the concept definitions, to realize the web of data approach, data representing actual entities are defined in an interlinked manner by connecting different pieces of entities with each other. In this way, explicit links between entities are formed and semantic web applications are able to traverse among the entities linked each other [44].

Linked Open Data [3] project aims to extend the original web with data sets which are connected with each other in line with the linked data approach. For the time being, a lot of organizations have already been published their data by associating with other data sets. Figure 3.1 shows the state of links between different datasets from various domains by September 2011.

[1] http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.html

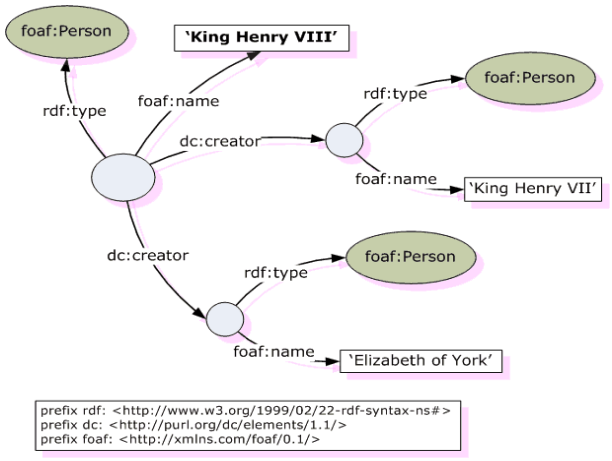

Figure 3.2: Representing persons with RDF [47]

3.1.1.2 Resource Description Framework(RDF)

RDF [10] is the base building blocks of data forming the semantic web. It is the defacto web resource representation language [45]. Although the initial intention to represent metadata about web resources, such as author or modification data of page, RDF is used to represent anything which is referenced through a URI [46]. The actual data regarding the resources in the semantic web can also be dereferenced i.e retrieved through the URIs of resources. The reason to choose to represent the web resources with RDF is not to display data the users, on the contrary to make data understanble by computers.

Representing web resources with RDF is convenient in terms of linked data approach, because RDF provides knowledge representation through RDF statements called as *triples* and each triple represents a property and value of the property for a resource. Mostly a triple is represented as (subject (s), predicate (p), object (o)) where *s* has a property with value *o*. In this structure *s* and *p* must be represented with URIs. However, *o* can be either a URI or a literal value. Figure 3.2 represents three people where the person named as *Elizabeth of York* and *King Henry VII* are the creators of the person named *King Henry VIII*. The same figure also shows the linked data approach where several vocabularies have been used to represent a single web resource e.g *King Henry VIII*.

3.1.1.3 Web Ontology Language (OWL)

With its most generic definition, OWL is a generic knowledge represantation language having different more specific implementations such as OWL Lite, OWL DL and OWL Full. This language is designed to machine readability of the information published rather than just showing to the users. OWL language is used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and the relationships between them is called an ontology [48]. OWL is an extension over RDF, XML and their schema definitions such that it offers an improved vocabulary for describing classes, individual, properties, relations between classes, relations between classes and individuals, typing of properties, etc. The sophisticated structure of this language also allows reasoning on the knowledge annotated with this language. Through a reasoning process, the implicit information hidden in the actual knowledge is extracted by making use of the relations between information items. Since design of the OWL language is notably suitable for the reasoning process, the prevalent reasoners are ones those work on OWL annotated data such as HermiT [29], Fact++ [49].

3.1.2 Data Persistence and Access

When it is considered from the perspective of document level persistence and access in the CMSes, the most common feature is the keyword search over the actual content of documents and their metadata [50]. There are also attitudes towards to enriching the full-text search with structured search methodologies [51, 52, 53]. In this section, we will give background information about different storage and search methodologies, which takes place in the approach proposed in this study, in different granularities.

3.1.2.1 Apache Lucene & Apache Solr

Apache Lucene [54] is an open source full-text search engine library written in Java programming language. However, it is being ported several other languages such as Delphi, Perl, C#, C++, Phyton, Ruby, PHP, etc. The indexing unit of the Lucene is textual documents with additional metadata fields to be indexed alongside the actual documents. Lucene has the flexibility to extract textual content from different formats of files such as PDF, HTML, Microsoft

Word, Open Document documents, etc.

Apache Solr [55] is an open source enterprise search platform which is set up on the full-text indexing and search capabilities of Lucene. Solr offers a high scalability thanks to its distributed search and index replication features. It can also be used as a standalone full-text search server within a servlet container such as Apache Tomcat [56], Glassfish [57], etc. Clients can access to Solr server through HTTP protocol [24] sending queries in XML [58] or JSON [59] formats. This removes any dependency to a specific programming langugage. Easy index configuration options and programming language independent features allows applications to use the Solr as an underlying indexing and search framework and offer faceted search features over the indexed content.

Faceted search is a search paradigm which guides the users for navigating on the multidimentional data [60]. It allows gradually constraining or expanding the documents or search results in an intuitive way. Constriction and expansion operations are realized by the facets and their associated values which match with the presented results. As long as users choose more facets to constrain documents, available facets and their associated values decrease vice versa as long as already chosen facets are deselected more and more possible facet and facet values appears.

Search Ranking with Apache Lucene

For the documents whose textual content can be extracted, Lucene provides scalable and high performancing indexing features. Lucene provides a ranked search over the indexed content. Ranking is done according to *tf*idf* (term frequency-inverse document frequency) weight. This weight is a numerical statistics indicating the importance of the word within a document contained in a collection or corpus [61]. Lucene also supports powerful query types such as wildcard queries, range queries, etc.

In detail, the following mathematical formula is used by the Lucene to calculate ranks of the resultant documents in a search process.

```
score_d = sum_t (tf_q * idf_t / norm_q * tf_d * idf_t / norm_d_t * boost_t)
* coord_q_d
```

This formula used for ranking also applies for the search mechanism proposed in our study. The meaning of the terms in the formula are as follows:

- **score_d:** Score for document d.
- **sum_t:** Sum for all terms t.
- **tf_q:** The square root of the frequency of t in the query.
- **tf_d:** The square root of the frequency of t in d.
- **idf_t:**

  ```
  log(numDocs / docFreq_t + 1) + 1.0
  ```

- **numDocs:** Number of documents in the index.
- **docFreq_t:** Number of documents containing t.
- **norm_q:**

  ```
  sqrt(sum_t((tf_q * idf_t)^2))
  ```

- **norm_d_t:** Square root of number of tokens in d in the same field as t.
- **boost_t:** The user-specified boost term for term t.
- **coord_q_d:** Number of terms in both query and document / number of terms in query.

This can be summarized as the scoring is based on term frequency and term density in the document.

3.1.2.2 Triple Stores

Considering the massive amount of linked data represented with RDF, there is a need to store and access to RDF data considering the nature of the RDF. Triple stores are dedicated databases to store RD [62] such that basic functionalities expected from a database such export, import, querying, etc are specialized for RDF representation in triple stores. Triple stores also provide inference (reasoning) services on the managed knowledge, resulting in extraction of implicit knowledge which is not originally contained in the initial knowledge.

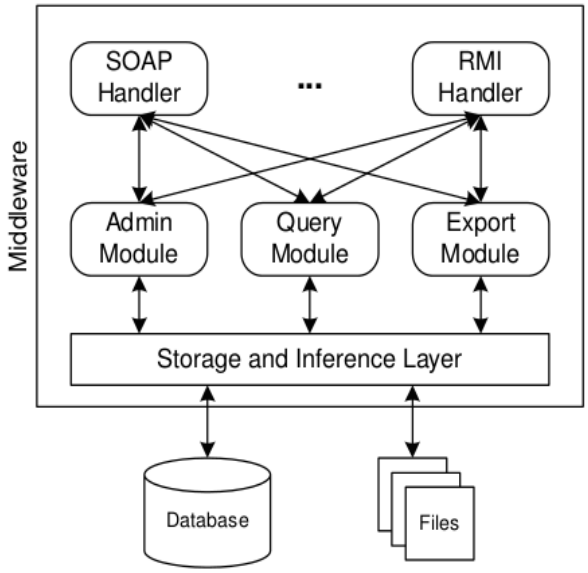

Figure 3.3: Generic Architecture of an RDF Store(Sesame) [64]

There are different approaches while building the triple stores. While some of the are implemented from scratch, some others are built on top of existing relational database engines [63]. Triple stores are mostly composed of two layers which are the repository and middleware layers [64]. As triple stores can use different mediums e.g files, databases, main memory for storage, repository layer capabilities are offered via unified interface which is independent from the underlying storage medium. Apart from the storage API, inference services also resides in the middleware layer. Sesame [65], one of the popular triple stores, applying this approach as seen in the Figure 3.3. The picture depicts also some of the middleware layer modules, which are not focused in this study.

For querying the RDF data, triple stores offer various ways such as implementing a proprietary API such as SeRQL [66], KAON Query [67] or implementing a query language such as SPARQL [39], RQL [68], etc. We focus only on SPARQL since it is the only RDF querying language used in our approach.

3.1.2.3 SPARQL

SPARQL is the abbreviation of the statement *SPARQL Protocol and RDF Query Language*. SPARQL is the W3C recommended query language for the semantic web which is formed by

interlinked RDF data. Since RDF is a directed graph data format, SPARQL would be a graph-matching query language consisting of triple patterns, conjunctions, disjunctions and optional query patterns. According to [69], a SPARQL query is composed of three parts: The first part is the *pattern matching* part where the constraints reside in triple forms. The constraints can be used together with the advanced features of SPARQL such as optional parts, unions, etc. The second part is *solution modifiers* part which is used to postprocess the raw output of the query with operators like distinct, order, limit, offset, etc. The last part of a SPARQL query is the actual *output* part which can be of different types such as yes/no queries, selections of values of the variables which match the patterns, construction of new triples from these values, and descriptions of resources.

3.1.2.4 Apache Clerezza

Apache Clerezza is an OSGi based framework aiming to provide a set of functionalities as RESTful services to manage the semantically linked data [70]. Clerezza's Smart Content Binding (SCB) layer allows storage of the RDF content in a compatible way with the W3C RDF specification [46]. Clerezza is able to use different triple stores such as Jena [71] or Sesame [65] as the underlying framework storing the content. Underlying triple stores are abstracted with the graph data model, which is the part establishing the compatibility with the W3C RDF specification. SCB layer also provides different facades to directly use Jena or Sesame APIs during the RDF graph processing. The RDF graph manipulating API choice is independent from the underlying storage mechanism. In other words, while using the Jena as underlying triple store, Sesame API can be used to manipulate RDF graphs.

3.1.2.5 LDPath

LDPath is an RDF query language which is a valuable side-product of Linked Media Framework (LMF) [72] project. It allows querying RDF as querying XML with XPath [73] and it provides advanced features well-suited for querying and retrieving resources from Linked Open Data cloud [74]. LDPath offers a high level querying layer over the RDF data abstracted as RDF backends. By default, it provides three different RDF backend implementations which are Linked Open Data cloud, Sesame triple store and generic RDF file backends. Besides, LDPath can be used as a query language to work on any triple store or RDF data set by im-

plementing the RDF backend interface and it is also being used as a configuration template to create semantic Solr indexes as in LMF or Stanbol, which will be explained in detail in the following sections of this chapter.

3.2 Semantic Indexing

Before going into the details of what kind of indexing mechanism is proposed in this study, we first explain what we mean by *semantic indexing*. In the context of full-text database systems, indexes per se are units that provides efficient retrieval of documents satisfying the given query, efficiently addition of new records and rank the results with respect to the given query such that users retrieve the the records of interest first [75]. Semantic indexes introduced in this study are indexes which are configured according to specific needs so that they are built on certain fields of documents that are inserted to the index.

3.2.1 Semantic Index Management

As stated earlier in this document, Contenthub is the component which is in charge of the semantic index management. It can manage multiple indexes and allows conveying of storage and search requests to a certain index. In the background, it uses the Apache Solr [55] as the underlying framework providing the actual indexes and realizing the indexing and search operations. In one more level deep, Solr uses Lucene framework and Lucene use inverted index file structure where each term extracted from documents points to a list of documents in which that term occurs [76].

Creation of new semantically meaningful indexes for certain use cases is realized thanks to the LDPath [74] language. An LDPath instance is called as an LDPath program, so from now on we will use this denomination. LDPath programs are composed of a **<field_name:rdf_path>** pairs. While the field name only indicates the name of an index field, the RDF path piece may specify properties about the field to be created such as its data type, language, etc. It is even possible to set advanced configurations for the index fields within the RDF path section. Contenthub provides services to create custom indexes based on a given LDPath program. That means once a user submits an LDPath program to Contenthub, a corresponding Solr index is created in the background.

```
title = foaf:name[@en] | fn:concat(foaf:givenname[@en]," ",foaf:surname[@en]) ::
        xsd:string (stored="false",multivalued="false");
```

Figure 3.4: A field name and corresponding RDF Path of an LDPath program

```
<field name="title" type="string" indexed="true" stored="false" multiValued="false" />
```

Figure 3.5: Solr schema configuration of the LDPath instruction in Figure3.4

In the Figure 3.4, a simple <*field_name:rdf_path*> pair is seen. Assume that this definition is one of the pairs defined in a complete LDPath program. This line indicates that *title* will be one of the fields in the index to be created. Its RDF path part *(foaf:name[@en] | fn:concat(foaf:givenname[@en]," ",foaf:surname[@en])* is related with document submission, so it will be explained later on in Section 3.2.2. However, the type assigned to the RDF path determines the field type to be created within the index. Furthermore, *(stored="false", multivalued="false")* is a part specifying Solr specific configurations. According to this instruction, index field will not be stored but only be indexed also it will not be possible to pass multiple values to this field. All Solr specific field options can be found at [77]. Once a Solr index is created based on an LDPath program including the <field_name:rdf_path>definition depicted in the Figure 3.4, the corresponding index configuration, which can be seen in Figure 3.5, is created for the *title* field.

Contenthub provides both RESTful and Java API to manage the underlying solr index. Indeed, the real job is done by the services implemented in different OSGi bundles composing the Contenthub component and the core services are wrapped as RESTful services so that they would be accessed and called in a technology independent way through the HTTP protocol from various environments.

3.2.2 Submitting Documents to Semantic Indexes

After creation of an index with an LDPath program, documents can be submitted into it. However, in addition to indexing the content, additional knowledge related with the content is extracted by Enhancer component of Stanbol. Enhancements are returned in RDF format and they include various information about the content of the document such as language,

named entities, possible references to resources corresponding to detected named entities in the Linked Data cloud and several other information about enhancements themselves. Enhancements of content items are stored in a triple store, abstracted by Apache Clerezza [70]. Afterwards, it would be possible to execute SPARQL queries on the enhancements. The Figure 3.6 shows an example enhancement in RDF/XML format corresponding with an organization typed named entity which was recognized in a textual content. It contains various information about the named entity as described below:

- **extracted-from:** This property is a reference to the owner document which this enhancement belongs to. It links to the ID of the content item as assigned by Stanbol.
- **confidence:** This is a numeric value indicating the relevance degree of external entity referenced in this enhancement with the named entity mentioned in the text where this enhancement is extracted from.
- **entity-type:** Indicates the type of the external entity referenced in this enhancement. In this example, the entity have the types of *Organization* and *Broadcast*. *Thing* is a general base type which can be used for all entities.
- **rdf:type:** This property indicates the type of the enhancement itself such as *Text Annotation*, *Entity Annotation*, etc. *Text Annotation* typed entities are the ones representing the named entities in the given textual content. The example enhancement is an *Entity Annotation* meaning that unlike the *Text Annotations*, this enhancement does not represent the named entity itself recognized in the text. Rather, it is a reference to an external entity representing the corresponding named entity having *Text Annotation* type.
- **entity-label:** Label of the represented external entity.
- **created:** Indicates the time when this enhancement was created.
- **creator:** This is a reference to an *Enhancement Engine* of Stanbol Enhancer.
- **entity-reference:** This property is a reference to external entity itself which is defined in an external data set included in the Linked Open Data cloud.
- **relation:** This property is a reference to the enhancement having *Text Annotation* type and the enhancement having this property is a candidate external entity in Linked Open Data cloud for the named entity recognized from the textual content.

```
<rdf:Description rdf:about="urn:enhancement-730b17cf-2937-4ce4-dd73-1783362d566b">
  <j.7:extracted-from rdf:resource="urn:content-item-sha1-3577ae887882be008f5ea60528f19a4aef5c4192"/>
  <j.7:confidence rdf:datatype="http://www.w3.org/2001/XMLSchema#double">197440.65625</j.7:confidence>
  <j.7:entity-type rdf:resource="http://dbpedia.org/ontology/Organisation"/>
  <j.7:entity-type rdf:resource="http://dbpedia.org/ontology/Broadcast"/>
  <j.7:entity-type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdf:type rdf:resource="http://fise.iks-project.eu/ontology/EntityAnnotation"/>
  <rdf:type rdf:resource="http://fise.iks-project.eu/ontology/Enhancement"/>
  <j.7:entity-label xml:lang="en">CNN</j.7:entity-label>
  <j.1:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2012-05-11T19:01:01.691Z</j.1:created>
  <j.1:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">org.apache.stanbol.enhancer.engines.entitytagging.impl.NamedEntityTaggingEngine</j.1:creator>
  <j.7:entity-reference rdf:resource="http://dbpedia.org/resource/CNN"/>
  <j.1:relation rdf:resource="urn:enhancement-cd601fb5-5a4a-29a4-5355-e3e8ab2a8e08"/>
</rdf:Description>
```

Figure 3.6: Enhancement representing the CNN Organization

```
413             ...
414             Iterator<Triple> it = sci.getMetadata().filter(null, Properties.ENHANCER_ENTITY_REFERENCE, null);
415             Set<String> contexts = new HashSet<String>();
416             while (it.hasNext()) {
417                 Resource r = it.next().getObject();
418                 if (r instanceof UriRef) {
419                     contexts.add(((UriRef) r).getUnicodeString());
420                 }
421             }
422             Map<String,Collection<?>> results = semanticIndexManager.executeProgram(ldProgramName, contexts);
423             for (Entry<String,Collection<?>> entry : results.entrySet()) {
424                 doc.addField(entry.getKey(), entry.getValue());
425             }
426             ...
```

Figure 3.7: Querying Entityhub for the recognized named entities

In the next step of indexing process, further details about the named entities contained in the document are obtained from the Linked Data Cloud through Entityhub component of Stanbol. Entityhub is queried with the LDPath instance, which was previously used to create the target index, in order to get detailed information about the entities. To express this process clearly, we go over the corresponding source code depicted in Figure 3.7.

The code snipped included in the Figure 3.7 is written in Java programming language. In the line 413, the variable named *sci* represent a document submitted to the Contenthub. In the same line, its enhancements are obtained in an RDF graph by *getMetadata()* method. Again in the same line, the triples having *entity-reference* property is requested from the graph. One *entity-reference* property must be included in each of *Entity Annotation* typed enhancements. So, by requesting triples having *entity-reference* property, we automatically obtain references to external entities corresponding with the named entities recognized in textual content. Between the lines 414 and 420, all references to external entities are collected in a *Set* object. After that in the line 421, *Semantic Index Manager* of Contenthub is used to execute the

LDPath program on the Entityhub. Through the *executeProgram* method of *Semantic Index Manager*, the LDPath program identified with the given parameter *ldProgramName* is executed on the entities passed in the *contexts* set. It can be understood from this code that the document submitted to the index which was already created with the same LDPath program identified with the *ldProgramName*. In the same line, results are obtained in a *Map* instance. The keys of the map are formed by the names of the index fields e.g *title* as seen in the figure above. The corresponding value of keys is a *Collection* of values for the field specified in the key. A collection can include any type of values such string, integer, date, etc. In the rest of the code snippet, the values obtained from the Entityhub are passed to the variable *doc* which is the Solr representation of the document to be indexed. As seen in the line 423, keys of the result map are added as fields and collection of corresponding values are added as the values of the fields to Solr representation of document.

While querying the Entityhub with an LDPath program for an external entity, the effective parts of an LDPath program are RDF paths of the <*field_name:rdf_path*> pairs composing the LDPath program itself. Starting from the given context URI [41] which is the identifier of an external entity, all of the RDF paths in an LDPath program are executed over the Linked Open Data cloud. This would make possible to obtain knowledge from various datasets by following the links defined in the RDF path. For the querying process, LDPath provides advanced features as well such as *Wildcard Selections*, *Unions*, *Functions*, etc. The RDF path part of the LDPath program depicted in the Figure 3.4 contains a *Union* function which has two operands. The first operand is *foaf:name[@en]* and the second one is *fn:concat(foaf:givenname[@en], " ", foaf:surname[@en])*. The result of this RDF path is obtained by either the first or the second operand. The first operand states that the result will be obtained by obtaining the *English* value of the *foaf:name* property of the external entity identified by the given context URI. On the other hand, the second operand is composed by a *Concatenation* function which takes three parameters. As a result of this function the English value of the *foaf:givenname* property and the English value of the *foaf:surname* property of the specified entity will be concatenated with a space character in between.

As a result of the additional knowledge gathering process, meaningful details of entities are obtained from the Linked Open Data cloud. This increases the quality of the annotations with respect to the context information given with the LDPath program. Another effort which may increase the annotation quality is the configuration of Entityhub and Enhancer compo-

nents. Entityhub component provides configuration of specific datasets from Linked Open Data cloud. Those datasets can be associated with *Keyword Linking Engines* so that they can be used during the content enhancement process. The configuration step is an optional one, but if it is done once in the beginning, named entities are detected using the domain specific datasets and details of the entities are obtained from them. After the initial configuration, the annotation and indexing process is fully automatic. It is enough to specify the name of the index (Solr core) during document submission. Based on the specified index, documents will be annotated according to the LDPath instance and indexed accordingly.

During the document submission process, it is also possible to provide optional metadata in the form of **<field:value>** pairs. If the passed field value is not defined in the target index, a dynamic field is created and it can still be used in the search process, because it will be automatically indexed. In the default index of Contenthub three such dynamic fields are created by default. These are *place*, *organization* and *person* fields. Since Stanbol Enhancer produces enhancements of person, place and organization types in default configuration, we provide these index fields with the values parsed from the enhancements of documents.

As a conclusion, the crucial point of the proposed semantic indexing mechanism is the materialization of all related knowledge i.e knowledge retrieved from Linked Data cloud or manually provided metadata along with the initial content. Having this valuable knowledge allows us to provide advanced search functionalities on top of them, which are described in Section 3.3.

3.3 Semantic Search

The semantic search machinery of our approach is mainly built on the additional semantic knowledge indexed along with the original content. We also benefit from the Linked Open Data during some of the provided search features. Throughout the search machinery the following search functionalities are available:

- simple keyword search,
- document retrieval via structured queries e.g. SPARQL [39], Solr Query [78],
- document filtering with faceted search,
- document exploration with related keywords suggested for the original query term.

To realize the search methods, the Contenthub provides following services:

- SPARQL Search,
- Solr Search,
- Related Keyword Search,
- Featured Search

Before going to into the details of various search mechanisms, we first explain how an indexed document with a known identifier is retrieved in the scope of Contenthub in the Section 3.3.1.

3.3.1 Single Document Retrieval

Documents are retrieved according to their identifiers which are returned to the users just after the document is stored. As the documents are stored within two main storage modules, which are namely Solr indexes and triple stores, once a document is requested through its identifier the two aforementioned storage modules are queried with the given identifier. The content itself, additional knowledge obtained through the LDPath program execution on the Entityhub and manually provided additional knowledge about the content are obtained from the Solr indexes. On the other hand, enhancement of content items are obtained from the RDF graph containing the enhancements of all documents, by executing a SPARQL query which is shown in the Figure 3.8.

The SPARQL query depicted in the Figure 3.8 is the query used for retrieving the document enhancements. Thanks to this query, union of triples having the identifier of the document, i.e the URI of the document as the object; and triples which are typed as *Entity Annotation* and belonging to the document specified with the given identifier.

3.3.2 SPARQL Search

Stanbol framework is able to serve RDF graphs through a SPARQL endpoint so that they can be accessed through HTTP. The Contenthub uses this facility of Stanbol and serves the enhancement graph which keeps the RDF enhancements of all documents submitted to the

```
"PREFIX fise: <http://fise.iks-project.eu/ontology/> "
+ "SELECT DISTINCT ?enhID WHERE { "
+ "  { ?enhID fise:extracted-from ?contentID . } UNION "
+ "  { ?enhancement fise:extracted-from ?contentID . "
+ "      ?enhancement a fise:EntityAnnotation . "
+ "      ?enhancement fise:entity-reference ?enhID . } "
+ "    FILTER sameTerm(?contentID, <" + contentID + ">) " + "}"
```

Figure 3.8: SPARQL query to retrieve the enhancements of a document

Contenthub. Through this SPARQL endpoint, documents can be queried; navigated by means of their enhancements. The enhancement graph also includes the resources representing the external entities located within external data sources. Therefore, it is possible to execute SPARQL queries considering those external entities. However further entities linked by the first level external entities cannot be included, unless they are detected as external entities from the submitted documents as well. By executing the SPARQL query depicted in the Figure 3.9, the identifiers of content items are requested such that the content items would have one or more enhancements pointing external entity. The referenced external entities are expected to be cities of Japan and have population more than 10000000.

In Section 3.3.1, we have already exemplified a SPARQL query execution in a different way. The difference is that while pulling out a single document, Contenthub uses the Java API of Clerezza to execute the query on the enhancement graph. On the other hand, the example we have given in this section demonstrates the dedicated endpoint of Stanbol to execute SPARQL queries on any registered graph through the RESTful services.

3.3.3 Solr Search

Solr indexes are the fundamental parts of the Contenthub's search infrastructure. This service makes it possible to query underlying Solr indexes directly. It provides keyword search over the indexes and in keyword search documents are retrieved not only considering their actual content, but also the related knowledge indexed along with the content. Thus, it would be possible to retrieve documents with a query term which is not contained in the actual document content but in additional knowledge regarding the original content. For instance, assume that a document contains the *Dennis Ritchie* entity. At the end of the indexing process, *C Pro-*

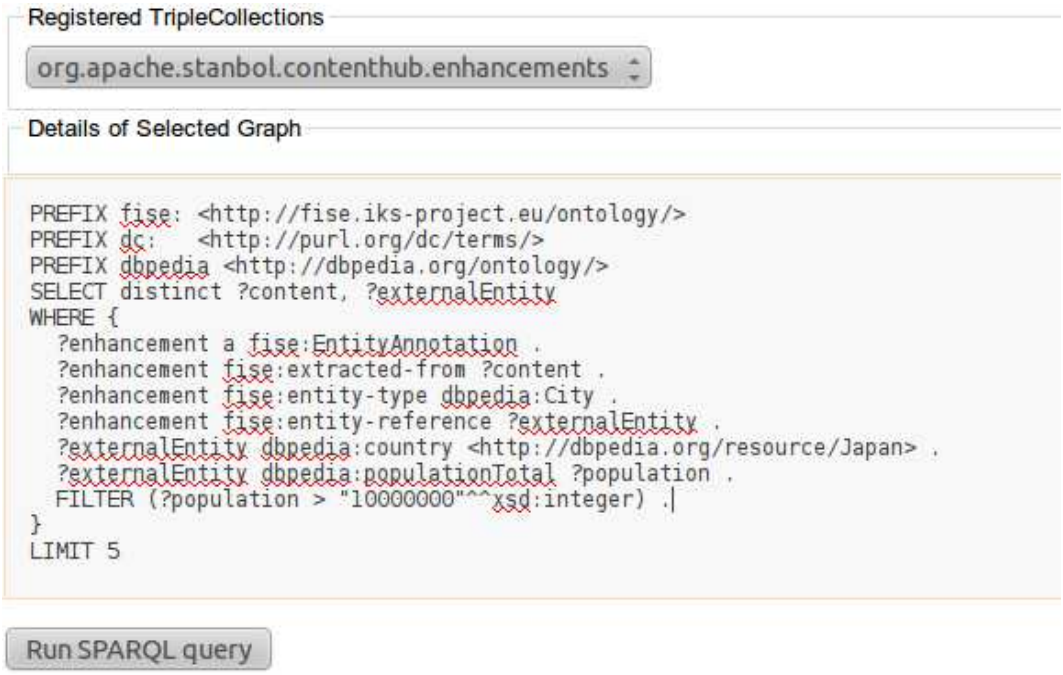

Figure 3.9: SPARQL query execution on enhancement graph

```
QueryResponse search(String queryTerm) throws SearchException;
QueryResponse search(String queryTerm, String indexName) throws SearchException;
QueryResponse search(SolrParams solrQuery) throws SearchException;
QueryResponse search(SolrParams solrQuery, String indexName) throws SearchException;
```

Figure 3.10: Methods provided by the Solr Search service

gramming concept is indexed within the *dbpedia:knownFor* property of the document along with the document itself. Afterwards, when a keyword search is done with the *C Programming* term, the document containing the *Dennis Ritchie* entity would be in results even if it does not contain the query term itself.

Thanks to the Solr Search service, executing Solr specific queries is also an option. Solr query syntax provides very advanced and flexible document retrieval facilities. SolrJ [79] API is used as the client to access underlying Solr indexes. The Figure 3.10 represents the methods that can be used by external clients. As can be seen in the figure, it is possible to send the query as a *String* object or as a Solr query within a *SolrParams* instance. Also, for both cases it is possible to specify a different Solr index. In all cases results are returned in

a *QueryResponse* instance. Both *SolrParams* and *QueryResponse* are classes defined in the SolrJ API.

Considering that most of the CMS providers are familiar with Solr framework, opportunity to direct usage of Solr indexes eases the integration of services. All Solr indexes can also be used through HTTP protocol.

3.3.4 Faceted Search

Solr framework offers the opportunity for faceted search over the managed indexes as a built-in capability. This search paradigm allows users to navigate over the search space in multiple dimensions. For such a navigation, documents contained in the search space should be classified along multiple categories. In our approach, the search space corresponds to the Solr indexes created for different use cases, different domains, different needs, etc. The documents lie within those Solr indexes and the document classification is done according to the fields defined in certain Solr indexes. In the course of faceted search in our approach, facets are constructed from the fields defined in Solr indexes.

Providing a set of intuitive facet regarding the stored documents is usually a challenging task [80]. While in some approaches, like ours, facets are constructed using the metadata fields or terms in the vocabularies [81, 82, 83], other uses logical rules are defined so that the more meaningful and easily comprehensible facets are presented to the users [80, 84]. Even though the facets are directly constructed based on the index fields in our approach, since users are able to define the index fields according to their needs and they are even able to associate custom RDF paths to populate the fields, throughout the system an adjustable facet construction mechanism is offered. In terms of the comparison with the classical best-first search approach, faceted search paradigm seems to provide more effective information-seeking support [85]. It is also popular among many popular online information access systems such e-commerce sites.

Default index of Contenthub does not include any index field originating from an LDPath program. Nevertheless, we create three dynamic index fields for the *place*, *organization* and *person* typed enhancements of documents and they would result in three corresponding facets. Values of the facets include the names of the external entities obtained from the DBPedia

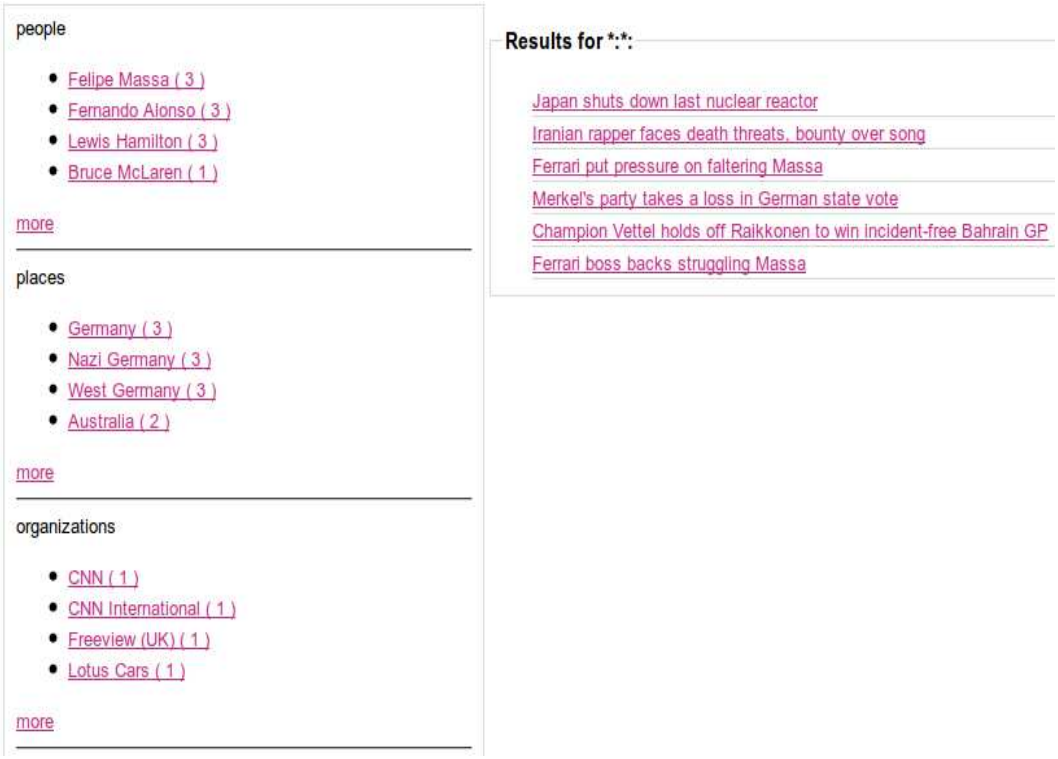

Figure 3.11: Facets included in the default index of Contenthub

[11]. In the Figure 3.11, aforementioned three facets can be seen. Possible facet values are obtained from all documents and the numbers on the right hand side of a facet value indicates the number of documents matching with that value of the facet.

3.3.5 Related Keyword Search

In addition to the document search over semantic indexes, the Contenthub provides a related keyword suggestion mechanism for the given initial query term. These services accept single keywords or tokens and return related keywords/tokens. This can be considered as a search functionality which returns new keywords instead of documents. The aim of this approach is to catch the intention of the users and offer them document navigation options with new keywords of interest in terms of specific domains or related keywords defined in standard vocabularies, datasets, etc.

Currently, three different related keyword suggestion services exist:

- **Ontology Resource Search:** This service looks for the specified keyword in the resources of the ontology which is given as an external ontology to the search operation. Search API of the Contenthub allows passing an ontology to be included in the related keyword suggestion phase of the search process. As ontology resources whose names to be compared with the given query term, first of all this engine considers the ontology individuals and classes defined by using the OWL language [48] annotations. Apart from the OWL classes and individuals, it also considers the resources having *http://www.apache.org/stanbol/cms#CMSObject* as value of their *rdf:type* properties. The *http://www.apache.org/stanbol/cms#CMSObject* type is the type of resources of the ontology produced by the CMS Adapter. Such ontologies represent the structure of the CMSes and by processing these ontologies actual structure of the CMSes are included in the search process. This topic will be covered in more detail in Chapter 4.

  To provide an efficient search over the ontology, we index the local names of the target resources using the LARQ [86] framework. This framework exploits the functionalities of Lucene to provide efficient indexing and search over the given ontologies. By exploiting the LARQ framework, we execute SPARQL queries directly over the indexed ontology to retrieve the similar terms considering the initial query term. In Figure 3.12, three SPARQL queries for the three types of target objects are seen.

  After an ontological resource is determined as relevant term with the search query, this engine considers the other ontology resources residing in the vicinity of the initial term. The additional ontology terms in the neighborhood of the initial ontology term are detected through a number of closure properties among the ontological concepts such as subclass and superclass. Furthermore, the engine allows specifying a subsumption property so that any custom hierarchical structure of the ontology can be exploited.

- **Wordnet Search:** Looks for related words (such as synonyms, hyponyms, hypernyms, etc...) from Wordnet dictionary [87] up to a configurable level and returns them as related keywords. As its name indicates Wordnet is a collection of interlinked meaningfully related words such as noun, verbs, adjectives and adverbs. By using Wordnet, we aim to provide users with semantically related keyword to be used to direct the search process.

- **Referenced Site Search:** RDF datasets obtained from the Linked Open Data cloud and managed in the scope of the Stanbol are named as *Referenced Sites*. As you remember,

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX pf: <http://jena.hpl.hp.com/ARQ/property#>
PREFIX ss: <http://stanbol.apache.org/contenthub/search/>
SELECT ?individual ?score WHERE {
    ?individual rdf:type ?type.
    ?type  rdf:type owl:Class.
    ?individual ss:hasLocalName ?name.
    (?name ?score) pf:textMatch '+query term*'.
}
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX pf: <http://jena.hpl.hp.com/ARQ/property#>
PREFIX ss: <http://stanbol.apache.org/contenthub/search/>
SELECT ?class ?score WHERE {
    ?class rdf:type owl:Class.
    ?class ss:hasLocalName ?name.
    (?name ?score) pf:textMatch '+query term*'.
}
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX pf: <http://jena.hpl.hp.com/ARQ/property#>
PREFIX ss: <http://stanbol.apache.org/contenthub/search/>
PREFIX cms: <http://www.apache.org/stanbol/cms#>
SELECT ?cmsobject ?score WHERE {
    ?cmsobject rdf:type cms:CMSObject.
    ?cmsobject ss:hasLocalName ?name.
    (?name ?score) pf:textMatch '+query term*'.
}
```

Figure 3.12: SPARQL queries to fetch similar terms for the original search query

Entityhub is responsible from the management of *Referenced Sites*. Given a keyword as input, this search looks into the RDF datasets through the Entityhub component and returns related entities accessed through specific properties. For example, for a specific person this service returns related entities which are obtained from DBPedia via *dbpedia:birthPlace*, *dbpedia:country*, etc... properties of the entity associated with that specific person. So, based on the specific properties, when the query is term is *Ankara*, this service may suggest the *Turkey* as a related keyword.

Related keywords are important in terms of the *exploration* aspect within the overall search process. A user may start from a point and instead of restricting the search results, might want to change direction based on the initial keyword. This provide an intutive navigation facility over the best-first search approach.

3.3.6 Featured Search

Featured search basically combines the capabilities of *Solr Search* and *Related Keyword Search* so that resultant documents and related keywords can be returned as a result of single service call. Thus, the target users i.e the CMS developers, CMS vendors could offer a complete search framework to their end users. In featured search component, the Contenthub provides a *tokenize* service. It performs a preprocessing on the search query and tries to extract entities. Entities can be formed of a single word or several words. If an entity has more than one word, it is searched as a whole. This means that we construct the underlying Solr query considering the whole name of the entity. For instance assume that the initial query term is *Michael Jackson New York concert*. Before any search operation, this query term is sent to Enhancer to be analyzed. After that the *Michael Jackson* and *New York* entities are detected and they are searched as a whole i.e as if the query term was like *"Michael Jackson" "New York" concert*.

In the scope of Featured Search service, we also provide an abstraction layer over the faceted search facilities so that the users who are not familiar with the Solr framework can benefit from the faceted search capabilities provided by Contenthub and integrate the services into their existing system in an smooth, straightforward way.

CHAPTER 4

INTEGRATION WITH CONTENT MANAGEMENT SYSTEMS

Semantic services offered by the Contenthub can be accessed through its RESTful or Java APIs. Besides these APIs, the CMS Adapter acts as a bridge between the content management systems and the Contenthub in terms of storage functionalities. The CMS Adapter interacts with the CMSes through the JCR and CMIS specifications. In other words, any content repository, compliant with JCR or CMIS specifications, can make use of CMS Adapter functionalities. Before diving into the details of CMS Adapter, a background about the CMSes and standards specifying the model of the data to be stored in the CMSes and accessing ways to it will be given. After giving the background information, we will explain how CMSes can be integrated in the course of indexing and search operations.

4.1 Background

4.1.1 Content Management Systems

A content repository or in other words a content management system is a hierarchical content store supporting structured and unstructured content, full text search, versioning, transactions, observation and more[88]. CMSes can be of different types such as Web Content Management Systems, Digital Asset Management Systems, Enterprise Content Management Systems, etc. Considering these different types, CMSes are environments used by content managers to manage the content items called as digital assets in mostly in a central repository. Apart from the actual content of digital assets which can be in different types such as text, video, audio, image, etc, CMSes also keep metadata about the content items [89]. For instance *author*, *last modification date*, etc. of a web page. Aside from the storage functionalities, CMSes offer

provide the content managers with authoring environment for the assets in which multiple authors are able to work on the same resource. Furthermore, CMSes serves traits for metadata creation and archiving for the content items together with versioning opportunities.

4.1.2 Content Repository Standards

Ad-hoc implementations of content repository models preclude the code portability, and hence, the application developers are obliged to work with multiple APIs. Therefore, the content becomes isolated in repositories which are available for specific applications, designed to access those specific content repositories [50]. Upon increasing number of implementations of different content repository models standards have been emerged. There are two aspects on which these standards propose common ways of usage:

- The structure of the data to be stored in content repositories
- Ways to access to the data stored in content repositories

4.1.2.1 Java Content Repository (JCR)

Java Content Repository (JCR) is a specification defining the model of data to be stored in content repositories. The model also proposes a Java API to access and modify the data to be used by content oriented applications. By providing a standardized model and API, JCR prevents the data isolated in proprietary repositories. In Figure 4.1, how JCR provides a common API so that different clients, applications would access to the repository through the standardized API is seen. Even though there would be more than one repository at the bottom layer in the figure, it is perceived as a single repository by the external clients. This design overcomes the problems of data isolation and code portability.

In the content repository model of JCR, the data is organized in a tree structure. A repository may have more than one tree and each tree is named as *workspace*. As suitable to the convention, each item on the tree is named as *node* except that the leaf nodes in the tree are named as *properties*. The actual content and metadata about the content reside in the properties. The non-leaf nodes provide the content managers with the opportunity of creating hierarchical content structures.

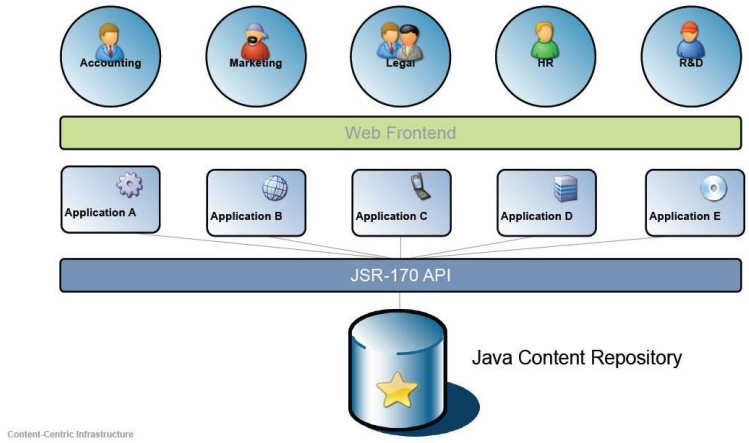

Figure 4.1: Standards-based content repository [90]

JCR offers various ways of querying the underlying data. In the first version (JCR 1.0) [5], it enforces the XPath [73] as the default query language. XPath is a language designed for retrieving the certain elements from an XML document. Considering analogy between an XML document and the hierarchical structure of the JCR model, XPath is a suitable choice. Furthermore, to support the content repositories originating from relational databases, first version of JCR supports SQL queries. To be able to execute SQL queries on the underlying content, JCR implementations provide a database view of the underlying repository. In the second version of JCR, although the methods in the first version become deprecated, they can still be used. On top of XPath and SQL, JCR 2.0 proposes an *abstract query model* along with two default implementations [91]:

- **JCR-SQL2:** This method offers a query syntax similar to SQL by mapping the abstract query model to a string serialization based on SQL.
- **JCR-LQOM (JCR Java Query Object Model):** This model is directly used within the Java programming language. Queries are constructed with Java objects corresponding with objects of the abstract query model.

4.1.2.2 Content Management Interoperability Services (CMIS)

Content Management Interoperability Services (CMIS) defines a domain model including a data model and capabilities for content management. To be used by applications working

with one or more content repository, CMIS offers a set of bindings including web services and AtomPub [6]. The domain model offered by CMIS does not aim to cover all of the content management related functionalities, instead it tries to extract functionalities common to all content repository implementations. On the contrary to the JCR specification, CMIS does not enforce a programming language, therefore it solves the interoperability problem of different content repository implementations in a larger scale.

The items managed by CMIS are named as *objects*. CMIS four types of objects. As in the case of JCR, all objects can have properties holding about the actual content. Default object types offered by CMIS is as follows:

- **Document Object:** Document objects represent the digital assets managed by the CMIS.
- **Folder Object:** Folder objects are container objects for other folder or document objects.
- **Relationship Object:** Relationship objects indicates between document and folder objects. Either the target or the source can be both document or folder objects.
- **Policy Object:** These objects are associated with the other objects indicating an administrative policy e.g access policy, modify policy, etc. about the associated object.

4.2 Contenthub Feed

Contenthub feed feature aims to synchronize the documents managed in JCR/CMIS compliant CMSes with the Contenthub component. Contenthub provides semantically management of documents by its indexing and retrieval functionalities. The synchronization process is carried out by functionalities serving for document submission to/deletion from Contenthub. This service eases the synchronization process by handling the submission and deletion operations using the identifiers, or paths of documents pertaining to the content repository. Even the actual representations of documents within the content repository can be used.

During the submission process, properties of the documents in the content repository are collected and sent to the Contenthub together with the content itself. Before indexing the content and its content repository specific properties through Solr, enhancements of the documents are

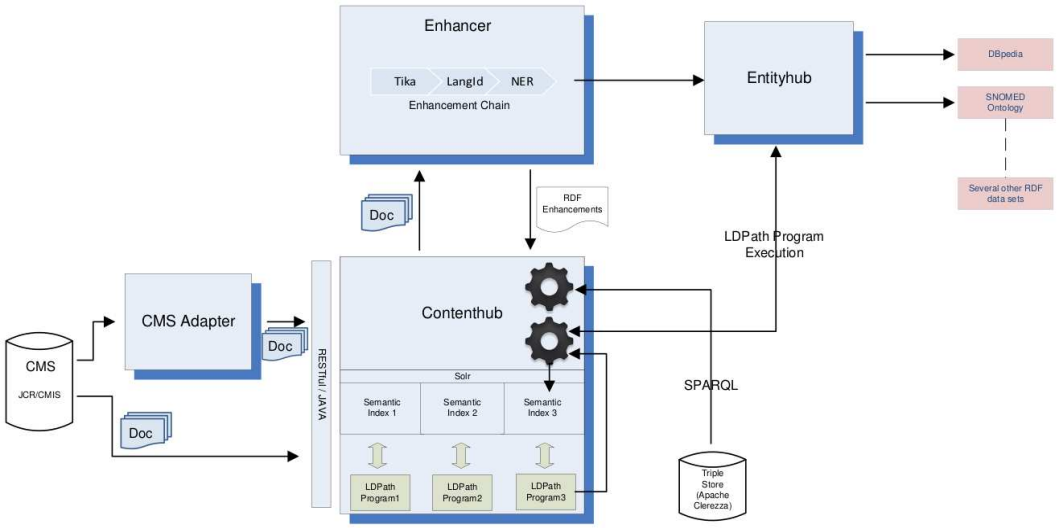

Figure 4.2: Semantic content management with Stanbol framework

also retrieved via Stanbol Enhancer. Eventually, all information is indexed and maintained in the scope of the Contenthub as explained in detail in Chapter 3. Indexing the properties of the documents allows utilization the metadata regarding documents during the search process. For instance, document classification would be possible via using the classification done by in the content repository. On top of that initial Contenthub brings more semantic metadata for the documents and provides more meaningful retrieval operations.

Figure 4.2 shows interactions which occur during the semantic content management process among the several Stanbol components. We have already covered the Contenthub related part of this diagram and now will focus on the CMS Adapter part. As seen in the figure, CMSes can directly use the Contenthub through its RESTful or Java API. As an alternative way, CMSes can also use the CMS Adapter for managing their documents in the Contenthub.

4.3 Bidirectional Mapping

4.3.1 Populating CMS

From one direction, bidirectional mapping feature makes it possible to populate content repository leveraging Linked Open Data. This provides a great easiness while populating initially content repository with certain hierarchies based on specific use cases. Thanks to this

feature, already existing RDF datasets from various domains to provide qualified classification/categorization for documents can be used. Considering the fact that more and more organizations publish their data into the Linked Open Data cloud increases importance of this feature. Apart from the already available data on the web, any RDF data can be mapped to the content repository. By mapping external RDF data, existing content repository items can be updated or new ones can be created.

Populating the CMS with the given RDF data is a process including two steps. To be able to process differrent kinds of RDF data, in the first step, the given data is transformed into a common format. The mapping from the original data to common format is done by a number of configuration as described below in the Table 4.1. Please note that some of the configurations are not used in the both directions of mapping process. Chosen resources from the external data are represented with the standard terms defined in the *CMS Vocabulary* which is explained in Section 4.3.3.

Table 4.1: Bidirectional RDF Mapping Configurations

Resource Selector	While annotating an external RDF, this configuration provides selection of resources from an RDF data. For example if this configuration is set with **rdf:type > skos:Concept**, resources having *skos:Concept* as their *rdf:type* property will be selected from the RDF data. On the other hand, while adding assertions to CMS vocabulary annotated RDF, for each resource having CMS_OBJECT_URI (from the *CMS Vocabulary*) as its rdf:type, a statement having predicate rdf:type and value skos:Concept will be added

Resource Name Predicate	While annotating an external RDF, this configuration indicates the predicate which points to the name of content repository item. A single URI such as **rdfs:label** or **http://www.w3.org/2000/01/rdf-schema#label** should be set for its value. If an empty configuration is passed, name of the content repository items will be set as the local name of the URI representing the content repository object. While adding assertions to CMS vocabulary annotated RDF, an assertion having the specified predicate will be added to RDF thanks to this configuration.
Children	This configuration specifies the children properties of content items. Value of this configuration should be like **skos:narrower > narrowerObject** or **skos:narrower > rdfs:label**. First option directly specifies the name of the child content repository item. In the second case, value *rdfs:label* predicate of resource representing the child item will be set as the name of the child item. This option would be useful to create hierarchies. It is also possible to set only predicate indicating the subsumption relations such as only **skos:narrower**. In this case name of the child resource will be obtained from the local name of URI representing this CMS object.
Default Child Predicate	This configuration is used only when generating an RDF from the repository. If there are more than one child selectors in *Children* configuration, it is not possible to know the predicate that will be used as the child assertion while adding assertions to CMS vocabulary annotated RDF. In that case, this configuration is used to set child assertion between parent and child objects. This configuration is optional. But if there is a case in which this configuration should be used and if it is not set, this causes missing assertions in the generated RDF.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://www.w3.org/2004/02/skos/core#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://dbpedia.org/resource/Category:Electric_fish">
    <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
    <j.0:broader rdf:resource="http://dbpedia.org/resource/Category:Electricity"/>
    <j.0:broader rdf:resource="http://dbpedia.org/resource/Category:Fish_sorted_by_adaptation"/>
    <rdfs:label xml:lang="en">Electric fish</rdfs:label>
    <j.0:narrower rdf:resource="http://dbpedia.org/resource/Category:Strongly_electric_fish"/>
    <j.0:narrower rdf:resource="http://dbpedia.org/resource/Category:Weakly_electric_fish"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://dbpedia.org/resource/Category:Weakly_electric_fish">
    <j.0:broader rdf:resource="http://dbpedia.org/resource/Category:Electric_fish"/>
    <rdfs:label xml:lang="en">Weakly electric fish</rdfs:label>
    <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 4.3: A sample external RDF to be mapped to the CMS

Target Path	This is the target path in the CMS. While populating the CMS with external RDF data, this path is used as the root path. The hierarchical structure emerging from the RDF is reflected under this path. On the other hand, while generating RDF from the CMS, only the content items under this path is considered.

Let's go over an example to get a clear idea on the bidirectional mapping feature. Assume that the initial external RDF is the one given in the Figure 4.3

And we have the configurations given in the Figure 4.4.

According to the given input and configurations, *skos:Concept* typed resources will be retrieved from the input data. As the name of the content items in the CMS, the value of the *rdfs:label* property of the selected resources will be used. *skos:narrower* properties of the selected resources will be considered as the property establishing the hierarchical structure. According to these mapping configurations we obtain a common format. This intermediate result is also an RDF data and it uses the terms defined in the *CMS Vocabulary* in Section 4.3.3. The intermediate result can be seen in the Figure 4.5 and they swill be mapped to the *rdfmaptest* folder of the CMS.

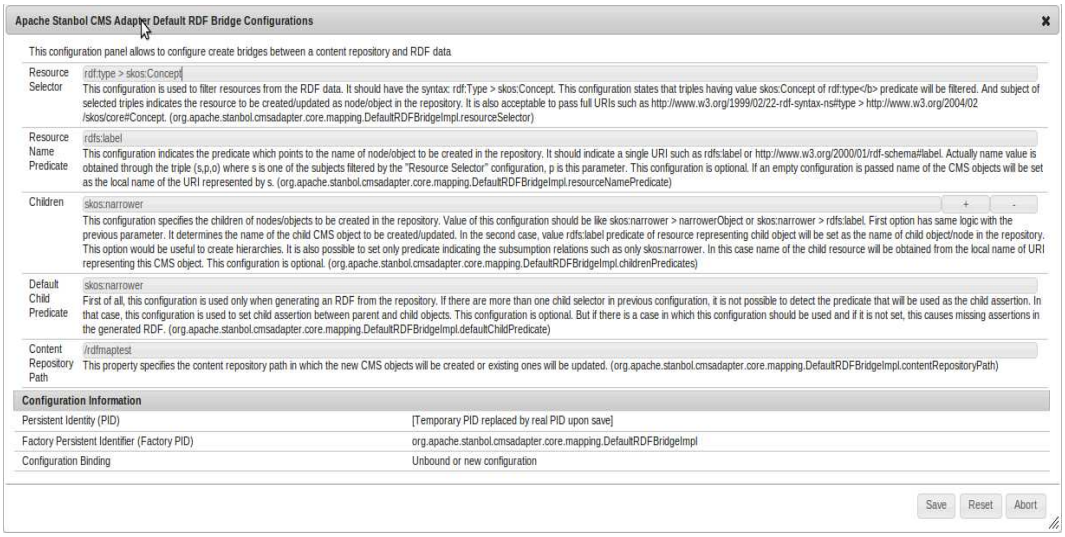

Figure 4.4: RDF mapping configurations

As seen in the Figure 4.5, there are three objects having *http://www.apache.org/stanbol/cms#CMSObject* as their *rdf:type* properties, although there are two elements in the initial input. Because, the *Electic_Fish* entity in the input has a *skos:narrower* property referencing the *Strongly_electric_fish*. Therefore, for this referenced entity, a nominal representation is created as well. These three object will be transformed under to *rdfmaptest* node in the CMS and the hierarchy between these objects will be processed by using their *http://www.apache.org/stanbol/cms#parentRef* properties. Also the names of the CMS objects will be retrieved from the *http://www.apache.org/stanbol/cms#na* property.

4.3.2 Exporting CMS

Content managers express the semantics they have in mind while defining the content items and their properties, and forming them into a particular hierarchy. However, there is no automated way to extract this semantics from the content repository structure, thus the implicit semantics given by the administrator is not formally expressed. Exporting the structure of the CMS together with the relations between the documents to RDF makes formalization of the implicit semantics possible. During this formalization process CMS Adapter does not interfere with the CMS itself and this prevents adapting CMS products internally to utilise the semantic functionalities provided by CMS Adapter.

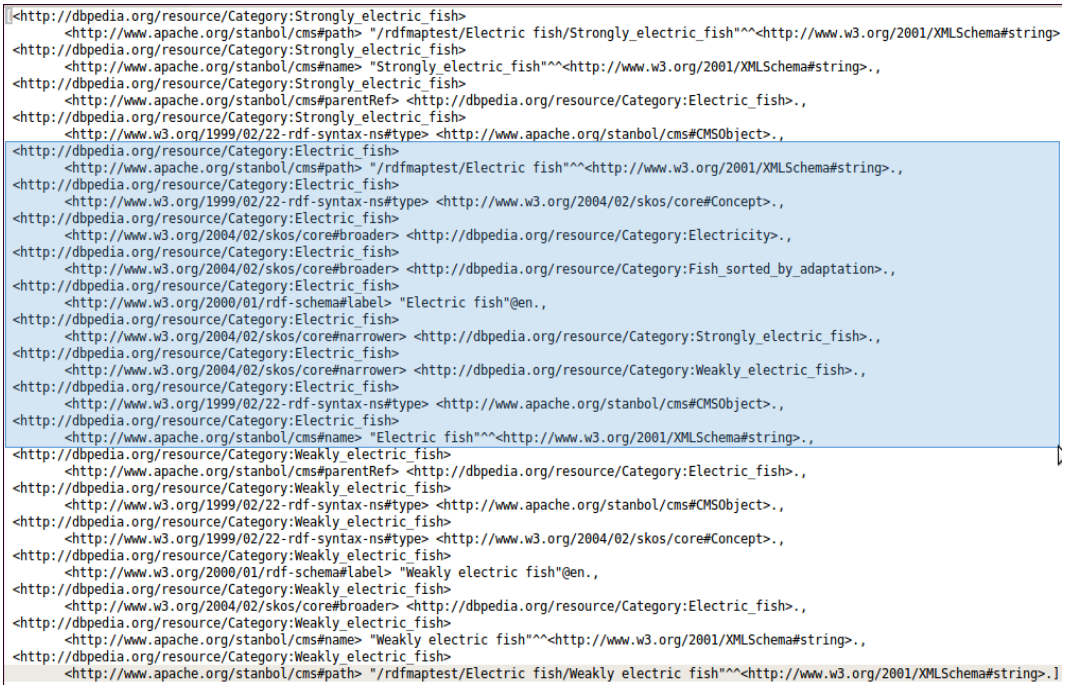

Figure 4.5: Intermediate RDF represented with common terms

In this direction, this feature enables content management systems to represent their content repository structure in RDF format. This helps building semantic services (e.g reasoning facilities on top of the existing content management systems) using their RDF representations. Moreover, that representation can be used in the *Related Keyword Search* feature of the Contenthub. Therefore, it would be possible to navigate on the documents, considering the document hierarchy in the content repository.

This process is also a two-step process. In the first step, the structure of CMS is directly transformed into an RDF. As a result, we would have CMS vocabulary annotated RDF. Based on the configurations explained in the Table 4.1 additional triples are added to the intermediate result.

4.3.3 CMS Vocabulary

This vocabulary aims to provide a standardized mapping between content repositories and RDF data. It includes a small number of terms which are used during the bidirectional mapping process. As well as general terms that are commonly used for both JCR and CMIS repositories, there are also JCR or CMIS specific terms as follows:

General Terms

- **CMS_OBJECT:** In a CMS vocabulary annotated RDF, if a resource has this URI reference as value of its rdf:type property, the subject of that resource represents a content repository item e.g a node in JCR compliant content repositories or an object in CMIS compliant content repositories.
- **CMS_OBJECT_NAME:** This URI reference represents the name of the content repository item.
- **CMS_OBJECT_PATH:** This URI reference represents the absolute path of the content repository item.
- **CMS_OBJECT_PARENT_REF:** This URI reference represents the item to be created as parent of the item having this property.
- **CMS_OBJECT_HAS_URI:** This URI reference represents the URI which is associated with the content repository item.

JCR Specific Properties

- **JCR_PRIMARY_TYPE:** This URI reference represents primary node of the content repository item associated with the resource within the RDF.
- **JCR_MIXIN_TYPES:** This URI reference represents the mixin type of the content repository item associated with the resource within the RDF.

CMIS Specific Properties

- **CMIS_BASE_TYPE_ID:** This URI reference represents the base type of the content repository item associated with the resource within the RDF.

CHAPTER 5

CASE STUDY IN HEALTHCARE DOMAIN

5.1 Preparation of Health Related Indexes

In the case study of our approach in healthcare domain, we use different health related datasets for different purposes. Before explaining where those datasets are being used, we explain datasets themselves and how they become ready to be processed by the Stanbol.

The original source of the datasets that we use in the case study is the National Center for Biomedical Ontology (BioPortal) [92]. From this portal, we use the following datasets:

- **SNOMED/CT[93]:** This is a very comprehensive clinical healthcare terminology containing terms about diagnosis, clinical findings, body structures, procedures, etc.
- **RxNORM[94]:** This dataset is about the generic and branded drugs and it aims to provide normalized names for those drugs. Also, it links the drug names to commonly used vocabularies in pharmacy management.
- **Adverse Reaction Terminology (ART) [95]:** This is a terminology aiming to provide a basis for coding of adverse reaction terms. It provides a hierarchical structure starting from body system/organ level for drug problems.

After transforming these datasets into RDF format, we have used the indexing component of Entityhub to bundle these datasets as different Solr indexes so that they can be used during the enhancement and storage operations. In the Appendix A a subsection from the RxNorm is seen.

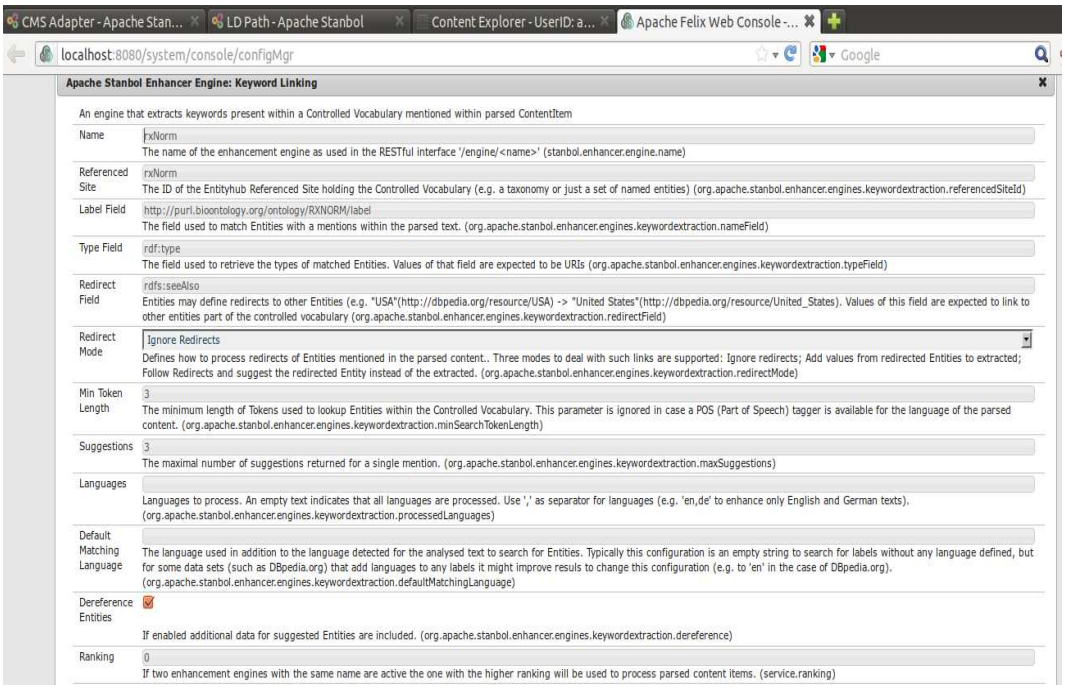

Figure 5.1: Configuring a Keyword Linking Engine for RxNORM Dataset

Indexing component of Entityhub produces a compressed zip file containing the Solr index representing the RDF dataset and a jar file which provides installation of the Solr index to the OSGi environment, where Stanbol runs, as an OSGi bundle which can be used by various components of Stanbol.

To be able recognize the named entities, which are related with the health domain, from the documents to be submitted, we need to configure the Enhancer component. This configuration is done by assigning the Solr indexes created for each RDF dataset with a separate KeywordLinkingEngine [36]. In the Figure 5.1, configuration of Keyword Linking Engine associated with the RxNORM dataset is seen. By adding a new enhancement engine for each dataset, we make Stanbol Enhancer to look up for the entities defined in the health related datasets during the document enhancement process.

5.2 Semantic Indexing of Documents

As the actual content management system to be enhanced with semantic functionalities, we have used the CRX product of Adobe [96]. CRX is a JCR compliant content management sys-

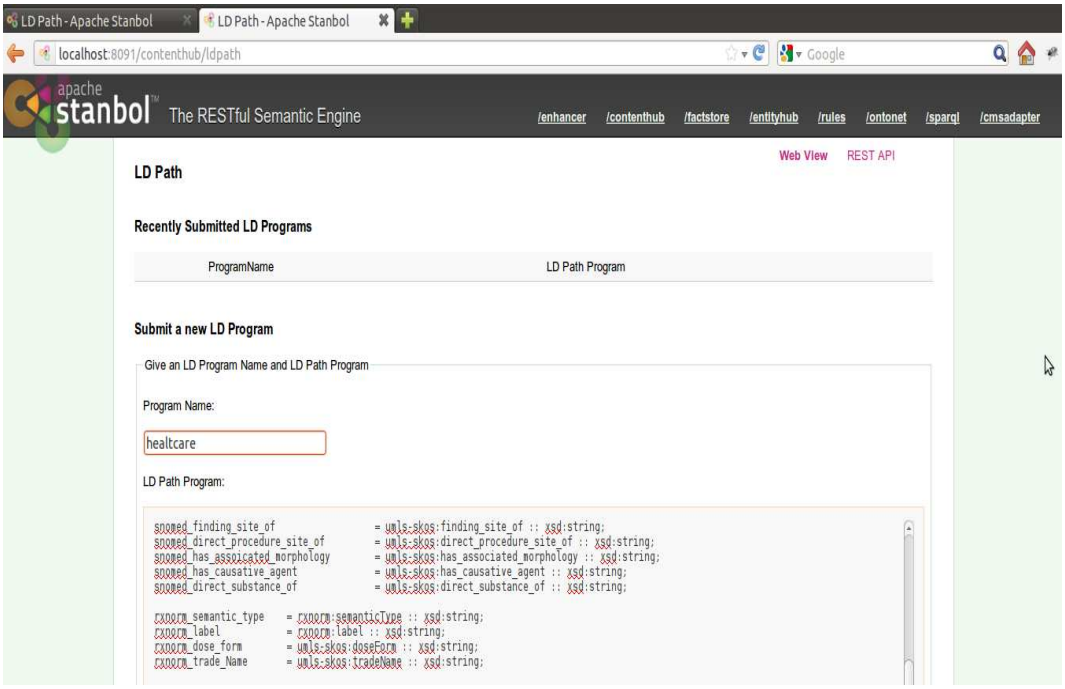

Figure 5.2: Submitting an LDPath program

tem. To simulate a content management environment working on health related documents, first we populated the system with health related documents having different topics such as cancer, diabetes, eye related diseases, etc.

The next step after populating the system with health related documents is indexing the documents in a semantic way in the scope of Stanbol's Contenthub. To do this, we create a Solr index using the LDPath [74]. To be able create an index which is compatible with the external datasets, we have analyzed the possible properties that entities of these datasets can have. Using those properties, we have created an LDPath and using the LDPath program we created a Solr index through the semantic index management functionalities of Contenthub. In the Figure 5.2, the screen for LDPath submission is seen. The index that was created with LDPath is used to index the documents managed within the CRX. The health related LDPath covering the three dataset can be seen in the Appendix B. In Appendix C, a cross-section from the configuration of Solr index, which is the index created after submitting the LDPath program, is seen. This part of the configurations indicates that each line in the LDPath program leads to a field definition in the index. Each of these index definitions are copied to the *stanbolreserved_text_all* field which is the field on which search is done. This means that in case of a keyword search, the fields created based on the LDPath program will also be considered.

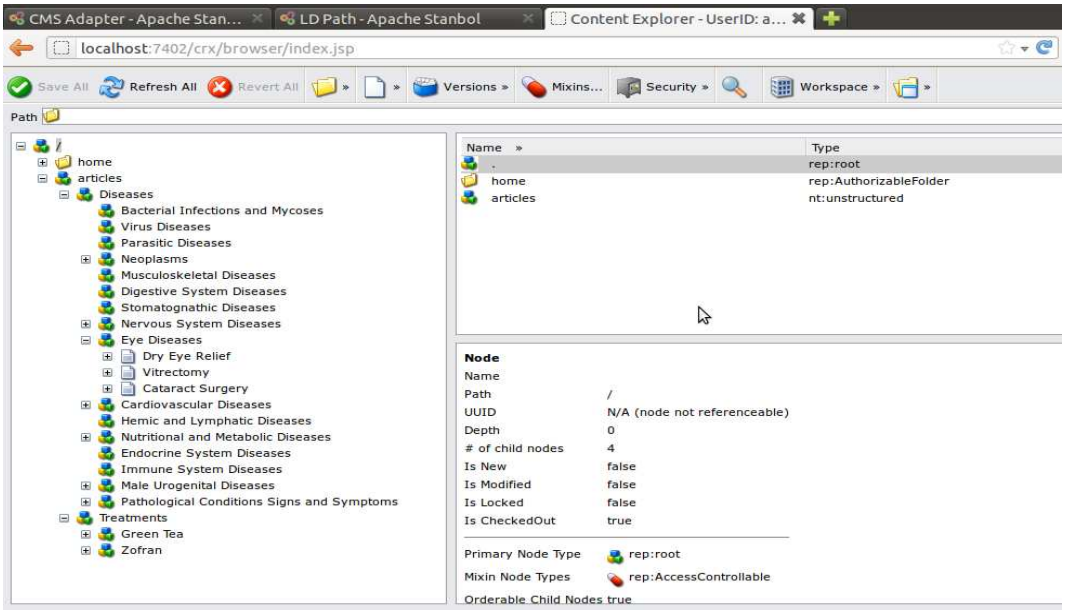

Figure 5.3: CMS Structure

After creating the semantic index, we submit the documents from CRX to Contenthub using the CMS Adapter component. As in the Figure 5.3, the health related articles are collected under the root of *articles* node. So, considering this structure we configure the CMS Adapter during the document submission process accordingly as in the Figure 5.4. As a result, all of the documents under the articles path will be submitted to the Solr index named as *healthcare*.

During the document submission process, as soon as the content arrives in the Contenthub, before any indexing operation, it is sent to Stanbol Enhancer and its enhancements are obtained. Appendix D shows a cross-section from the enhancements of a health related document, it shows the *EntityAnnotation* and *TextAnnotation* typed enhancements. The enhancements obtained in RDF format are stored in a triple store abstracted by Apache Clerezza. Enhancements of all of the documents are collected in a single RDF graph so that a SPARQL query can be executed considering all documents.

As soon as the content enhancement process is completed, Contenthub realizes one last additional semantic knowledge gathering. In this activity, Contenthub uses the named entities recognized during the content enhancement process. It requests additional knowledge for each named entity by querying the Entityhub with same LDPath program which was used to create the healthcare index. As a result, only relevant information of the entities for this use case is obtained. Appendix E shows the additional information obtained from the RxNorm

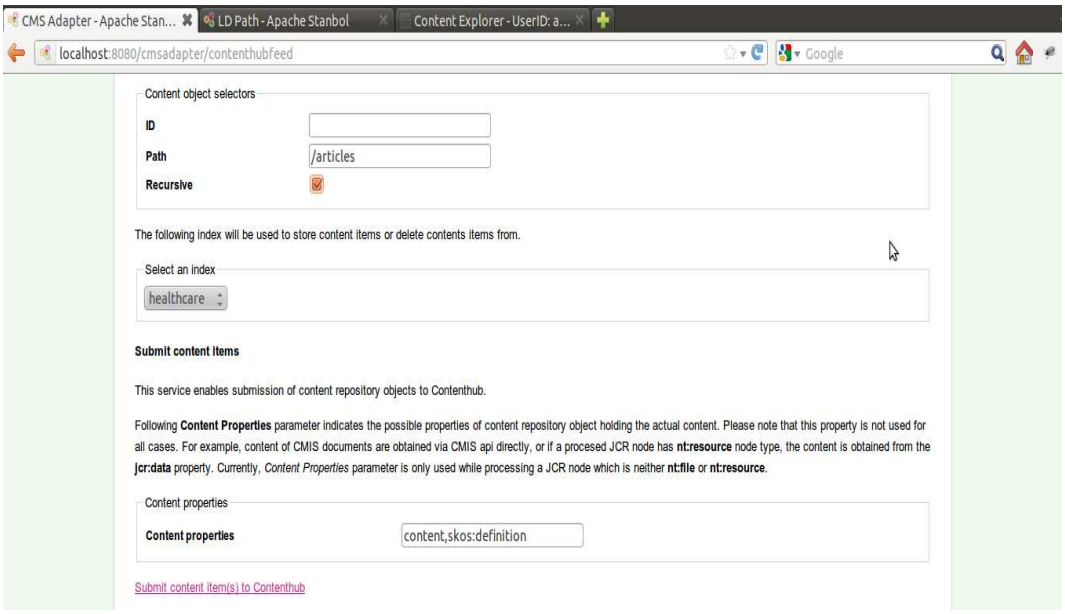

Figure 5.4: Submitting documents to Contenthub

dataset for the "*Aspirin 500 MG / Caffeine 40 MG Oral Tablet* entity. Therefore, the additional information contains values regarding to the properties that are defined for the RxNorm dataset.

At the end of the indexing process, the healthcare index, which was created considering the health specific properties, is filled with semantically meaningful information obtained from external RDF datasets. The additional information obtained for submitted documents will be used to provide semantic search functionalities for the documents.

5.3 Semantic Search over the Documents

In our evaluation, by making use of the indexed content and knowledge, we have applied faceted search for the document retrieval in a semantically meaningful way.

First, we initiated the search process by doing a keyword search with the keyword *diabetes* to get all of the documents including the diabetes keyword. As a result we obtained the results as depicted in the Figure 5.5. In addition to the documents results, on the left hand side, facets matching the results are presented. Each facet result has possible values together with number of documents that match for the corresponding value of the facets. The facets

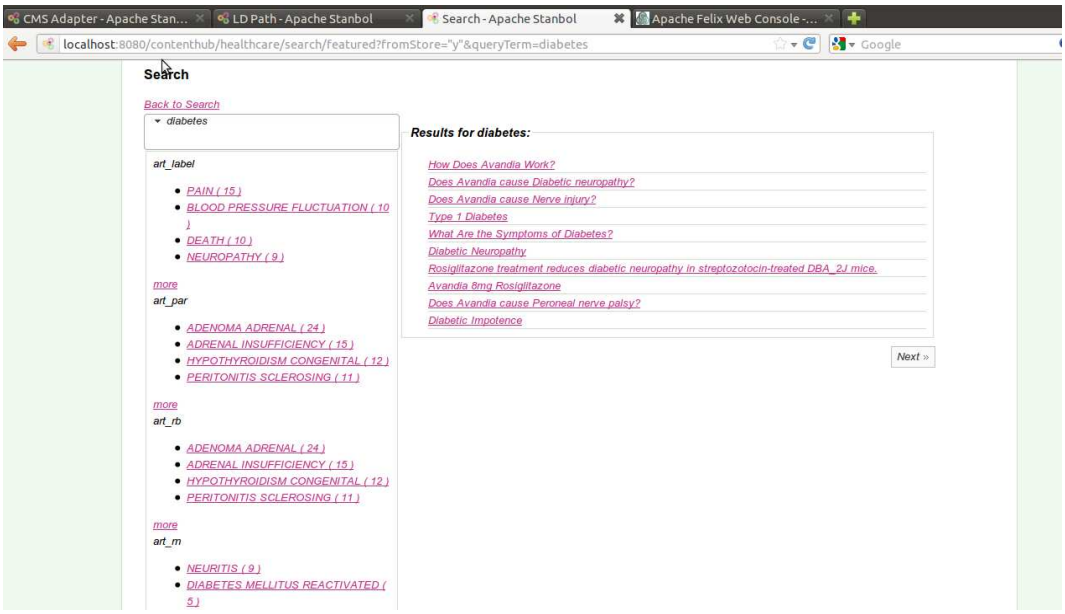

Figure 5.5: Search results for the diabetes keyword

corresponds with the fields defined in the LDPath which was used to create the healthcare index.

Facets related with all of the three datasets can be seen in the Figure 5.6.

In the next step, we constrain the documents according to finding site of diseases. For this operation, we use a field related with the SNOMED/CT dataset. Disease entities within the SNOMED/CT have a property named h*as_finding_site* which indicates the finding site of a disease within the body structure. We choose the *nerveÂ_structure* value of this facet. The meaning of this constraining operation is that remaining results, after choosing the facet value, mention about the nerve structure of the body as finding site of the diabetes or any other related disease mentioned in the documents.

In the second step of the faceted search, we would like to further constrain the documents according to a specific drug or medication. Conveniently, we use a facet related with the RxNORM dataset. We use the *rxnorm_label* facet to filter the results and choose *Avandia* value of this facet. As a result, we get the results depicted in the Figure 5.7.

In the last step of the faceted search, search results are constrained according to a specific adverse reaction by using a facet related with the adverse reaction terminology dataset. This time, we choose the *art_label* facet and choose the *Headache* value. As a result, there remains

Figure 5.6: Facets related with datasets used

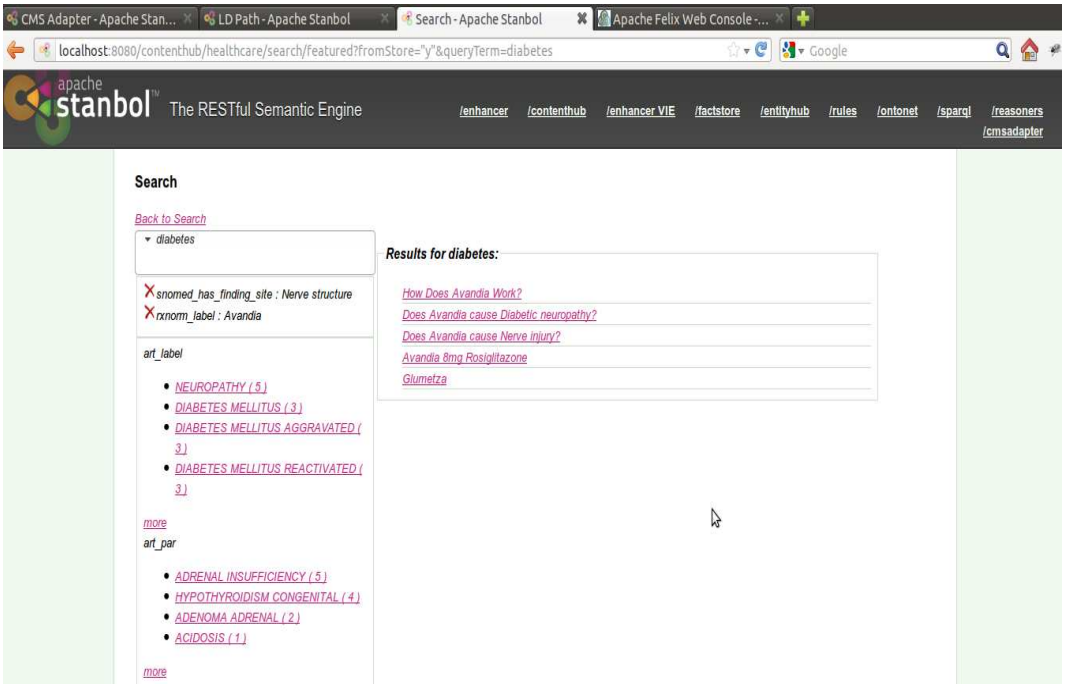

Figure 5.7: Constrained search results for Avandia constraint

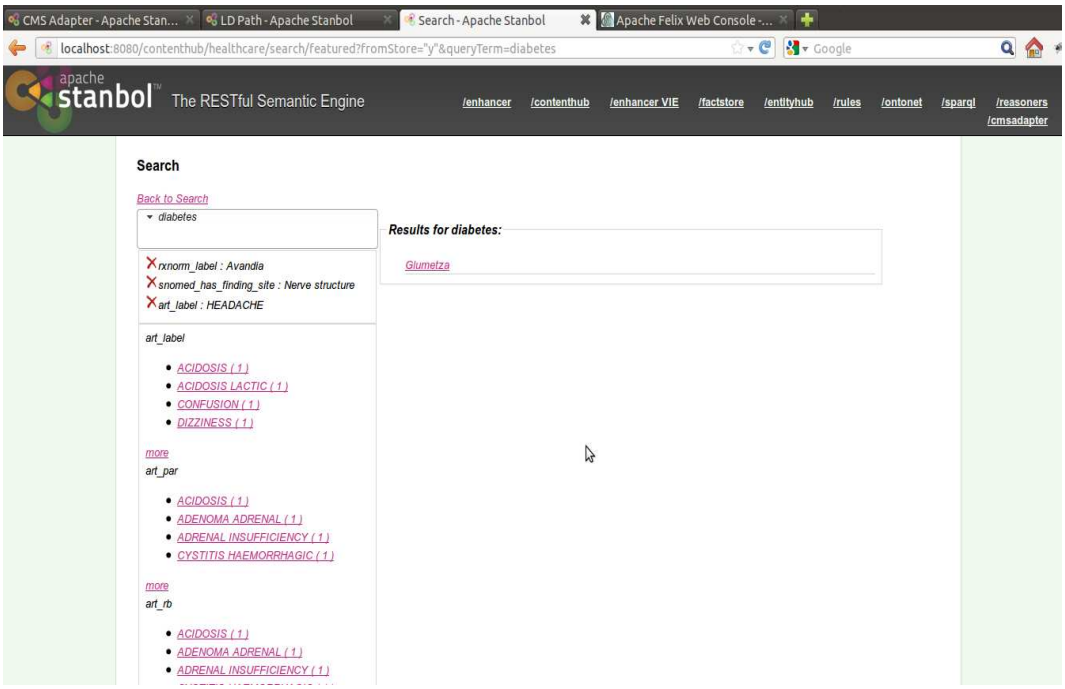

Figure 5.8: Constrained search results after selecting headache constraint

only a single document satisfying the chosen facet constraints as seen in the Figure 5.8.

The remaining document in the search results is a diabetes related document which mentions about the nerve structure of the body as finding site of the diabetes or any related disease; Avandia as a specific drug/medicament and Headache as an adverse reaction regarding the Avandia or any related drug.

In this way, we have demonstrated a semantically meaningful flow of document filtering from the health domain perspective: First a finding site regarding with a disease, then a specific drug/medicament for the disease and lastly an adverse reaction about the chosen drug. In this way, it is also possible to navigate on documents by following a different path while choosing the facet constraints.

CHAPTER 6

RELATED WORK

In this section, we give related studies with the work we present in this thesis. Considering the way of integration of several defacto and novel technologies, our study differentiates from existing approaches. The study done in [1] describes a complete reference architecture for content management systems with semantic capabilities. This is the overall architecture covering our approach. Contenthub already lies in the "Persistence" layer of this architecture. Nevertheless, as an extension to this architecture, we introduce the CMS Adapter as component interacting with JCR/CMIS compliant CMSes. This allows interacting with the "Content" column of reference architecture of a semantic CMS depicted in the Figure 6.1.

Configurable semantic bridges to extract semantics of JCR/CMIS compliant repositories into an OWL model based ontology are described in the scope of the work done in [7]. This approach proposes navigation of documents based on the class hierarchy of generated ontology expressing the semantics of the CMS. However, the study assumes that all terms to classify/categorize documents should be defined beforehand in a separate hierarchy within the CMS. Also, this is a one-way approach which only provides extracting semantic information from a CMS but not feed the CMS with semantically enriched content. In our study, apart from extracting semantics of a CMS we also update it with external RDF data. Furthermore, our approach proposes various document retrieval, search and browsing methods. As well as providing ontology based navigation, our approach also considers semantic annotations during document retrival operations.

LMF [72] is a framework offering storage and retrival functionalities for media content. LMF supports annotation of media content using the Linked Open Data and storage of the annotations along with the documents and publishes the stored content and its metadata in an

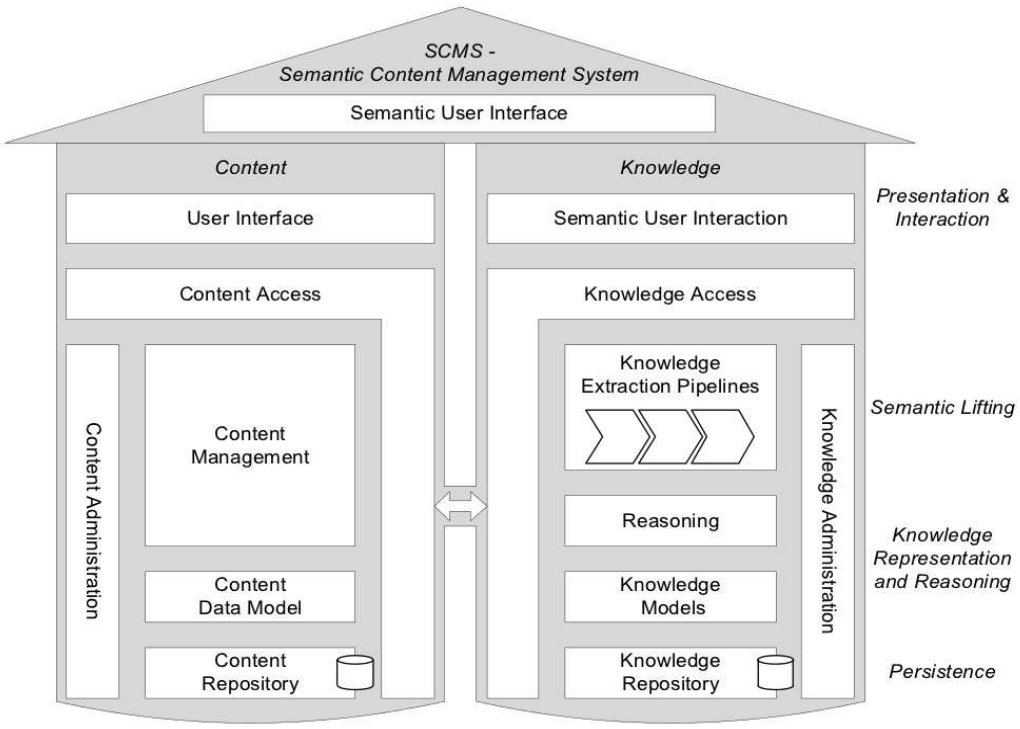

Figure 6.1: Semantic Content Management System Reference Architecture [1]

interlinked manner. Thanks to the Solr field definition language (LDPath), LMF allows creation of dynamically adaptable indexes for specific use cases. We use this dynamic index machinery as a base to storage and search capabilities of Contenthub. On top of that we propose more diverse document navigation features by means of related keywords retrieved from various sources such as WordNet, DBPedia or any external ontology. Besides, we provide services which extract content together with its metadata from JCR/CMIS compliant content repositories and manage them in our system to benefit from semantic indexing and search features.

The KAON framework which is built in the study [97] uses domain ontologies to expand original query terms. Once a query term is matched with a concept defined in the domain ontology, the query term is expanded based on the subconcepts, synonyms and associated predicates of the initial concept. This approach carries documents including more domain specific keywords to the upper lines in the search results. However, without materialization of additional metadata along with the original content, retrival of semantically related documents is achieved upto a limited extent. The proposed flexible indexing infrastructure overcomes the

limitation of dependence to query term by storing semantic metadata together with the initial content and considering this metadata during different kinds of retrieval operations.

KIM [98] stores documents together with their annotations and provides various ways to obtain documents. It allows document retrieval based on the contained named entities within documents and their attributes, even considering the ontology and knowledge base used for setting types of named entities and filling their attributes. HealthFinland [80] is a national semantic publishing network and portal. In this system, documents are provided by various content producers together with manually added annotations using the concepts defined in the health domain ontologies. HealthFinland provides faceted search by mapping ontology terms to facets manually considering the target audience. These systems provide index structures which is compatible with their annotation machinery, however we propose a framework which is also able to generate custom indexes which are suitable for specific needs of users and offer various and expandible search features.

We have compared various functionalities provided by various frameworks that are relevant to our study. As understood from the Table 6.1, there is no other framework offering a complete set of services to enhance CMSes with semantic capabilities. Especially, "Integration with CMSes" and "Dynamic Index Creation" are the features distinguishing the Stanbol from other frameworks.

Table 6.1: Comparison of relevant frameworks with Stanbol

	Named Entity Recognition	Integration with the LOD	Domain Specific Enhancements	Dynamic Index Creation	Search Query Contextualizing	Integration with CMSes
LMF	X	X		X		
KIM	X	X	X		X	
HealthFinland	X	X	X		X	
KAON					X	
Stanbol	X	X	X	X	X	X

CHAPTER 7

CONCLUSION

In this thesis, we have proposed a methodology offering semantic storage and retrieval services to be exploited by the content management systems which are not capable of managing documents together with their semantic information. The proposed approach has been realized in two of the components of Apache Stanbol project, which are the Contenthub and the CMS Adapter, respectively.

The Contenthub is the component providing semantic storage and search. It uses Apache Solr as the underlying framework. The Contenthub allows creation of semantic Solr indexes which can be adapted for any specific use case by specifying the fields to be created in the index and configuring their properties using an RDF path language, namely LDPath. Submitted documents are enhanced via the Enhancer component of Stanbol. Enhancements include detailed information about the named entities those contained in the document. Details of the named entities are retrieved using the LDPath instance from the Linked Data cloud so that they would be compliant with the custom semantic index. All of the additional, semantic knowledge is stored along with the document.

The Contenthub provides different kinds of search functionalities on the managed documents using the additional knowledge. In addition to keyword search, it is possible to perform search with structured queries. For instance, Solr indexes can be directly queried using the Solr Query syntax or SPARQL queries can be executed over the RDF graph which keeps the enhancements of all of the managed documents. The Contenthub also provides a related keyword search service which retrieves related keywords from various sources such as an arbitrary ontology, Wordnet and DBPedia (and any other RDF dataset) for the given query term.

The CMS Adapter aims to ease the document management in the Contenthub for JCR/CMIS compliant content repositories. It allows submission and deletion of documents, processing their repository specific properties. Another capability of the CMS Adapter is the mapping facility between the content repository data models and external RDF data. Given any RDF data, this feature makes it possible to update the documents residing in the content repository. From the other direction, it provides the functionality of representing the structure of the content repository in RDF format so that actual structure of content repository can be used in the semantic operations of Contenthub.

In this study, we introduce a framework which makes use of the latest developments in the information extraction, information retrieval and semantic web areas. The objective is to bring different, stand-alone implementations together and address the semantic requirements of the CMSes. In accordance with this purpose the Contenthub and CMS Adapter provide an easy way for the CMS developers to employ powerful semantic services into their implementations.

REFERENCES

[1] Fabian Christ and Benjamin Nagel. A reference architecture for semantic content management systems. In *Proceeding of the Enterprise Modelling and Information Systems Architectures Workshop 2011 (EMISA'11)*, volume P-190, pages 135–148, Hamburg, Germany, September 2011.

[2] Hui-Chuan Chu, Ming-Yen Chen, and Yuh-Min Chen. A semantic-based approach to content abstraction and annotation for content management. *Expert Systems with Applications*, 36(2, Part 1):2360 – 2376, 2009.

[3] Linking open data. `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[4] Elizabeth Orna. *Information Strategy in Practice*. Gower Publishing Limited, Aldershot, Hants, United Kingdom, 2004.

[5] Jsr 170: Content repository for javatm technology api. `http://jcp.org/en/jsr/detail?id=170`.

[6] Content management interoperability services (cmis) version 1.0. `http://docs.oasis-open.org/cmis/CMIS/v1.0/os/cmis-spec-v1.0.html`, 2009. [Online]. [Accessed: Aug. 8, 2012].

[7] G. B. Laleci, G. Aluc, A. Dogac, A. Sinaci, O. Kilic, and F. Tuncer. A semantic backend for content management systems. *Know.-Based Syst.*, 23(8):832–843, December 2010.

[8] M. Erdmann, A. Maedche, H.-P. Schnurr, and S. Staab. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In *Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, August 2000.

[9] Christian Bizer and et al. The linked data – the story so far. *Special Issue on Linked Data, International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[10] Dave Beckett and Brian McBride. Rdf/xml syntax specification (revised). `http://www.w3.org/TR/rdf-syntax-grammar/`. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[11] Dbpedia. `http://dbpedia.org/About`. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[12] Apache stanbol. `http://incubator.apache.org/stanbol/`. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[13] Suat Gönül. Contenthub (5 minutes tutorial). `http://incubator.apache.org/stanbol/docs/trunk/components/contenthub/contenthub5min`.

[14] Suat Gönül. 5 minutes documentation for cms adapter. `http://incubator.apache.org/stanbol/docs/trunk/components/cmsadapter/cmsadapter5min`.

[15] Apache stanbol components. http://incubator.apache.org/stanbol/docs/trunk/components. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[16] Ken Arnold and James Gosling. *The Java programming language (2nd ed.)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[17] Freemarker. http://freemarker.sourceforge.net/. [Online]. [Accessed: Aug. 8, 2012].

[18] The osgi architecture. http://www.osgi.org/About/WhatIsOSGi. *OSGi Alliance*. [Online]. [Accessed: Aug. 8, 2012].

[19] Osgi Alliance. *Osgi Service Platform, Release 3*. IOS Press, Inc., 2003.

[20] Apache felix. http://felix.apache.org/site/index.html. *Apache Software Foundation*.

[21] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. AAI9980887.

[22] Bill Burke. *RESTful Java with Jax-RS*. O'Reilly Media, Inc., 1st edition, 2009.

[23] Jersey. http://jersey.java.net/. [Online]. [Accessed: Aug. 8, 2012].

[24] Roy Thomas Fielding and et al. Hypertext transfer protocol - http/1.1 method definitions. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html, 2004. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[25] Apache maven. http://maven.apache.org/. [Online]. [Accessed: Aug. 8, 2012].

[26] Sling launchpad plugin. http://sling.apache.org/site/maven-launchpad-plugin.html. [Online]. [Accessed: Aug. 8, 2012].

[27] Stanbol enhancer. http://incubator.apache.org/stanbol/docs/trunk/components/enhancer. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[28] Stanbol reasoners. http://incubator.apache.org/stanbol/docs/trunk/components/reasoner. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[29] Hermit owl reasoner. http://www.hermit-reasoner.com/. [Online]. [Accessed: Aug. 8, 2012].

[30] The rdfs reasoner. http://jena.apache.org/documentation/inference/index.html\#rdfs. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[31] Stanbol ontology manager. http://incubator.apache.org/stanbol/docs/trunk/components/ontologymanager/. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[32] Sling-stanbol. https://github.com/retobg/sling-stanbol. [Online]. [Accessed: Aug. 8, 2012].

[33] Apache sling. http://sling.apache.org/site/index.html. *Apache Software Foundation.*

[34] The named entity recognition engine: detect named entities from unstructured text content. http://incubator.apache.org/stanbol/docs/trunk/components/enhancer/engines/namedentityextractionengine.html. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[35] Welcome to apache opennlp. http://opennlp.apache.org/. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[36] The keyword linking engine: custom vocabularies and multiple languages. http://incubator.apache.org/stanbol/docs/trunk/components/enhancer/engines/keywordlinkingengine.html. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[37] Kuang-Hwei (Janet) Lee-Smeltzer. Finding the needle: controlled vocabularies, resource discovery, and dublin core. *Library Collections, Acquisitions, and Technical Services*, 24(2):205 – 215, 2000.

[38] The named entity tagging engine: linking text annotations to (external) datasets of entities. http://incubator.apache.org/stanbol/docs/trunk/components/enhancer/engines/namedentitytaggingengine.html. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[39] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/, 2008. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[40] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[41] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. RFC 3986 Uniform Resource Identifier (URI): Generic Syntax. Technical report, January 2005.

[42] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.98. http://xmlns.com/foaf/spec/. [Online]. [Accessed: Aug. 8, 2012].

[43] Christian Bizer, Richard Cyganiak, and Tom Heath. How to publish linked data on the web. Web page, 2007. Revised 2008. Accessed 07/08/2009.

[44] Renato Iannella. Semantic web architectures, 2010.

[45] RDF/XML syntax specification. Available online at http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, February 2004.

[46] Frank Manola and Eric Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium, February 2004.

[47] Relationship to rdfs/owl ontologies. http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/#secmodellingrdf. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[48] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language. http://www.w3.org/TR/owl-features/, 2004. *W3C*. [Online]. [Accessed: Aug. 8, 2012].

[49] Dmitry Tsarkov. Fact++. http://owl.man.ac.uk/factplusplus/. [Online]. [Accessed: Aug. 8, 2012].

[50] Güneş Aluç. Design and implementation of an ontology extraction framework and a semantic search engine over jsr-170 compliant content repositories. Master's thesis, Middle East Technical University, June 2009.

[51] Holger Bast, Fabian Suchanek, and Ingmar Weber. Semantic full-text search with ester: Scalable, easy, fast. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, ICDMW '08, pages 959–962, Washington, DC, USA, 2008. IEEE Computer Society.

[52] Holger Bast, Alexandru Chitea, Fabian Suchanek, and Ingmar Weber. Ester: efficient search on text, entities, and relations. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 671–678, New York, NY, USA, 2007. ACM.

[53] John Davies and Richard Weeks. Quizrdf: Search technology for the semantic web. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4 - Volume 4*, HICSS '04, pages 40112–, Washington, DC, USA, 2004. IEEE Computer Society.

[54] Apache lucene. http://lucene.apache.org/core/. *Apache Software Foundation*. [Online].

[55] Apache solr. http://lucene.apache.org/solr/. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[56] Apache tomcat. http://tomcat.apache.org/. *Apache Software Foundation*. [Online]. [Accessed: Aug. 8, 2012].

[57] Glassfish. http://glassfish.java.net/. [Online]. [Accessed: Aug. 8, 2012].

[58] Liam Quin. Extensible markup language (xml). http://www.w3.org/XML/, 2009. *W3C*.

[59] Introducing json. http://www.json.org/index.html. [Online]. [Accessed: Aug. 8, 2012].

[60] Ori Ben-Yitzhak, Nadav Golbandi, Nadav Har'El, Ronny Lempel, Andreas Neumann, Shila Ofek-Koifman, Dafna Sheinwald, Eugene Shekita, Benjamin Sznajder, and Sivan Yogev. Beyond basic faceted search. In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 33–44, New York, NY, USA, 2008. ACM.

[61] Gerard Salton. Developments in automatic text retrieval. *Science*, 253(5023):pp. 974–980, 1991.

[62] Vladimir Mironov, Nirmala Seethappan, Ward Blondé, Erick Antezana, Bjorn Lindi, and Martin Kuiper. Benchmarking triple stores with biological data. *CoRR*, abs/1012.1632, 2010.

[63] DINGLEY ANDREW PETER. Storage and management of semi-structured data, 7 2003.

[64] Alice Hertel, Jeen Broekstra, and Heiner Stuckenschmidt. *RDF Storage and Retrieval Systems*.

[65] openrdf.org. `http://www.openrdf.org/`. [Online]. [Accessed: Aug. 8, 2012].

[66] Jeen Broekstra. Serql : A second generation rdf query language requirements for rdf querying implementing the requirements : Serql. *Most*, pages 1–4, 2003.

[67] York Sure Thomas Gabel and Johanna Voelker. Kaon - an overview. Technical report, Institute AIFB, University of Karlsruhe, 4 2004.

[68] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. Rql: a declarative query language for rdf. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 592–603, New York, NY, USA, 2002. ACM.

[69] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, September 2009.

[70] Welcome to apache clerezza. `http://incubator.apache.org/clerezza/`. *Apache Software Foundation.*

[71] Apache jena. `http://jena.apache.org/`. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[72] T. Kurz, S. Schaffert, and T. Bürger. Lmf: A framework for linked media. In *Workshop on Multimedia on the Web (MMWeb) at the iSemantics Conf.*, pages 16 –20, Austria, sept. 2011.

[73] Xml path language. `http://www.w3.org/TR/xpath/`. *W3C.* [Online]. [Accessed: Aug. 8, 2012].

[74] Ld path - a path-based query language for querying the linked data cloud. `http://code.google.com/p/ldpath/`. [Online]. [Accessed: Aug. 8, 2012].

[75] Justin Zobel, Alistair Moffat, and Ron Sacks-Davis. An efficient indexing technique for full text databases. In *Proceedings of the 18th International Conference on Very Large Data Bases*, VLDB '92, pages 352–362, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[76] Jian Wan and Shengyi Pan. Performance evaluation of compressed inverted index in lucene. In *Proceedings of the 2009 International Conference on Research Challenges in Computer Science*, ICRCCS '09, pages 178–181, Washington, DC, USA, 2009. IEEE Computer Society.

[77] Common field options. `http://wiki.apache.org/solr/SchemaXml#Common_field_options`. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[78] Solr query syntax. `http://wiki.apache.org/solr/SolrQuerySyntax`. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[79] Solrj. `http://wiki.apache.org/solr/Solrj`. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[80] Osma Suominen, Eero Hyvönen, Kim Viljanen, and Eija Hukka. Healthfinland-a national semantic publishing network and portal for health information. *Web Semant.*, 7(4):287–297, December 2009.

[81] M.C. Schraefel, Maria Karam, and Shengdong Zhao. mSpace: Interaction design for user-determined, adaptable domain exploration in hypermedia. In *AH 2003: Workshop on Adaptive Hypermedia and Adaptive Web Based Systems*, August 2003.

[82] A. Steven Pollitt. The key role of classification and indexing in view-based searching. In *Proceedings of the 63rd International Federation of Library Associations and Institutions General Conference (IFLA'97)*, 1997.

[83] Marti Hearst, Ame Elliott, Jennifer English, Rashmi Sinha, Kirsten Swearingen, and Ka-Ping Yee. Finding the flow in web site search. *Commun. ACM*, 45(9):42–49, September 2002.

[84] Eero Hyvönen, Eetu Mäkelä, Mirva Salminen, Arttu Valo, Kim Viljanen, Samppa Saarela, Miikka Junnila, and Suvi Kettula. Museumfinland-finnish museums on the semantic web. *Web Semant.*, 3(2-3):224–241, October 2005.

[85] Daniel Tunkelang. *Faceted Search*. Morgan & Claypool, 1st edition, 2009.

[86] Larq - free text indexing for sparql. `http://jena.sourceforge.net/ARQ/lucene-arq.html`. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[87] Christiane Fellbaum. Wordnet and wordnets. In Keith Brown, editor, *Encyclopedia of Language and Linguistics*, pages 665–670, Oxford, 2005. Elsevier.

[88] Welcome to apache jackrabbit. `http://jackrabbit.apache.org/`. *Apache Software Foundation.* [Online]. [Accessed: Aug. 8, 2012].

[89] Alice Grant. Content management systems. `http://www.ukoln.ac.uk/nof/support/help/papers/cms/`. *UKOLN*. [Online]. [Accessed: Aug. 8, 2012].

[90] Introduction to jcr. `http://www.slideshare.net/uncled/introduction-to-jcr`. [Online]. [Accessed: Aug. 8, 2012].

[91] Jsr 283: Content repository for javatm technology api version 2.0. `http://jcp.org/en/jsr/detail?id=283`.

[92] Bioportal. `http://bioportal.bioontology.org/`. [Online]. [Accessed: Aug. 8, 2012].

[93] Snomed clinical terms. `http://bioportal.bioontology.org/ontologies/1353`. [Online]. [Accessed: Aug. 8, 2012].

[94] Rxnorm. `http://bioportal.bioontology.org/ontologies/1423`. [Online]. [Accessed: Aug. 8, 2012].

[95] Who adverse reaction terminology. `http://bioportal.bioontology.org/ontologies/1354`. [Online]. [Accessed: Aug. 8, 2012].

[96] Crx - content application platform. `http://www.day.com/day/en/products/crx.html`. *Adobe.* [Online]. [Accessed: Aug. 8, 2012].

[97] Wallace Anacleto Pinheiro and Ana Maria de C. Moura. An ontology based-approach for semantic search in portals. In *Proceedings of the Database and Expert Systems Applications, 15th International Workshop*, DEXA '04, pages 127–131, Washington, DC, USA, 2004. IEEE Computer Society.

[98] Borislav Popov, Atanas Kiryakov, Angel Kirilov, Dimitar Manov, Damyan Ognyanoff, and Miroslav Goranov. KIM - Semantic Annotation Platform. *Journal of Natural Language Engineering*, 10(3-4):375–392, September 2004.

APPENDIX A

A CROSS-SECTION FROM THE RXNORM ONTOLOGY

```
<rdf:Description rdf:about=
"http://purl.bioontology.org/ontology/RXNORM/Ins_106075">
    <umls-skos:UMLS_CUI>C0357752</umls-skos:UMLS_CUI>
    <semanticType>Clinical Drug</semanticType>
    <skos:altlabel>ALUMINUM ACETATE @ 13% @ DROPS</skos:altlabel>
    <skos:altlabel>ALUMINUM ACETATE 13% DROPS</skos:altlabel>
    <umls-skos:doseForm>Otic Solution</umls-skos:doseForm>
    <umls-skos:TUI>T200</umls-skos:TUI>
    <label>aluminum acetate 130 MG/ML Otic Solution</label>
    <skos:notation>106075</skos:notation>
    <rdf:type rdf:resource=
        "http://purl.bioontology.org/ontology/RXNORM/106075"/>
</rdf:Description>
<rdf:Description rdf:about=
"http://purl.bioontology.org/ontology/RXNORM/Ins_1091144">
    <umls-skos:UMLS_CUI>C2682501</umls-skos:UMLS_CUI>
    <semanticType>Clinical Drug</semanticType>
    <umls-skos:doseForm>Extended Release Capsule</umls-skos:doseForm>
    <umls-skos:TUI>T200</umls-skos:TUI>
    <label>
        Methylphenidate Hydrochloride 10 MG Extended Release Capsule [Ritalin]
    </label>
    <skos:notation>1091144</skos:notation>
    <rdf:type rdf:resource=
```

```
        "http://purl.bioontology.org/ontology/RXNORM/1091144"/>
</rdf:Description>
<rdf:Description rdf:about=
"http://purl.bioontology.org/ontology/RXNORM/Ins_108382">
    <umls-skos:UMLS_CUI>C0361046</umls-skos:UMLS_CUI>
    <semanticType>Clinical Drug</semanticType>
    <umls-skos:doseForm>Extended Release Tablet</umls-skos:doseForm>
    <umls-skos:TUI>T200</umls-skos:TUI>
    <label>
        Aspirin 500 MG Extended Release Tablet [Anadin All-Night]
    </label>
    <skos:notation>108382</skos:notation>
    <rdf:type rdf:resource=
        "http://purl.bioontology.org/ontology/RXNORM/108382"/>
</rdf:Description>
```

APPENDIX B

LDPATH INSTANCE FOR HEALTHCARE DOMAIN

```
@prefix rdfs : <http://www.w3.org/2000/01/rdf-schema#>;
@prefix umls-skos : <http://purl.bioontology.org/ontology/umls-skos/>;
@prefix rxnorm : <http://purl.bioontology.org/ontology/RXNORM/>;
@prefix snomed : <http://purl.bioontology.org/ontology/SNOMEDCT/>;
@prefix art : <http://purl.bioontology.org/ontology/WHO/>;

health_notation = skos:notation :: xsd:string;
umls_cui = umls-skos:UMLS_CUI :: xsd:string;
tui = umls-skos:TUI :: xsd:string;

snomed_semantic_type = snomed:semanticType :: xsd:string;
snomed_label = snomed:label :: xsd:string;
snomed_isa = umls-skos:isa :: xsd:string;
snomed_ctv3id = umls-skos:ctv3id :: xsd:string;
snomed_has_specimen_substance =
    umls-skos:has_specimen_substance :: xsd:string;
snomed_has_specimen = umls-skos:has_specimen :: xsd:string;
snomed_causative_agent_of = umls-skos:causative_agent_of :: xsd:string;
snomed_specimen_substance_of =
    umls-skos:specimen_substance_of :: xsd:string;
snomed_has_component = umls-skos:has_component :: xsd:string;
snomed_active_ingredient_of =
    umls-skos:active_ingredient_of :: xsd:string;
snomed_replaces = umls-skos:replaces :: xsd:string;
```

```
snomed_replaced_by = umls-skos:replaced_by :: xsd:string;
snomed_has_finding_site = umls-skos:has_finding_site :: xsd:string;
snomed_indirect_procedure_of =
    umls-skos:indirect_procedure_of :: xsd:string;
snomed_procedure_site_of = umls-skos:procedure_site_of :: xsd:string;
snomed_part_of = umls-skos:part_of :: xsd:string;
snomed_finding_site_of = umls-skos:finding_site_of :: xsd:string;
snomed_direct_procedure_site_of =
    umls-skos:direct_procedure_site_of :: xsd:string;
snomed_has_assoicated_morphology =
    umls-skos:has_associated_morphology :: xsd:string;
snomed_has_causative_agent =
    umls-skos:has_causative_agent :: xsd:string;
snomed_direct_substance_of =
    umls-skos:direct_substance_of :: xsd:string;

rxnorm_semantic_type = rxnorm:semanticType :: xsd:string;
rxnorm_label = rxnorm:label :: xsd:string;
rxnorm_dose_form = umls-skos:doseForm :: xsd:string;
rxnorm_trade_Name = umls-skos:tradeName :: xsd:string;

art_semantic_type = art:semanticType :: xsd:string;
art_label = art:label :: xsd:string;
art_rb = umls-skos:rb :: xsd:string;
art_rn = umls-skos:rn :: xsd:string;
art_par = umls-skos:par :: xsd:string;
```

APPENDIX C

LDPATH RELATED CONFIGURATIONS OF SOLR INDEX

```
<schema>
...
<fields>
  ...
  <field indexed="true" multiValued="true"
    name="snomed_ctv3id" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
    name="snomed_active_ingredient_of" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
name="snomed_causative_agent_of" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
name="snomed_has_finding_site" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
name="art_par" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
name="snomed_has_component" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
name="rxnorm_label" stored="true" type="string"/>
  <field indexed="true" multiValued="true"
name="snomed_has_specimen" stored="true" type="string"/>
  ...
</fields>

...
```

```
<copyField dest="stanbolreserved_text_all" source="snomed_ctv3id"/>
<copyField dest="stanbolreserved_text_all"
  source="snomed_active_ingredient_of"/>
<copyField dest="stanbolreserved_text_all"
  source="snomed_causative_agent_of"/>
<copyField dest="stanbolreserved_text_all"
  source="snomed_has_finding_site"/>
<copyField dest="stanbolreserved_text_all" source="art_par"/>
<copyField dest="stanbolreserved_text_all" source="snomed_has_component"/>
<copyField dest="stanbolreserved_text_all" source="rxnorm_label"/>
<copyField dest="stanbolreserved_text_all" source="snomed_has_specimen"/>
...

</schema>
```

APPENDIX D

A CROSS-SECTION ENHANCEMENT OF A HEALTH RELATED DOCUMENT

```
<rdf:Description rdf:about=
"urn:enhancement-d956c7ff-c193-4e95-3a42-477fbc81ec9f">
  <rdf:type rdf:resource=
    "http://fise.iks-project.eu/ontology/Enhancement"/>
  <rdf:type rdf:resource=
    "http://fise.iks-project.eu/ontology/TextAnnotation"/>
  <j.10:extracted-from rdf:resource=
    "urn:content-item-12fb1d64-c5f4-4e6f-9f63-d9cfc87a3fd2"/>
  <j.7:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2012-05-17T17:46:30.753Z
  </j.7:created>
  <j.7:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    org.apache.stanbol.enhancer.engines.keywordextraction.engine.
    KeywordLinkingEngine
  </j.7:creator>
  <j.10:start rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#integer">3993</j.10:start>
  <j.10:end rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
    4012
  </j.10:end>
  <j.10:selection-context xml:lang="en">)
    Computed tomography (CT) scan Ultrasound Biopsy Surgery.
```

```
  </j.10:selection-context>
  <j.10:selected-text xml:lang="en">scan Ultrasound</j.10:selected-text>
  <j.10:confidence rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    0.722500040531159
  </j.10:confidence>
</rdf:Description>
<rdf:Description rdf:about=
"urn:enhancement-3ff4502a-f622-9a21-67be-cdf8d7313e47">
  <rdf:type rdf:resource="http://fise.iks-project.eu/ontology/Enhancement"/>
  <rdf:type rdf:resource=
    "http://fise.iks-project.eu/ontology/EntityAnnotation"/>
  <j.10:extracted-from rdf:resource=
    "urn:content-item-12fb1d64-c5f4-4e6f-9f63-d9cfc87a3fd2"/>
  <j.7:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2012-05-17T17:46:30.753Z
  </j.7:created>
  <j.7:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    org.apache.stanbol.enhancer.engines.keywordextraction.engine.
    KeywordLinkingEngine
  </j.7:creator>
  <j.10:confidence rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
    0.722500040531159
  </j.10:confidence>
  <j.7:relation rdf:resource=
    "urn:enhancement-d956c7ff-c193-4e95-3a42-477fbc81ec9f"/>
  <j.10:entity-reference rdf:resource=
    "http://purl.bioontology.org/ontology/SNOMEDCT/Ins_146511005"/>
  <j.10:entity-label>Ultrasound scan</j.10:entity-label>
</rdf:Description>
<rdf:Description rdf:about=
"urn:enhancement-96846c41-6f28-5274-45d3-7d7b3745a620">
  <rdf:type rdf:resource="http://fise.iks-project.eu/ontology/Enhancement"/>
  <rdf:type rdf:resource=
```

```
    "http://fise.iks-project.eu/ontology/TextAnnotation"/>
  <j.10:extracted-from rdf:resource=
    "urn:content-item-12fb1d64-c5f4-4e6f-9f63-d9cfc87a3fd2"/>
  <j.7:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
    2012-05-17T17:46:26.063Z
  </j.7:created>
  <j.7:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    org.apache.stanbol.enhancer.engines.langid.LangIdEnhancementEngine
  </j.7:creator>
  <j.7:language>en</j.7:language>
</rdf:Description>
```

APPENDIX E

DOMAIN SPECIFIC ENTITY PROPERTIES


```
{
    "id": "http:\/\/purl.bioontology.org\/ontology\/RXNORM\/Ins_248628",
    "rxnorm_semantic_type": [{
        "type": "text",
        "value": "Clinical Drug"
    }],
    "health_notation": [{
        "type": "text",
        "value": "248628"
    }],
    "umls_cui": [{
        "type": "text",
        "value": "C0790292"
    }],
    "tui": [{
        "type": "text",
        "value": "T200"
    }],
    "http:\/\/stanbol.apache.org\/ontology\/entityhub\/query#score": [{
        "type": "value",
        "xsd:datatype": "xsd:double",
        "value": 1
    }],
    "rxnorm_dose_form": [{
```

```
        "type": "text",
        "value": "Oral Tablet"
    }],
    "rxnorm_label": [{
        "type": "text",
        "value": "Aspirin 500 MG \/ Caffeine 40 MG Oral Tablet"
    }]
}
```