DESIGN AND IMPLEMENTATION OF AN ONTOLOGY EXTRACTION FRAMEWORK AND A SEMANTIC SEARCH ENGINE OVER JSR-170 COMPLIANT CONTENT REPOSITORIES

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

BY

GÜNEŞ ALUÇ

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

JUNE 2009

Approval of the thesis:

DESIGN AND IMPLEMENTATION OF AN ONTOLOGY EXTRACTION FRAMEWORK AND A SEMANTIC SEARCH ENGINE OVER JSR-170 COMPLIANT CONTENT REPOSITORIES

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

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

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering** _____

Prof. Dr. Asuman Doğaç
Supervisor, **Department of Computer Engineering, METU** _____

Assoc. Prof. Dr. Nihan Kesim Çiçekli
Co-supervisor, **Department of Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu
Department of Computer Engineering, METU _____

Prof. Dr. Asuman Doğaç
Department of Computer Engineering, METU _____

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

Assoc. Prof. Dr. Ahmet Coşar
Department of Computer Engineering, METU _____

Yıldıray Kabak
SRDC Ltd. _____

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: GÜNEŞ ALUÇ

Signature :

ABSTRACT

DESIGN AND IMPLEMENTATION OF AN ONTOLOGY EXTRACTION FRAMEWORK AND A SEMANTIC SEARCH ENGINE OVER JSR-170 COMPLIANT CONTENT REPOSITORIES

Aluç, Güneş
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Asuman Doğaç
Co-Supervisor : Assoc. Prof. Dr. Nihan Kesim Çiçekli

June 2009, 117 pages

A Content Management System (CMS) is a software application for creating, publishing, editing and managing content. The future step in content management system development is building intelligence over existing content resources that are heterogeneous in nature. Intelligence collected at the knowledge base can later on be used for executing semantic queries. Expressing the relations among content resources with ontological formalisms is therefore the key to implementing such semantic features.

In this work, a methodology for the semantic lifting of JSR-170 compliant content repositories to ontologies is devised. The fact that in the worst case JSR-170 enforces no particular structural restrictions on the content model poses a technical challenge both for the initial build-up and further synchronization of the knowledge base. To address this problem, some recurring structural patterns in JSR-170 compliant content repositories are exploited. The value of the ontology extraction framework is assessed through a semantic search mechanism that is built on top of the extracted ontologies. The work in this thesis is complementary to the “Interactive Knowledge Stack for small to medium CMS/KMS providers (IKS)” project

funded by the EC (FP7-ICT-2007-3).

Keywords: Content Management System, ontology extraction, Java Content Repository, semantic search

ÖZ

JSR-170 UYUMLU İÇERİK HAVUZLARI ÜZERİNDE ONTOLOJİ ÇIKARIM İSKELETİ VE ANLAMSAL ARAMA MOTORUNUN TASARIM VE UYGULANMASI

Aluç, Güneş
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Asuman Doğaç
Ortak Tez Yöneticisi : Doç. Dr. Nihan Kesim Çiçekli

Haziran 2009, 117 sayfa

İçerik Yönetim Sistemleri; içeriğin oluşturulması, yayınlanması, düzenlenmesi ve yönetilmesine olanak sağlayan yazılımlardır. Gelecekte hedeflenen ise bu sistemlerin heterojen yapıdaki mevcut içerikten bilgisayar tarafından işlenebilir anlamsal bilgiyi çıkarabilecek düzeye getirilmesidir. Böylelikle, bilgi tabanı kullanılarak anlamsal sorgular yapılabilecektir. Bu ve benzeri niteliklerin sağlanabilmesi için öncelikle içerik kaynakları arasındaki ilişkilerin ontolojilerle ifade edilmesi gerekmektedir.

Bu çalışma kapsamında JSR-170 uyumlu içerik havuzlarının anlamsal olarak ontolojilere yükseltilmesini sağlayacak metodolojiler geliştirilmektedir. Fakat JSR-170 modelinin içerik yapısı üzerinde belki de gereğinden fazla sunduğu esneklik, gerek bilgi tabanının oluşturulması gerek senkronizasyonun sağlanması karşısında bir engel oluşturmaktadır. Bu sorunu çözmek için, JSR-170 uyumlu içerik havuzlarında sık kullanılan birtakım yapısal desenlerden faydalanılmıştır. Geliştirilen bu ontoloji çıkarım iskeletinin katma değeri ise ontolojiler üzerinde çalışan anlamsal arama motoru aracılığıyla değerlendirilmiştir. Bu tez çalışması, Avrupa Komisyonu tarafından desteklenen “Interactive Knowledge Stack for small to medium CMS-

/KMS providers (IKS)” projesini (FP7-ICT-2007-3) tamamlayıcı niteliktedir.

Anahtar Kelimeler: İçerik Yönetim Sistemleri, ontoloji çıkarım, JSR-170, anlamsal arama

To my dearest sister, Deniz...

ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to my supervisor, Prof. Dr. Asuman Doğaç, 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, Assoc. Prof. Dr. Nihan Kesim Çiçekli, 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 in preparing the prototype.

I am deeply grateful to my family for their love and support. Without them, this work could not have been completed.

I am highly indebted to my friends, Tuncay Namlı, Mustafa Yüksel, Mehmet Olduz, Yıldıray Kabak and all the other colleagues at the Software Research and Development Center, whose help, stimulating suggestions and encouragement helped me at all times in this research.

I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing the financial means to sustain this work.

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

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABox	Assertional Knowledge
CMIS	Content Management Interoperability Services
CMS	Content Management System
CND	The Compact Namespace and Node Type Definition
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer
DIG	Description Logic Implementation Group
HTTP	Hypertext Transfer Protocol
IKS	Interactive Knowledge Stack for small to medium CMS/KMS providers
IPTC	International Press Telecommunications Council
IR	Information Retrieval
JAR	Java Archive
JCR	Java Content Repository
JSR-170	The Content Repository for Java technology API
JSR-283	The Content Repository for Java technology API Version 2.0
JTA	The Java Transaction API
OWL	The Web Ontology Language
RDBMS	Relational Database Management System
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
REST	Representational State Transfer

RIA	Rich Internet Application
RQL	RDF Query Language
SAIL	The Storage and Inference Layer
SeRQL	Sesame RDF Query Language
SPARQL	The SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TBox	Terminological Knowledge
URI	Uniform Resource Identifiers
W3C	The World Wide Web Consortium
XML	The Extensible Markup Language
XPATH	XML Path Language
XSD	XML Schema Definition

CHAPTER 1

INTRODUCTION

"Content is the feeder mechanism for all business processes. And always has been [1]."

Content management is becoming more essential for businesses as the amount of digital content continues to grow [1]. In 1992, there were just 1,000 pages on the Web. As of June 2000, over two billion Web pages were posted on the Internet [2]. Recently, Google has announced that its index reached the "one trillion" mark [3]!

Content management systems have taken the responsibility to manage such growing volumes of enterprise content resources. In this respect, a content management system (CMS) is an integrated environment to manage content acquired from different data sources. The primary task of a content management system is to organize content in a hierarchy so that user requests are handled more efficiently. Metadata based search indices are built to help categorize content and respond to user queries [4].

The content management lifecycle contains a collection of iterative processes that start with content acquisition and end with the delivery or publishing of content [1]. Throughout the lifecycle, content management systems permit content to be modeled and edited, search indices to be built, and finally content to be searched for. On the other hand, even though there are ongoing efforts to integrate semantic features, they are far from making the whole lifecycle semantically enabled. For example, state-of-the-art content management systems support metadata extraction to a degree that facilitates keyword-based search and content categorization; yet, enhancements are necessary for full alignment with the domain knowledge. Furthermore, what is commercialized as "semantic search" is merely based on some ad-hoc techniques such as synonym matching, similarity search and "Did you mean this?"

suggestions. However, really, support for structural search as well as for faceted-navigation is desired.

To overcome these limitations, first, we propose to formulate the structural relations present in the content repository by ontologies; where, an ontology is an explicit specification of a conceptualization [5]. Even though at an initial stage the ontologies represent no further semantics than what is already implied by the model, the content management system does not provide any means for the automated processing of the relations; hence, semantic lifting is unavoidable. Furthermore, such formalization provides the basis for alignment with other domain and horizontal knowledge.

As the second step, we propose a semantic search mechanism that combines the full-text search capabilities of content management systems with the knowledge accumulated in the knowledge base. As argued in [6], ontologies represent human knowledge explicitly in a form that is suitable for automated processing and if used in combination with full-text search, then enhanced semantic search features can be supported.

In this regard, the work in this thesis can be evaluated in two dimensions; that is, the design and implementation of both an ontology extraction framework and a semantic search engine. The suggested solution works on top of JSR-170 compliant content repositories, where JSR-170 offers interface functions to propriety content repository implementations. The work is complementary to the "Interactive Knowledge Stack for small to medium CMS/KMS providers (IKS)" project funded by the EC (FP7-ICT-2007-3). In summary, our aim is to:

- Exploit the power of ontologies that provide machine interpretable means to express and process the semantic information in the CMS content model,
- Establish a bridge between the content repository and the knowledge base,
- Provide a formal basis for alignment with extra domain knowledge,
- Seamlessly integrate structural search facilities into keyword-driven interfaces,
- Provide support for faceted-navigation of content resources.

This thesis is organized as follows. First, we discuss the various enabling technologies: on the one hand, we have the content repositories and on the other hand we have the knowledge

representation schemes and ontology engineering tools. In Chapter 3, we outline the various issues related to the ontology extraction process, the content modeling patterns we have exploited and finally the mapping schemes in our JCR-to-Onto Bridge solution. In Chapter 4, the implementation details of our hybrid search mechanism are presented. The power of the JCR-to-Onto Bridge framework is demonstrated in Chapter 5 together with the semantic search features enabled when the proposed hybrid search engine is executed on top of the learnt ontology. Our discussion then proceeds with a comparison of our approach with the related work in this area. Finally, we summarize our efforts and outline the possible future research directions. It is worth mentioning that chapters 3, 4 and 5 can be read on their own; that is, they have their introduction, body and conclusion sections.

CHAPTER 2

BACKGROUND ON ENABLING TECHNOLOGIES AND STANDARDS

2.1 Ontology Representation Languages

As explained in [5], an ontology is an explicit specification of a conceptualization. Ontologies describe the set of objects and the relationships among these objects through a representational vocabulary that allows knowledge-based programs to make inferences. In this respect, various representational schemes have evolved, whose details are discussed in the following sections.

2.1.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a language for representing information on the Web [7]. It is a family of World Wide Web Consortium (W3C) [8] specifications. Although the primary intention has been to represent metadata about Web resources (i.e. title, author, and modification date of a Web page); the concept of a "Web resource" can be generalized to account for a more general modeling or representation.

The RDF data model consists of subject-predicate-object expressions called RDF triples formalized as P(S; O); that is, a subject S has a predicate (or property) P with value O, where S and P are Uniform Resource Identifiers (URI)s [9] and O is either a URI or a literal value. RDF allows subjects and objects to be interchanged allowing an object from one triple to play the role of a subject in another triple [10]. Collectively, these chain of RDF triples form a graph of nodes and arcs (Figure 2.1) representing the resources, their properties and property values [11].

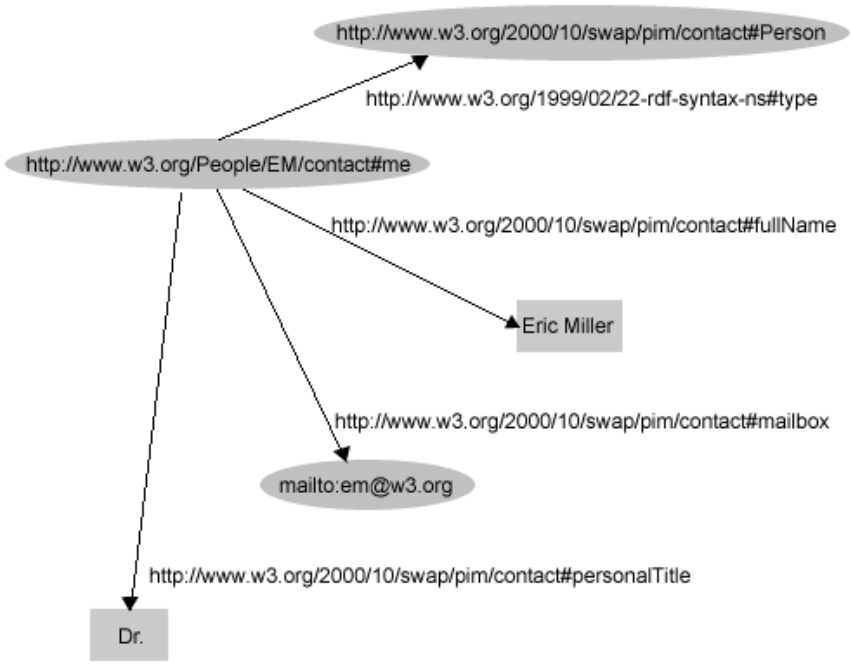

Figure 2.1: An RDF graph describing Eric Miller [11]

The value of RDF is in its power to express simple statements about resources where this information is intended to be processed by applications. Furthermore, RDF provides a common framework for expressing this information so that it can be exchanged between applications without a loss of meaning [11].

2.1.2 Resource Description Framework Schema (RDF-S)

Apart from being able to make simple statements about resources, communities also need the ability to define the vocabulary or equivalently the terminology they intend to use along with their statements. The terminology includes class definitions, property definitions and interrelations among both class and property instances. The RDF Vocabulary Description Language 1.0: RDF Schema provides a mechanism to define such terminologies [12]. Interestingly, the specification itself explains how to use RDF triples to describe these RDF vocabularies. As explicitly stated in [10], the only difference with “normal” RDF expressions is that in RDF-S an agreement is made on the semantics and the interpretation of certain terms and statements.

Classes: It is intuitive to divide the resources into groups called classes. Classes, too, are represented as RDF resources. Therefore, it is necessary to differentiate between resources which are RDF-S classes and which are not. The subject-predicate-object triple "*MotorVehicle* **rdf:type** *rdfs:Class*" has the meaning that the resource MotorVehicle is in fact an RDF-S class. Following this, the triple "*companyCar* **rdf:type** *MotorVehicle*" describes companyCar as a member or equivalently an instance of the MotorVehicle class [11]. Please note that in this example, "rdfs:Class" is a reserved term and has an agreed semantics and interpretation. The class and instance relations are expressed with "normal" RDF triples.

Properties: The properties of a class are specific characteristics that describe the members of the class. In RDF Schema, properties are described using the RDF class "rdf:Property", and the RDF Schema properties "rdfs:domain", "rdfs:range" [11]. Consider the following triples:

MotorVehicle	**rdf:type**	*rdfs:Class*
VehicleParts	**rdf:type**	*rdfs:Class*
hasPart	**rdf:type**	*rdf:Property*
hasPart	**rdfs:domain**	*MotorVehicle*
hasPart	**rdfs:range**	*VehicleParts*

In this example, *MotorVehicle* and *VehicleParts* are RDF-S classes. *hasPart* is defined as an *rdf:Property* and the domain and range restrictions indicate that the RDF statements (triples) using the *hasPart* property have instances of *MotorVehicle* as subjects and instances of *VehicleParts* as objects.

In addition to the aforementioned constructs, "rdfs:subClassOf" and "rdfs:subPropertyOf" are used to define subsumption relations among classes and properties, respectively.

2.1.3 Web Ontology Language (OWL)

The Web Ontology Language (OWL) [13] is a revision of the DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL) [14] web ontology language. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms, called ontologies [13], [15]. Although RDF Schema has a similar ob-

jective, in most cases, its semantics is inadequate to perform useful reasoning tasks. In fact, OWL has an enhanced syntax for expressing meaning and semantics than Extensible Markup Language (XML) [16], RDF, and RDF-S. For a full list of constructs, please refer to [13].

OWL Lite, OWL DL and OWL Full are respectively the three increasingly expressive sublanguages of OWL. OWL Lite is the least expressive of all and does not go beyond the definition of classification hierarchies and some simple constraints such as restricted cardinality constraints (i.e. while it supports cardinality constraints, it only permits cardinality values of 0 or 1 [13].). On the other hand, OWL DL provides maximum expressiveness provided that computational completeness and decidability can be guaranteed. Finally, OWL Full removes all computational guarantees to obtain full expressiveness. The choice between the three sublanguages depends on the extent to which users require the expressiveness provided by each. The choice between OWL DL and OWL Full also depends on whether the users require reasoning to be computationally complete and decidable or not.

2.2 Knowledge Persistence and Access

We have seen in Section 2.1 that the intuition behind various knowledge representation schemes was to enable the processing of the asserted knowledge by applications. In this respect, mechanisms should exist for the storage, querying of and inferencing over the statements made about resources.

A triplestore is a purpose-built database for the storage and retrieval of RDF triples; although it behaves much like a relational database, it is optimized specifically for the storage and retrieval of subject-predicate-object triples. Given some thought, a triplestore may either provide a generic model to support the storage and retrieval of instances of any RDF Schema or be tailored to support the fast-retrieval of instances of a particular schema only. The latter sacrifices support for a generic representation in exchange of performance.

Following are some of the state-of-the art frameworks for the storage, inferencing and querying of RDF triples:

2.2.1 Sesame

Sesame is an open source framework for the storage, querying and inferencing of the RDF data [17]. The overall architecture of Sesame is depicted on Figure 2.2. Sesame may be used as a persistence system for RDF and RDF Schema or as a library for applications that need to access and work with RDF internally [18]. OWL ontologies are simply treated on the level of RDF graphs [19]. Sesame decouples the access layer from the actual storage mechanism chosen (e.g. relational databases, in-memory, filesystems, keyword indexers, etc.) by means of its Storage and Inference Layer (SAIL) [10].

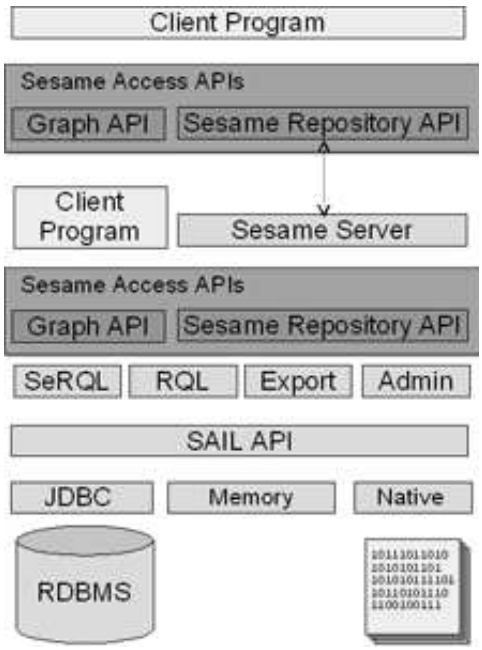

Figure 2.2: The Sesame architecture [20]

Sesame enables querying through a declarative query language called the RDF Query Language (RQL) [21] and is currently providing support for the Sesame RDF Query Language (SeRQL) [22]; a query language that has its foundations in RQL. As outlined in [21], queries over RDF documents and RDF schemata may be evaluated in three levels of abstraction: syntactic level, structural level and semantic level. Neither the syntactic nor the structural level information is expressive enough to retrieve facts implicitly inferred through the RDF-S semantics. In this respect, querying at the semantic level requires some preprocessing so as to

extract the full knowledge that an RDF-S description entails. RQL works towards this goal.

Sesame has support for three basic types of reasoning: the RDF Schema reasoning, “custom reasoning” and OWLIM [23] reasoning. RDF Schema reasoning works as follows: Given a set of RDF and/or RDF Schema, Sesame finds the implicit information and adds it to the repository as the facts are being asserted [24]. However, it is limited to the RDF-S semantics and therefore does not support reasoning over user-defined transitive, symmetric or inverse properties. The “custom reasoning” approach enables RDF Schema reasoning to be extended with user-defined inference rules [25]. Finally, OWLIM is a specific Storage and Inference Layer (SAIL) for Sesame which supports partial reasoning over OWL Description Logic Programs (OWL DLP) [26].

2.2.2 Jena2

Jena [27] is an open-source Java framework that provides in-memory or persistent storage and an abstraction over RDF graphs [28]. Its rich application programming interface for manipulating RDF graphs serves as a toolkit for semantic web programming. It provides additional support for working with RDF-S and OWL representations.

The choice of storage in Jena may be backed by either a relational or an object database. However, the challenge is to provide storage in an efficient and flexible manner. For this purpose, different models have been proposed. The normalized approach treats subjects and predicates as having URI values whereas for an object the choice is between a URI and a literal value. Statements, literal values and resources are therefore stored in separate tables shown on Figure 2.3.

2.3 Ontology Engineering

Protégé [29] and swoop [30] are among the various ontology engineering frameworks designed for creating, editing and debugging ontologies. Here, our discussion will be on the Protégé framework.

Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development [29]. Protégé-OWL is

Statement Table

Subject	Predicate	ObjectURI	ObjectLiteral
201	202	null	101
201	203	204	null
201	205	101	null

Literals Table

Id	Value
101	Jena2
101	The description - a very long literal that might be stored as a blob.

Resources Table

Id	URI
201	mylib:doc
202	dc:title
203	dc:creator
204	hp:JenaTeam
205	dc:description

Figure 2.3: The Jena schema (normalized) [28]

one such developed plug-in that can be used to edit ontologies in OWL, to access description logic reasoners, and to acquire instances for semantic markup [31]. As discussed in [32], Protégé-OWL supports the three species of OWL; namely, OWL-Lite, OWL-DL and OWL-Full. With Protégé-OWL, named classes and properties (i.e. functional, inverse functional, transitive, symmetric) can be created, classes can be set as disjoint. By connecting to an external reasoner, it is possible (i) to compute the inferred ontology class hierarchy and (ii) to check whether or not it is possible for the class to have any instances (i.e. consistency checking).

2.4 Overview of Reasoners

In their paper titled “Benchmarking OWL Reasoners”, Bock et al. classify reasoners on the one hand based on the level of complexity and expressiveness they support and on the other hand based on their scalability [19]. In this respect they argue that classical description logic reasoners that implement tableau algorithms are able to classify large, expressive ontologies, but they often provide limited support for large number of instances. Conversely, database-like reasoners are able to handle large amounts of assertional facts, but are in principle limited in terms of the logic they support. In their benchmark, Bock et al. distinguish between load and response times to demonstrate strengths and weaknesses of the reasoners that follow these

paradigms. They start out with simple ontologies whose TBox (terminological knowledge) is relatively small. In assessing the scalability of the reasoner, they evaluate these measurements with respect to differently scaled ABoxes (assertional knowledge), but constant TBox. Their experiments have been performed on Sesame [18], OWLIM [23], KAON2 [33], HermiT [34], RacerPro [35] and finally Pellet [36].

It is stated in [19] that regarding classification, RacerPro outperforms other systems in terms of load and total execution times. If however load time is of minor importance, then HermiT with its novel hypertableau method performs best in classifying ontologies. Lightweight reasoning and storage systems such as Sesame and OWLIM are not advantageous over other reasoners in the expressivity fragments they are specifically tailored to process. Finally, resolution based systems such as KAON2 are not suitable for TBox reasoning tasks at all.

When it comes to answering conjunctive queries, Bock et al. argue that KAON2 is the best system with respect to the overall performance to load and respond, and that it shows a favorable scalability even with large ABoxes. Even though for expressive ontologies Pellet responds faster than KAON2, KAON2 is much faster in loading and more scalable than Pellet.

The reasoning tasks we will utilize in the hybrid search solution described in Chapter 5 will mostly be limited to those of TBox reasoning. On the other hand, we cannot really assess the complexity of the ontologies at this stage, because it heavily relies on the content repository model and the mappings defined (see Chapter 3). Therefore, to be able to plug-in the reasoner that best works with the extracted ontology, we will restrict ourselves to those that have support for the Description Logic Implementation Group (DIG) interface [37]. The DIG interface (often just known as DIG) provides a standardized XML interface to Description Logic Reasoners. The interface defines a simple protocol along with a concept language and accompanying operations.

2.5 Full-text and Structural Search

Full-text search is the prevailing paradigm employed by the content management system community. On the other hand, there are ongoing efforts to integrate full-text search with structural search [38], [6], [39], [40] and [41]. In this section, we present the popular technologies used

in both of these approaches; first in full-text search, and then in structural queries.

Apache Lucene [42] is an open-source full-text search engine library written entirely in Java. Yet, there are now a number of ports or integrations to other programming languages such as C/C++, C#, Ruby, Perl, Python, PHP, etc. Lucene is considered as a scalable Information Retrieval (IR) library. Information retrieval refers to the process of searching for documents, information within documents or metadata about documents [43]. By providing library functions, Lucene allows such search capabilities to be added to other applications. As long as text can be driven from it, Lucene does not care about the source of the data, its format, or even its language. The scoring mechanism in Lucene is based on the "term frequency-inverse document frequency (tf*idf)" vector product scheme [44]. "tf*idf" is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

Apache SOLR [45] is a full-text search server based on Lucene. SOLR provides a Representational State Transfer (REST)ful [46] Application Programming Interface (API) over Hypertext Transfer Protocol (HTTP) [47] - Extensible Markup Language (XML) [16]), and therefore, facilitates the development of faceted-search applications. Faceted-search is a search paradigm in which the user is given the option to navigate the search space by a progressive narrowing of choices in multi dimensions [48].

The SPARQL Protocol and RDF Query Language (SPARQL) [49] is the W3C candidate recommendation query language for RDF [7] and it grants the means to query the required and optional graph patterns along with their conjunctions and disjunctions. As explained in [50], SPARQL is essentially a graph-matching query language where the query consists of three parts: the pattern matching part, solution modifiers and the output. The pattern matching part supports union, nesting and filtering of possible matches, and the possibility to choose the data source to be matched by a pattern. The solution modifiers are operators like projection, distinct, order, limit, and offset. Finally, the output of a SPARQL query can be of different types: yes/no results, values that match the patterns, new triples from these values, and descriptions of resources. The full formal description along with a complexity analysis is provided in [50].

2.6 Persistence Issues in Content Management Systems

With the emerging need to organize data in hierarchies that evolve; there has been a shift towards the more flexible content repositories to manage content within a content management system. A content repository is a hierarchical content store with support for structured and unstructured content and various features such as full text search, versioning, transactions, observation, and more [51].

One such content repository is what is informally known as the Java content repository (JCR). As argued in [52], hierarchical content repositories are favored over traditional relational database management systems (RDBMS) due to their flexibility and support for unstructured content. Although a relational schema would be more advantageous for performing queries involving joins and update operations, it requires that the underlying structural associations among content items be known in advance. Unfortunately, this is not the case for most CMS applications where the content is inherently unstructured and is enriched as the application evolves. For a detailed comparison of the content repositories versus the relational model as the choice of the persistence layer in a content management system, please refer to [52].

2.6.1 Persistence Mechanisms in Content Management Systems - Trends

As more and more CMS vendors took up on the content repository model, various implementations have emerged and each vendor provided its own interface for interacting with the underlying content repository. On the one extreme, the abundance of ad-hoc content repository model implementations could pose some significant technical problems [53]:

1. Application developers need to work with a numerous number of ad-hoc interfaces,

2. Code portability is hindered,

3. Content becomes isolated in “information silos” where it is available only to the applications designed to access that specific content repository.

2.6.2 JSR-170

The Content Repository for Java technology API [54] (JSR-170, or more informally referred to as the JCR API) is a Java specification produced to overcome the aforementioned technical difficulties. As depicted on Figure 2.4, it provides access to content repositories in a standard way; independently of the underlying implementation [55].

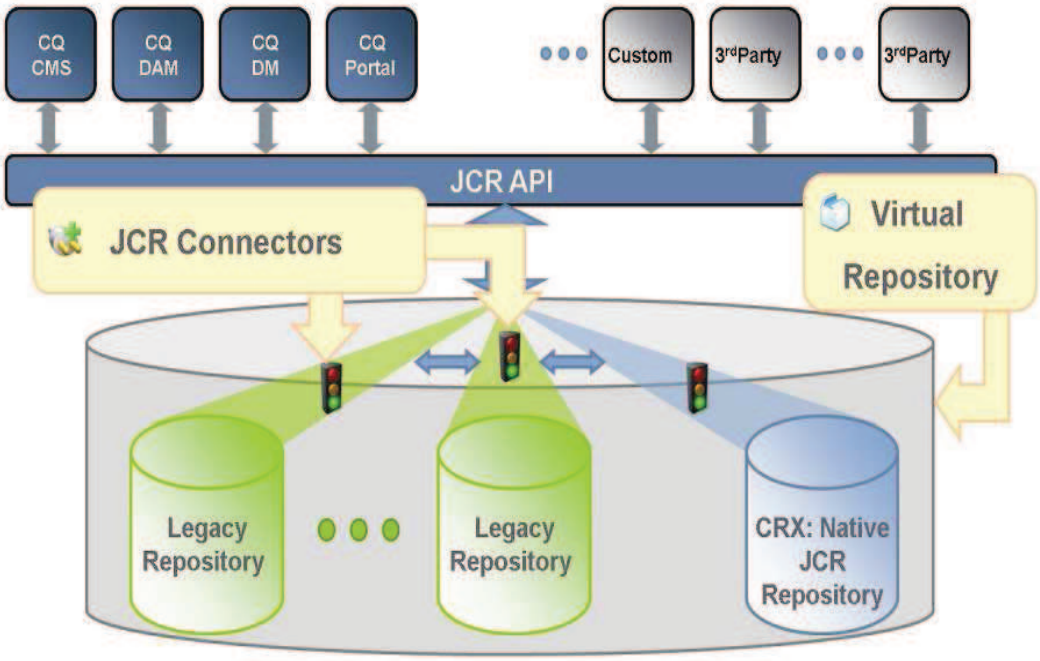

Figure 2.4: The JCR API provides a uniform interface over legacy content repositories [56]

JCR provides a functional view over the content repository. The content is organized in a tree structure: the leaves are called properties and the non-leaves are called nodes. A property is where the actual data or its associated metadata is stored. On the other hand; nodes, which may have other nodes or properties as its children, help application developers build the desired hierarchy over the content. Each tree has exactly one root; but the content repository may contain multiple trees called workspaces.

JCR enforces the XML Path Language (XPath) syntax [57] for its query language. Given the context, XPath is definitely a suitable choice; the tree structure in the JCR workspace is inherently analogous to that of an XML document. On the other hand, support for only a subset of the XPath language - a runtime Structured Query Language (SQL) [58] translatable

subset - is sufficient for JSR-170 compliance. As argued in [55], the rationale behind this decision is to ease database-backed implementations' integration. It is worth noting that as long as they meet the minimum requirements, repositories are free to support the full XPath syntax or additionally the SQL standard.

The principal idea behind JSR-170 is as follows: If products now provide a JSR-170 compliant view over their content repositories, then all applications may traverse the content in a uniform manner and thus content no longer becomes isolated in information silos. To ease the uptake of JSR-170, various compliance levels have been defined [59]. A content repository can be read-only or write compliant and may choose to implement some advanced features such as versioning, queries using SQL [58] or support for the Java Transaction API (JTA) [60]. Apache Jackrabbit [51] is a reference implementation of the specification.

The Content Repository for Java Technology API Version 2.0 (JSR-283) [61] is a work-in-progress towards the extension to JSR-170. Access control management, workspace and nodetype management and addition of new standardized node types are some of these extensions proposed by the authors of the specification.

2.6.3 Content Management Interoperability Services (CMIS) [62]

Even though, JSR-170 and JSR-283 provide a uniform means of accessing content repositories, these solutions are purely Java based, hence reproducing the vendor lock-in problem on a different scale. In this respect, an OASIS technical committee: "OASIS Content Management Interoperability Services (CMIS) TC" has been formed to enable information sharing across content management repositories using platform-neutral Web services and Web 2.0 interfaces [62]. The standard will define a domain model and set of bindings to work with one or more content management repositories/systems.

The technical committee does not aim at producing a specification addressing the full features a content management repository should implement. In fact, it follows the "least common denominator" approach; that is, the committee tries to extract functionalities common to all content repository implementations [63]. JCR on the other hand, fixes a specific model that inevitably would require some tailoring on the ad-hoc content repository to be fully-compliant.

As opposed to the nodes and properties, the domain model proposed by CMIS defines four

types of objects within a repository [63]:

1. Documents represent individual content objects in the repository.
2. Folders represent organizational containers in which documents (or other folders) can be stored. Folder objects are used to organize fileable objects.
3. Relationships represent loose relationships between exactly 2 objects (documents or folders) in the repository.
4. Policies represent administrative policies that may be applied to objects.

2.7 Remarks

In this chapter we have briefly overviewed the technologies relevant to our discussion in the upcoming chapters of the thesis. On the one hand, we have seen the different languages developed for the representation of knowledge. Furthermore, we have argued that mechanisms should exist for the storage, querying of and inferencing over the knowledge. In this respect, we have studied the Sesame and the Jena2 triple-stores. Along with these, we have also explored the various ontology engineering and reasoning frameworks. The last but not the least, we have seen some full-text and structural search mechanisms.

On the other hand, we have discussed the content management system paradigm and have explored the various content repositories used. We have seen two standards, namely; JSR-170 and CMIS, which have been (or are being) developed for defining interface functions to access different content repositories.

In Chapter 3, our discussion will proceed with the details of a framework whose role is to extract semantic information from content repositories. The extracted ontologies will be aligned or merged with additional domain or horizontal ontologies; hence enabling even further statements to be inferred through reasoning. Finally, in the following chapter; that is Chapter 4, the focus will be on the hybrid search mechanism developed that combines the power of full-text and structural search and utilizes the extracted ontology in the background.

CHAPTER 3

ONTOLOGY EXTRACTION FROM JSR-170 COMPLIANT CONTENT REPOSITORIES

3.1 Background

As presented in the earlier chapters; although some steps towards semantically enabled content management system solutions were taken, there is a lack of a holistic approach. For example, even though synonym matching and similarity search are some enhancements over text-based search; content management systems do not fully exploit the implied semantics in the content repository when utilizing these techniques. In synonym matching, the search set is expanded with synonymous keywords without taking into account the inherent semantic relations within the repository. On the other hand, when performing similarity search, resources with the same tags are simply assumed to be relevant where one should also exploit the structural relations among these tags.

To overcome these limitations, we propose to formulate - by means of an ontology extraction framework - the structural and semantic relations present in a content repository. The aim is threefold:

1. *Exploit the power of ontological formalisms that provide machine interpretable means to express and process semantic information*: Even though at an initial stage the extracted ontologies represent no further semantic information than what is already implied by the content model, the content model itself does not provide any means for the automated processing of the semantic relations. Consequently, there is a need for an ontological representation of the semantics in the content model.

2. *Establish a bridge between the content repository and the knowledge base*: The content repository is in constant growth. Not only the repository is populated with new content resources but also the repository model changes over time. These changes must be reflected on the knowledge base, that is; both the semantic relations and the ontology individuals should be updated.

3. *Provide a formal basis for alignment with extra domain knowledge*: In most cases it is desirable to enhance the extracted semantic information with additional domain knowledge. In fact, as will be discussed in the following sections, the real value of the ontology extraction framework comes from its promise that the extracted ontologies establish a formal basis for alignment with other domain or horizontal ontologies.

In our work, the focus is on the hierarchical rather than the relational content repository model due to its contemporary popularity among the content management system development community. Besides, techniques for extracting ontologies from relational schemas have already been proposed and exploited [64], [65], [66], [67] and [68], whereas no such work exists for the former case.

As JSR-170 or more informally known as the Java Content Repository (JCR) API enables uniform access to different content repository implementations, we will build our framework on JSR-170 rather than providing a separate implementation for each content repository. In this chapter, we will explore the possible techniques for extracting semantic information from JSR-170's hierarchical content structure; where the node type definitions, the super-subordinate relations present in the workspace hierarchy and the links among properties and individual nodes implicitly offer means to extract some valuable semantic information.

The chapter proceeds as follows: first, we briefly discuss what type of semantic information there is to extract from the JSR-170 content repository model. The implicit semantic information in the node type definitions and the workspace hierarchy altogether set up the motivation behind our ontology extraction framework: "JCR-to-Onto Bridge". Next, we discuss how the extracted ontologies could be aligned or merged with others to provide value-added services. In the proceeding section, we outline the various limitations of our approach and discuss how we deal with these challenges. Finally, we explain in detail JCR-to-Onto Bridge mapping cases with examples.

3.2 Motivation

3.2.1 Semantics Implicit in Node Type Definitions

In JSR-170, the node type definitions may be used to impose structural restrictions on content resources. Through defining a custom node type, one may control the corresponding types of each child node and/or property for a particular target node. The type of a child node may be restricted to one of the built-in or custom-defined node types. On the other hand, for properties, the user can choose from a predefined set of built-in types only: STRING, BINARY, DATE, LONG, DOUBLE, BOOLEAN, NAME, PATH, and REFERENCE. Providing value constraints or default values for a property is also possible. Finally, an inheritance hierarchy may be defined over custom node types where such relationship is expressed by the "supertype" property.

```
<cul = 'http://www.srdc.com.tr/iks/culture'>
[cul:ImagesType]
-cul:imageURL (STRING) *

[cul:CulturalHeritageItem]
-cul:location (STRING) m
-cul:popularity (STRING) < 'Very Popular', 'Popular', 'Not Popular'
+cul:images [cul:ImagesType]

[cul:Monument]>cul:CulturalHeritageItem
-cul:yearBuilt (DATE) m

[cul:AncientStructureAndBuilding]>cul:CulturalHeritageItem,mix:referenceable
-cul:type (STRING) m
```

Figure 3.1: Declaration of various custom node types

Figure 3.1 is a sample declaration of various node types in the Compact Namespace and Node Type Definition (CND) notation [69]. According to the example, a node of type "cul:CulturalHeritageItem" should have a child property named "cul:location" whose value must be a STRING . A value constraint is defined over the property "cul:popularity"; where, the only permitted values are the STRINGs: "Very Popular", "Popular" and "Not Popular". A "cul:CulturalHeritageItem" node may optionally contain a child node of type "cul:ImagesType" which acts as a bag for the associated image URLs. Finally, both "cul:Monument" and

“cul:AncientStructureAndBuilding” node types inherit their structure from the “cul:CulturalHeritageItem” node type and enhance it with additional attributes. A formal explanation of the syntax and grammar of CND is provided in [69].

From the node and property type declarations for a particular JCR repository alone, it is possible to construct classes, data type properties and in some cases, object property definitions. Later on, the workspace nodes, properties and their values will be used to instantiate the individuals for the generated ontology schemas. The procedure can be summarized as follows:

1. Each JCR node type definition corresponds to an ontology class.
2. JCR properties of type STRING, LONG, DOUBLE, BOOLEAN or DATE imply a data type property construction.
3. In ontological formalisms, the JCR properties of type PATH, REFERENCE or NAME can be expressed as object properties.
4. It is possible to represent the JCR node type inheritance hierarchy as a set of super class/sub-class relations among ontology classes.
5. When the content repository is populated with instances of the declared types and properties, so can the generated ontologies be.

As outlined above, one could possibly start by constructing ontology classes for each node type definition. In this case, it is intuitive to create a separate OWL class for each JCR node type defined. Although custom naming solutions may be followed, a default strategy could be to name these classes based on their native naming conventions (i.e. by using JCR namespaces and JCR node type names).

In most cases, property type definitions directly correspond to OWL data type properties. The simplest case is when the JCR property is of type STRING, LONG, DOUBLE, BOOLEAN or DATE. In our approach, we choose to define an OWL data type property whose domain is the class generated from the type definition of the parent JCR node and whose range points to one of the built-in OWL data types string, long, double, boolean and dateTime. A more complex case is when the JCR property has associated value constraints. In this case, it is possible to use the owl:oneOf construct to define an enumerated datatype [70].

For JCR properties, whose types are not one of STRING, LONG, DOUBLE, BOOLEAN nor DATE, a different method has to be applied:

- The underlying semantics for property definitions of type PATH, NAME or REFERENCE imply a stronger relation than what can be expressed by an OWL data type property. In fact, OWL object properties are more suitable for this purpose. A REFERENCE type is for storing the UUID of a node to which the property gives reference [55]. In case of a REFERENCE, referential integrity is maintained by the content repository. PATH properties serve a similar purpose by pointing to nodes in the workspace through path expressions. However, the repository does not enforce referential integrity. NAME is a specialized case for PATH, in which the path element lacks spatial locators. For these definitions, we choose to create an OWL object property whose domain is the class generated from the type definition of the parent JCR node and whose range is, in the least restrictive sense, an "nt:base" class. In JCR, nt:base is defined as the root node type from which all node types can be inherited and therefore, its OWL representation may serve as an abstract entity for the aforementioned range consideration.

- For value constraints defined over PATH properties, it is difficult to predict in advance whether or not the constraint has any implication on the range of the generated OWL object property since they are simply regular expressions defined over some path values. However, the value constraint for the REFERENCE property is interpreted as a node type name. In fact, it restricts the types of nodes to which the REFERENCE property may refer. In such a case, our "nt:base" assumption for the generated OWL object property can be restricted to account for the selection.

- The BINARY type implies that the particular JCR property is for storing a document rather than its metadata. Therefore, there will not be any added semantic value in representing this relation as a data typetype property nor an object property. However, a link to the actual content resource can be created, whose value will be clearer in Chapter 4.

When it comes to the representation of the JCR node type inheritance hierarchy, the most intuitive approach would be to use superclass-subclass relations in OWL. It is particularly valid for custom-defined as well as for built-in node types (Figure 3.2). Therefore, for each supertype defined, a corresponding superclass relation should be generated. In this respect,

multiple supertypes pose no particular problem since OWL allows multiple superclasses to be defined for a single class.

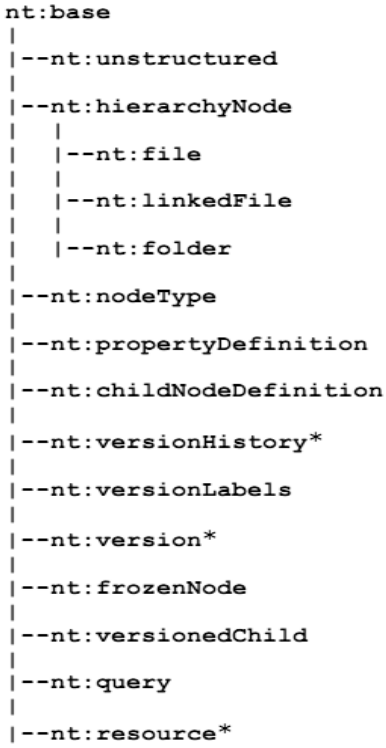

Figure 3.2: Inheritance hierarchy among built-in node types [55]

In Appendix A, some of the ontology constructs used to express the underlying semantic relations in the node type definition example provided in Figure 3.1 are depicted. The “cul:popularity” property on which some value restrictions were provided in the node type definition is expressed as an enumerated data type property. The node type inheritance is reflected on the ontology with the proper use of the “rdfs:subClassOf” predicate. Finally, the object property “hasImages” has been created for associating the class generated from the “cul:CulturalHeritageItem” node type definition with the class generated from the node type definition of “cul:ImagesType”. The intuition behind this will be clearer in Section 3.2.2.

3.2.2 Semantics Implicit in the Workspace

There are cases where the node type definitions are insufficient to extract all the relationships among content resources. In fact, node types help restrict the structure of a content resource; however, may fail in fully expressing the hierarchical implications embedded in the overall tree structure. As an example, consider the following JCR workspace configuration presented on Figure 3.3.

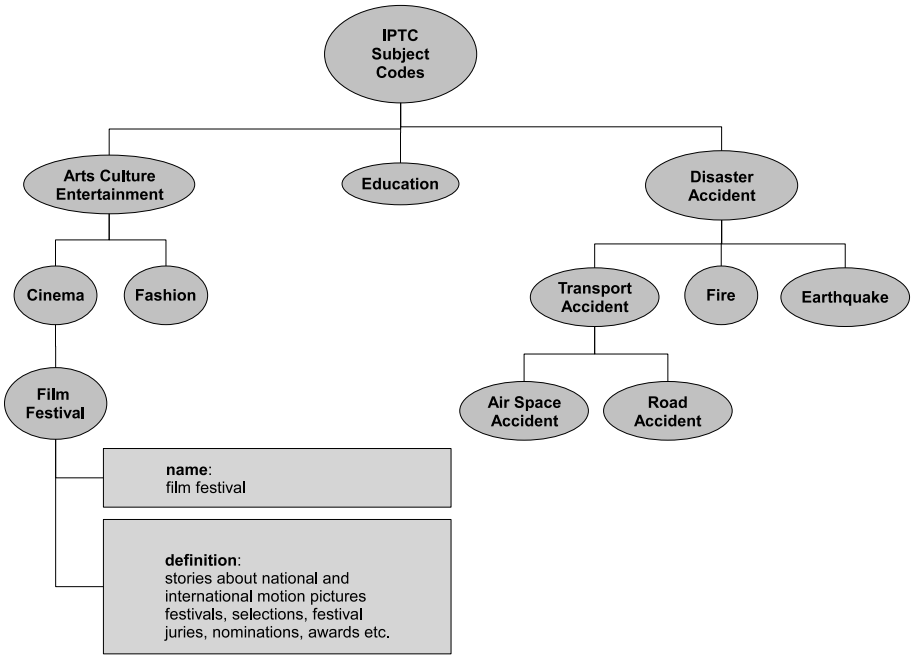

Figure 3.3: JCR workspace configuration depicting the IPTC News Subject Codes

The tree rooted at “IPTC Subject Codes” displays a subset of the hierarchy defined in the “Subject Code” vocabulary [71] of the International Press Telecommunications Council (IPTC) [72], News Codes taxonomy [73]. Subject Code is a three level system for describing content by a well defined set of terms. Topics of level Subject provide a description of the editorial content of a News at a high level, a SubjectMatter provides a description at a more precise level and finally a SubjectDetail at a rather specific level [74].

In Figure 3.3, each JCR node corresponds to a particular concept in the IPTC News Codes - Subject Code vocabulary. Furthermore, the JCR nodes are described by the two JCR proper-

ties, namely; “name” and “definition” [1]. The properties are declared to be STRINGs. In this particular example, the node and property type definitions go as far as defining the allowed set of the values (i.e. STRING) and how a Subject Code is described (i.e. a Subject Code node should have two properties: “name” and “definition”). However, what is more valuable here is that the subject codes are organized in a subsumption hierarchy, that is; “Cinema” is a broader subject than “Film Festival”; where “Arts, Culture and Entertainment” is broader than both. Such semantic relationships are implicit in the workspace hierarchy rather than the node type definitions.

Semantic relations in the form of subsumption are not the only candidates for what is implicit in the workspace hierarchy. In fact, one could possibly construct “part-whole” or even other relationships for different tree arrangements. A collection of book chapters; where each chapter consists of an introduction, several body sections and a conclusion is a typical use case for the former case. The scenario simply suggests a construction based on a “part-whole” relationship rather than subsumption.

At this point, one could argue that it is actually the node type definitions that account for such structural deductions. However, even when the nodes are not strictly typed, that is; say they are left as “nt:unstructured”, we should be able to recognize these structural patterns and so their implications. As depicted on Figure 3.4, rather than the node types, same name siblings and repeating sub-trees account for the semantics in this particular configuration.

3.2.3 Implications

Up to now we have seen how node type definitions along with the workspace hierarchy can be helpful in extracting ontology classes, properties and individuals. There are some trivial cases; such as the JCR properties of type STRING, LONG, DOUBLE, BOOLEAN or DATE, where simply some data type properties are constructed. The case with REFERENCE-, PATH- or NAME- type JCR properties is more complex; where object properties may be more suitable. Node type definitions or a particular set of JCR nodes (see Figure 3.3) may be valuable in extracting class definitions. In the former case, the inheritance is explicitly expressed by the hierarchy within the node types. On the other hand, in the latter case, one has to look at the parent-child relationships in the JCR nodes. In all cases, same name siblings and repetitions

[1] For space considerations the properties for only the “Film Festival” JCR node are displayed.

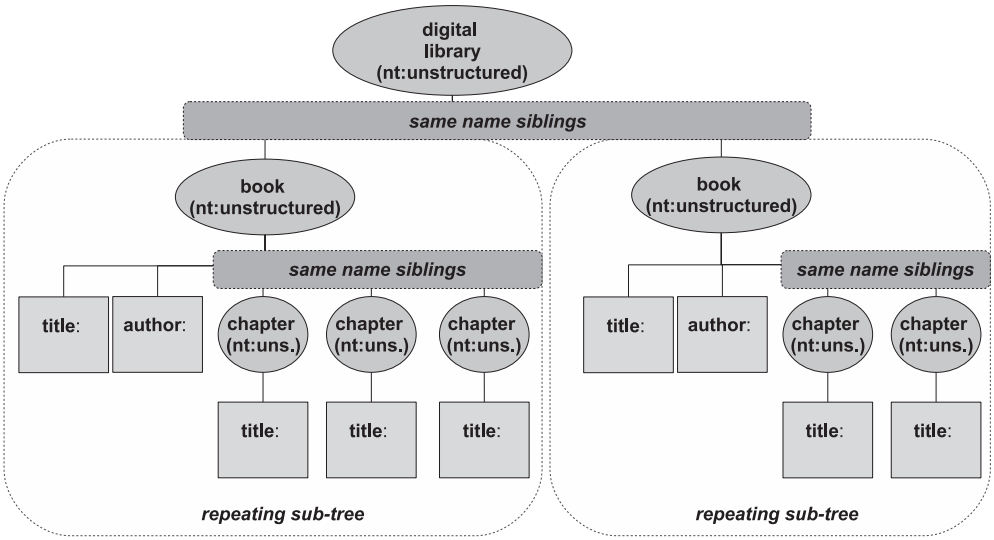

Figure 3.4: Example workspace configuration with “part-whole” implications

can be used to formulate other semantic relationships.

As demonstrated in the aforementioned examples, even a small-scale JCR application is built with different types of semantic associations hard-coded in its content repository’s node type definitions, taxonomical structures and nodes that share common patterns. If the content management system applications are given the means to exploit this semantic information, then value-added services such as automatic item categorization, finding related content and different forms of semantic search can be built. The content repository alone does not facilitate the development of these services; as it does not provide a formal representation to interpret and reason over the semantics implicit in the repository. Consequently, applications have to deal with this problem internally, and for most of the time by impromptu solutions. In this respect, our motivation is to devise an ontology extraction framework - namely, the JCR-to-Onto Bridge framework - so as to formally represent the otherwise implicit semantic information and later on enhance it with additional horizontal or domain knowledge. The benefits of this approach can be summarized as follows:

1. Exploit the power of ontological formalisms that provide machine interpretable means to express and process semantic information

2. Establish a bridge between the content repository and the knowledge base

3. Provide a formal basis for alignment with extra domain knowledge

3.3 Basis for Alignment with External Horizontal and Domain Ontologies

Before going into the details of JCR-to-Onto Bridge and the mapping schemes, it is worth illustrating that the ontology extraction framework sets up a basis for alignment with already available horizontal and domain ontologies. The value-added features enabled by JCR-to-Onto Bridge's initial representation of the semantics in the content repository can further be improved if the extracted ontologies are augmented with additional semantic data. Techniques for semi-automatic merging and alignment of ontologies exist in literature [75], and therefore will not be repeated in this section. On the other hand, our focus is to demonstrate the added-value of integration of such techniques and what JCR-to-Onto Bridge offers to facilitate this integration.

Consider the following JCR tree representation on Figure 3.5 of the association between certain concepts and tags in one of Flickr [76]'s (a photo sharing site) tag-clouds. As described by [77], a tag-cloud is a list of the most popular tags, usually displayed in alphabetical order, and visually weighted by font size. In a tag-cloud, when a user clicks on a tag, (s)he obtains an ordered list of tag-described resources, as well as a list of other related tags. In the example, the individual tags are organized in clusters where the clusters form a hierarchy of concepts. For example, "streetart", "wall", "stencil", "paint" etc. are tags associated with the cluster "graffiti", which in turn forms a subset of the more generic cluster "art". Furthermore, we assume that resources available in the other nodes of the workspace that are not explicitly shown on Figure 3.5 are associated with these tags.

By (i) lifting each JCR node in the example shown on Figure 3.5 to an ontology class, (ii) establishing the inheritance hierarchy among the classes and finally (iii) instantiating the JCR properties (that represent particular tags) as individuals of the generated classes, it is possible to formally relate resources tagged with items that belong to say, the same cluster, or speaking in ontological terms, that are individuals of the same class. However, for some applications this still turns out to be insufficient and a need to enrich these semantic relations with additional knowledge may be necessary. For this particular example, the WordNet ontology [78] is a possible candidate. As depicted on Figure 3.5, when aligned with the WordNet ontology,

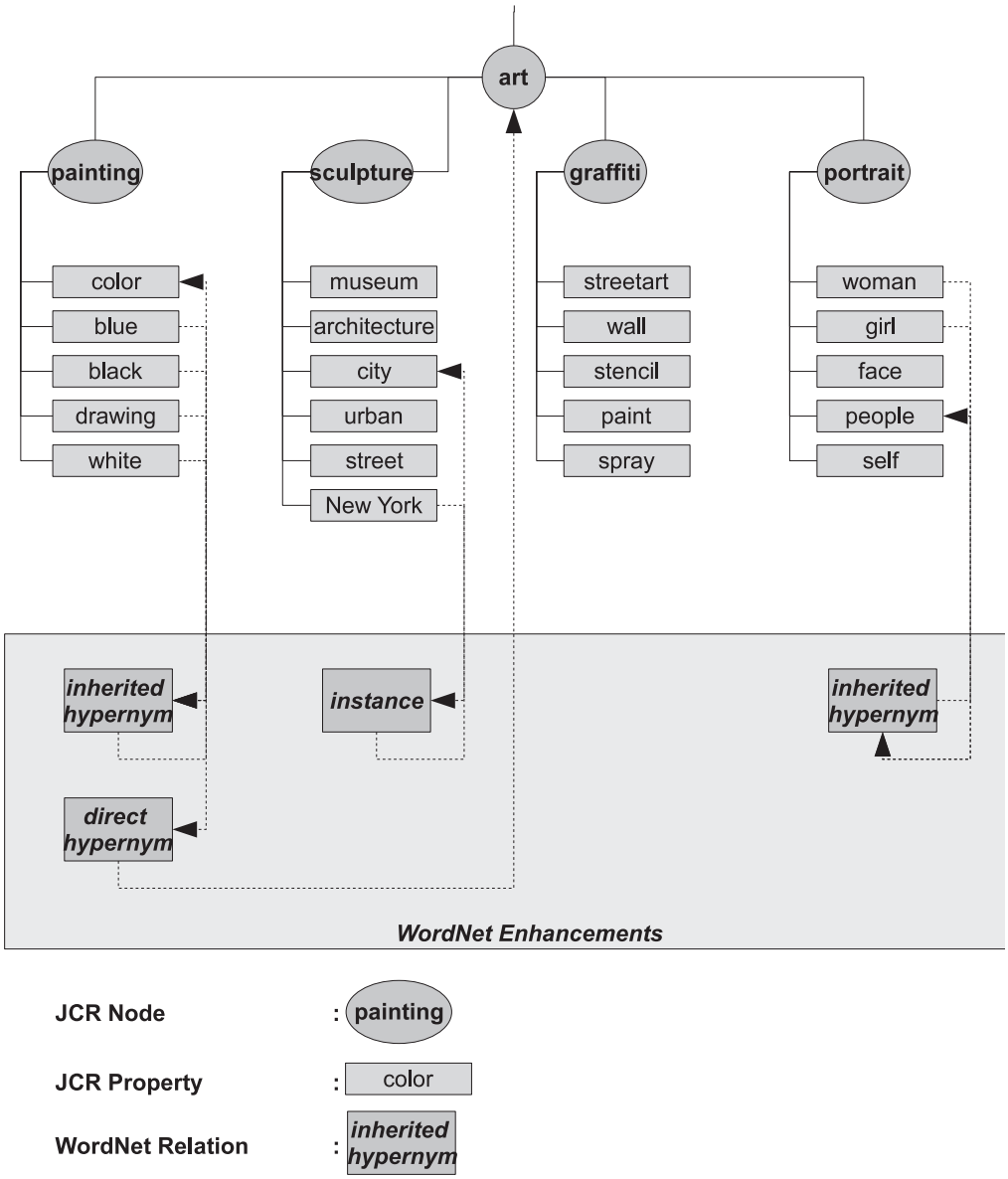

Figure 3.5: Association between the terms in the tag cloud

the tag “color” becomes the hypernym [2] of the tags “blue”, “black” and “white”. Similary, the tag “people” - apart from belonging to the same tag cluster - is now the hypernym of the tags “girl” and “woman”. The tag “New York” becomes an instance of the tag “city” and finally, the tag cluster “art” is inferred as the hypernym of the tag “drawing”. Now, an application that uses this information in search may provide a faceted browsing of content or likewise a more

[2] In linguistics, a hypernym is a word or phrase whose semantic range includes that of another word, its hyponym.

precise ranking of search results based on similarity. In the former case, all resources tagged with any of the items in the “painting” cluster would be equivalently related whereas after the alignment process, resources referring to the hyponyms of the tag “color” (i.e. “blue”, “black”, “white”, etc.) would be ranked more relevant.

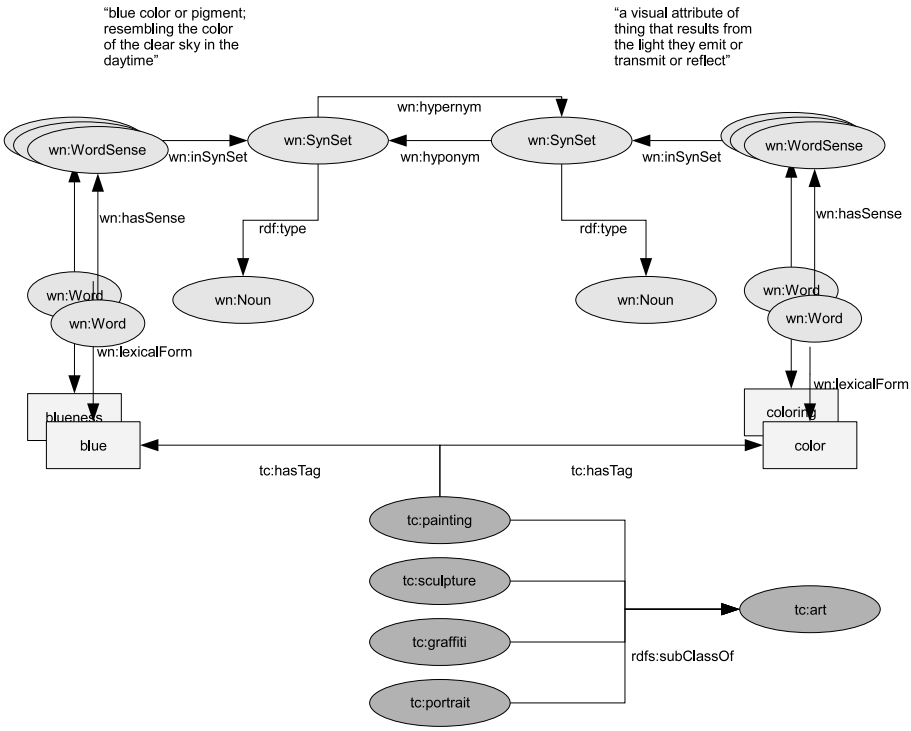

Figure 3.6: Technical illustration of how tags can be annotated with the WordNet ontology

3.4 Exploitation of Modeling Patterns

The problem with the aforementioned approach is that: i) node type definitions are not fully exploited by the user community, ii) the flexibility provided by JCR makes it difficult to predict in advance what type of relations are to be extracted from the workspace. On the other hand, the degree of flexibility that the content repository model provides is limited in practice by the level of freedom the applications built on top of the repository can support at run-time. That is; if every node is defined to be “nt:unstructured”, then the applications should have the capability to resolve the type information of a particular node from either its content or

structure at run-time. Consequently, even if not explicitly encoded in the workspace node type definitions, some structural patterns, that both the working applications and our ontology extraction components could exploit, do exist.

Our aim with the JCR-to-Onto Bridge approach is, therefore, to design a configurable ontology extraction framework that makes use of the structural patterns present in the workspace. The configuration should be flexible enough to handle different workspace arrangements and hence let the users identify the semantic relations as freely as possible. Based on our discussion in sections 3.2 and 3.3, these semantic relations will account for the construction of:

- ontology classes
- relations among ontology classes (e.g. subsumption)
- object and data type properties
- individuals

When the node type definitions fail in fully describing the semantic structure of the workspace, it is just a matter of identifying the workspace nodes or properties that can be used in either of these constructions. Put in other words, we are trying to develop a scheme by which the user maps the appropriate set of nodes and properties in the workspace to different ontology construction processes. To facilitate this process, we have developed a Graphical User Interface (GUI) on top of the JCR-to-Onto Bridge framework. The GUI lets the user browse the whole JCR repository, select a set of nodes and properties and finally associate with each set of selected node or property, a predefined ontology construction process. Slowly, the user builds a group of mapping definitions that are used in the ontology extraction process. In this regard, our discussion in this section will mainly revolve around the following:

1. the different possible workspace arrangements,
2. the native JCR queries in selecting the appropriate set of workspace nodes and properties,
3. the various mapping templates (i.e. ConceptBridge, SubsumptionBridge, PropertyBridge, EnforcedPropertyBridge and InstanceBridge) corresponding to the different constructions outlined earlier.

JSR-170 specification enforces the XPath syntax [57] for its query language. Given the context, XPath is definitely a suitable choice; the tree structure in the JCR workspace is inherently analogous to that of an XML document. On the other hand, support for only a subset of the XPath language - a runtime SQL translatable subset - is sufficient for JSR-170 compliance. As argued in [55], the rationale behind this decision is to ease database-backed implementations' integration. It is worth noting that as long as they meet the minimum requirements, repositories are free to support the full XPath syntax or additionally the SQL standard. Therefore, in this thesis, we will not make a distinction between the SQL translatable subset of XPath and the SQL itself, and illustrate our examples in SQL. A detailed mapping between the two is provided in [55].

A native JCR query, whether XPath or SQL, specifies a set of nodes in the workspace that satisfy any of the following constraints [55]:

- Type constraint: Specifies the primary node type of the returned nodes
- Property constraint: Limits the returned nodes to those having particular properties having particular values
- Path constraint: Restricts the returned nodes to a subtree in the workspace described by a path expression

The above-described constraints are sufficient for the purposes of our mapping scheme. Now, consider the following cases:

3.4.1 Mapping Cases for the Construction of Ontology Classes

A class description that is not encoded in the node type definitions may exist in the workspace in either of the two patterns:

1. Hierarchical Categorization: a classification hierarchy in the workspace where the nodes, or a subset of the nodes satisfying certain criteria, correspond to ontology classes (e.g. Figure 3.3).
2. Flat Categorization: a tree in the workspace where the direct children of the root, or children satisfying certain criteria, correspond to ontology classes (e.g. Figure 3.4).

In the former case, an example query selecting the nodes to be used in a class construction could be:

SELECT * *from nt:base* **WHERE** *jcr:path* **LIKE** *'/%/IPTC_Subject_Codes/%'*

XPath Query*: /jcr:root//IPTC_Subject_Codes//element(*, nt:base)*

The query would then select all nodes that are either direct or indirect descendents of the node named "IPTC_Subject_Codes" and would trigger a class construction for each. The regular expression "%" in the path constraint "//IPTC_Subject _Codes/%" accounts for this selection. The type constraint, that is; "from nt:base" indicates that the selected nodes could be of any JCR node type[3].

The query works suitably well for the example provided in Figure 3.3; however, as one would agree it is not the only arrangement possible. Consider now the following hierarchical categorization displayed on Figure 3.7 for which the query needs to be slightly modified:

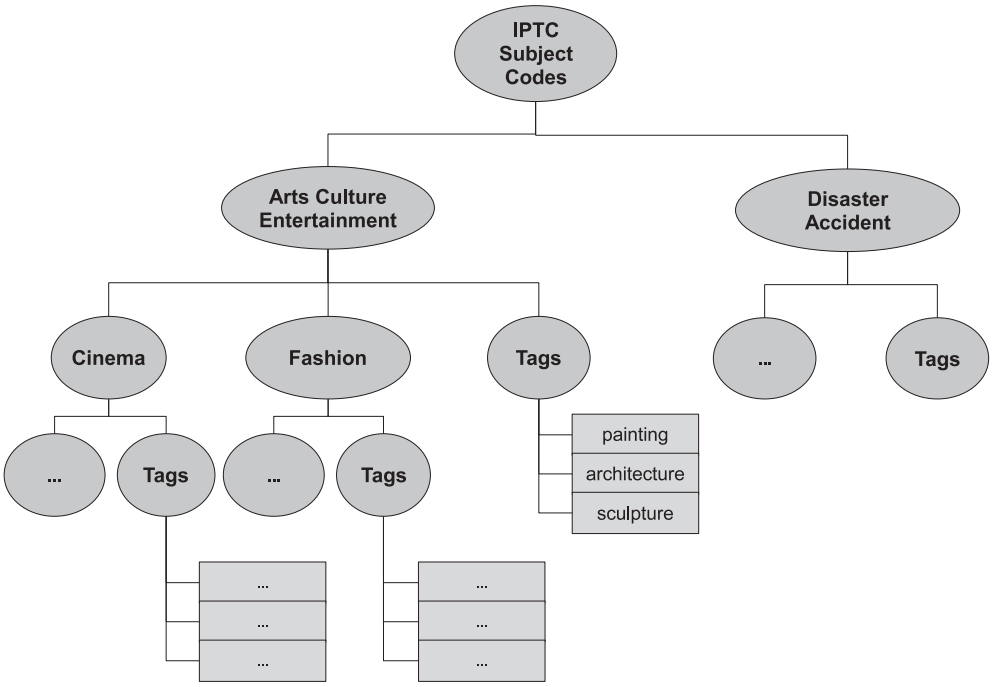

Figure 3.7: Classes described in a hierarchical categorization pattern

[3] In JSR-170, "nt:base" is defined as the node type from which all built-in and custom node types are derived.

The only difference between Figure 3.3 and Figure 3.7 is that in the latter case some nodes do not communicate ontology class construction. For each subject code, a node named “Tags” has been added to store the list of associated tags with that particular subject. Otherwise, the hierarchy is nothing but a categorization of the IPTC News Subject Codes.

In this case, the query selecting only the nodes that exhibit the class role would be:

SELECT * *from nt:base* **WHERE** *jcr:path* **LIKE** *‘/%/IPTC_Subject_Codes/%’*

AND NOT *jcr:path LIKE ‘/%/IPTC_Subject_Codes/%/Tags’*

Up to now, we have seen some examples of hierarchical categorization. On the other hand, a typical example for flat categorization is the case that of a root node having children with different names, where each set of same-name-siblings is used to construct a different ontology class. Obviously, in such an example, the structure of the ontology class would be derived from the structure of the nodes in the same-name-sibling set, given that the structure remains constant throughout.

A possible query to be used in the construction of an ontology class corresponding to the “chapter” entity in Figure 3.4 could be:

SELECT * *from nt:base*

WHERE *jcr:path* **LIKE** *‘/%/digital_library/book[%]/chapter[%]’*

In JCR-to-Onto Bridge, the mapping definition that corresponds to a class construction both for the hierarchical and the flat categorization case is the “ConceptBridge” shown on Figure 3.8.

The “Query” component conveys the native JCR query that selects the node to be formalized as an ontology class. Unless a value is provided for the “ConceptName” element, the class is named after the JCR node from which it is extracted. The class may be declared as the subclass of another; however, in that case, a “SubsumptionBridge” has to be declared. Finally, object and data type properties, whose domains become the class in question, can be generated on-the-fly with an appropriate “PropertyBridge” declaration. The details of both “SubsumptionBridge” and “PropertyBridge” will be presented in the upcoming sections.

```
<xsd:element name="ConceptBridge">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Query"
                type="tns:QueryType" />
            <xsd:element name="ConceptName"
                type="tns:NameType" minOccurs="0" />
            <xsd:element ref="tns:SubsumptionBridge"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="tns:PropertyBridge"
                minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3.8: The XSD for "ConceptBridge"

3.4.2 Mapping Cases for Establishing Relations among Ontology Classes

Either during the ontology class construction process, or independently, a class may be asserted as the subclass of another. For example, for the hierarchical categorization case shown on Figure 3.3, the class formalized from a node should be asserted as the subclass of what is formalized by its parent node. In other words, based on Figure 3.3, both Cinema and Fashion are subclasses of the ontology class Arts_Culture_Entertainment and similarly, Film_Festival is a subclass of the generated Cinema class.

Suppose we would like the ontology extraction framework to assert these subclass relationships as each super class is generated. However, this poses a problem. Let us recall the query used in selecting these particular nodes that are meant to be lifted up to the ontology classes:

Outer Query*:*

SELECT * *from nt:base* **WHERE** *jcr:path* **LIKE** *'/%/IPTC_Subject_Codes/%'*

XPath Query*: /jcr:root//IPTC_Subject_Codes//element(*, nt:base)*

What is desired is a way to express for each selected node above, another node in the workspace that represents its subclass (or subclasses). For the particular example, it would be the chil-

dren of the selected node. By writing two independent queries - one for the selection of the super class nodes and one for the selection of the subclasses - it is not possible to achieve the desired behavior. In fact, one needs to write the second query relative to the first one, where the path expression in the second query should be executed relative to each node selected by the first. Unfortunately, JCR does not natively support such behavior. Consequently, the notion of a query has been extended; in fact, $SELECTION_PATH is a reserved keyword in JCR-to-Onto Bridge that lets the path expression in the outer query be used in that of the inner.

Inner Query*:*

SELECT * *from nt:base* **WHERE** *jcr:path* **LIKE** *'$SELECTION_PATH/%'*

AND NOT *jcr:path* **LIKE** *'$SELECTION_PATH/%/%'*

In this example, the outer query selects all descendents of the node IPTC_Subject_Codes whereas the inner query is executed for each node selected by the outer query and it selects the direct children of the descendent. If used in an ontology class construction process, the implications would be to create an ontology class for each descendent of IPTC_Subject_Codes and assert a subclass relationship for each direct children of the descendent node. In JCR-to-Onto Bridge, the mapping definition that is used for asserting such subsumption relations between ontology classes is the "SubsumptionBridge" element whose schema is provided on Figure 3.9.

```
<xsd:element name="SubsumptionBridge">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="PredicateName"
                type="tns:QueryType" />
            <xsd:element name="SubjectQuery"
                type="tns:QueryType" minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3.9: The XSD for "SubsumptionBridge"

The "SubjectQuery" element conveys the "Outer Query" described earlier in this section. It

selects the node from which the super class is extracted. Conversely, the “PredicateName” element is built around the “Inner Query”. It selects the nodes from which the sub classes are extracted. It is worth noting that the use of the “SubjectQuery” element is optional. In case the “SubsumptionBridge” is declared within a “ConceptBridge”, the query to select the super class is already available; hence, it should not be repeated within the “SubsumptionBridge”.

3.4.3 Mapping Cases for the Construction of Object and Data Type Properties

The following patterns in the workspace account for the construction of object or data type properties:

- Case 1: A node in the repository that is selected for ontology class construction (e.g. ConceptA) references another node in the repository through a JCR property of type REFERENCE, PATH or NAME, where the referenced node is also selected for ontology class construction (e.g. ConceptB). The pattern is depicted on Figure 3.10. It is worth noting that the JCR property may or may not be the direct descendent of the node in question; but yet a relative path exists to the property.

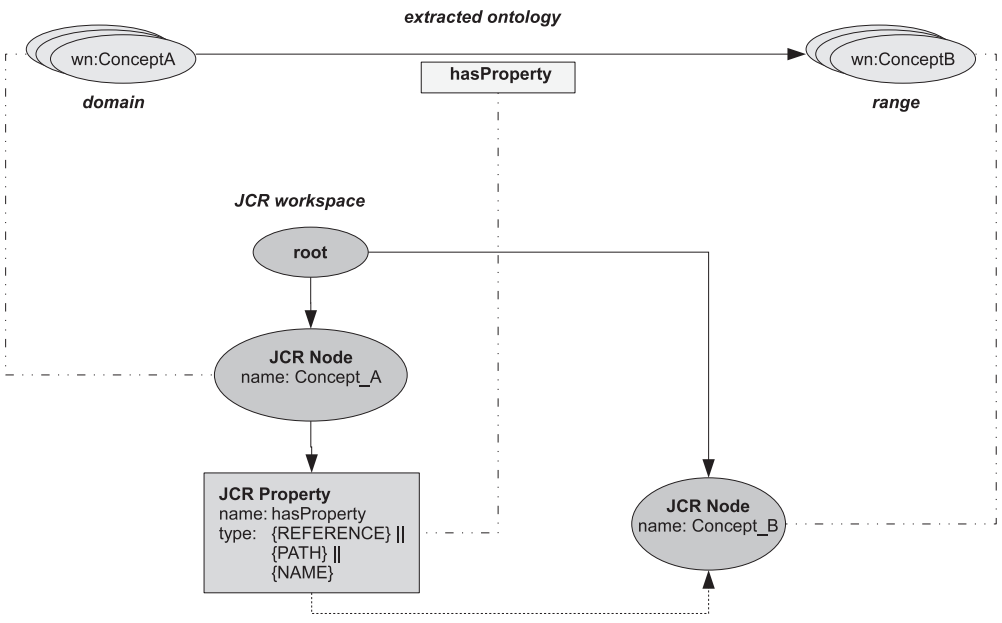

Figure 3.10: Workspace configuration for Mapping Case - 1

- Case 2: A node in the repository that is selected for ontology class construction (e.g. ConceptA) has a JCR property of type BINARY, BOOLEAN, DATE, DOUBLE, LONG or STRING. The pattern is depicted on Figure 3.11. It is worth noting that the JCR property may or may not be the direct descendent of the node in question; but yet a relative path exists to the property.

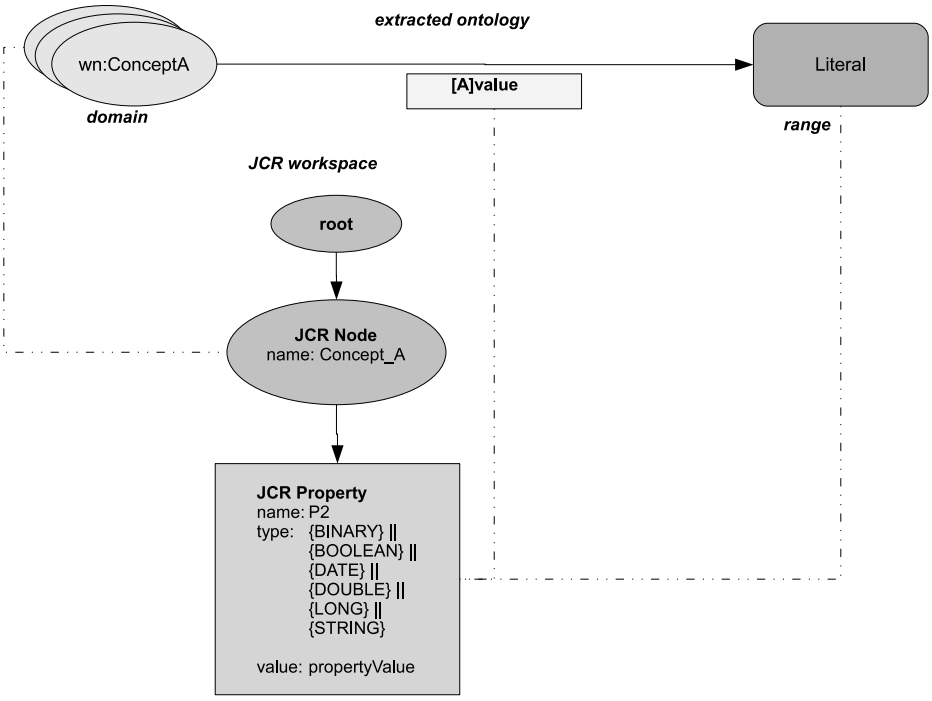

Figure 3.11: Workspace configuration for Mapping Case - 2

- Case 3: A node references two other nodes in the repository through JCR properties of type REFERENCE, PATH or NAME, where the referenced nodes are selected for ontology class construction (e.g. ConceptA and ConceptB, respectively). The pattern is depicted on Figure 3.12. It is worth noting that the JCR properties may or may not be the direct descendents of the node in question; but yet a separate relative path to each exists.

- Case 4: A node in the repository has two JCR properties where one is a property of type REFERENCE, PATH or NAME and the other BINARY, BOOLEAN, DATE, DOUBLE, LONG or STRING. The node references a second node in the repository through

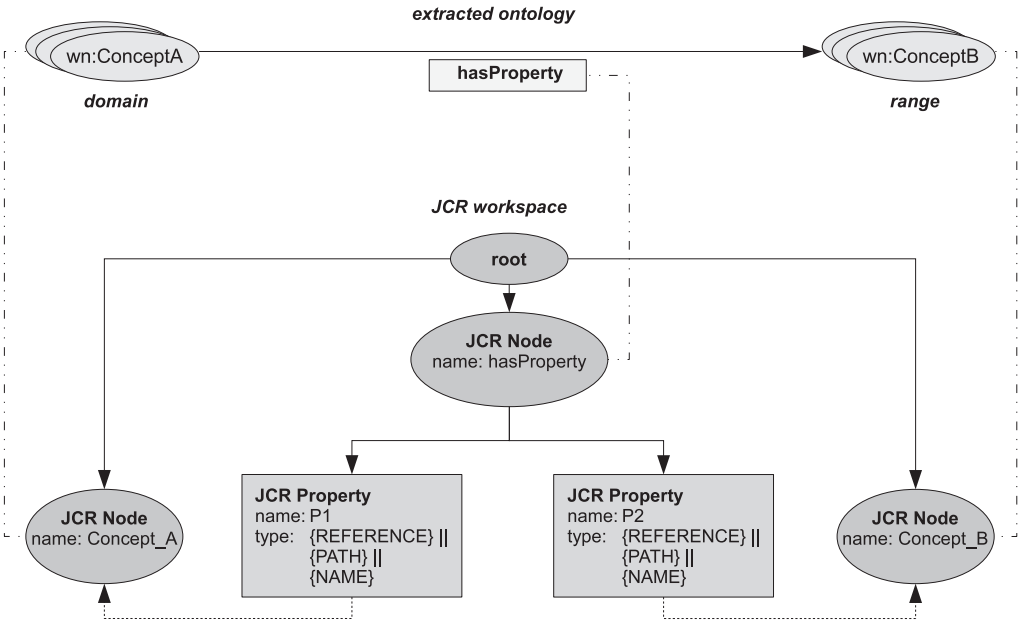

Figure 3.12: Workspace configuration for Mapping Case - 3

the formerly described JCR property of type REFERENCE, PATH or NAME, and the referenced node is selected for ontology class construction (e.g. ConceptA). The pattern is depicted on Figure 3.13. It is worth noting that the JCR properties may or may not be the direct descendents of the node in question; but yet a separate relative path to each exists.

Case 1 and Case 3 refer to patterns where the straightforward construction would be that of an object property. The domain and the range of the object property are explicitly the classes described by the JCR nodes in context. On the other hand, Case 2 and Case 4 are more suitable for the construction of data type properties: The class described by the JCR node in context designates the domain of the data type property and the type attribute of the JCR property determines the built-in ontology type to be used for the property range.

Another observation to make is that for Case 1 and Case 2, two native queries are sufficient to locate the nodes and properties accounting for the property construction. The first query selects the node (or set of nodes) to be used in setting the domain of the property. The second query depends on a relative path expression to select the JCR property to be used as the basis

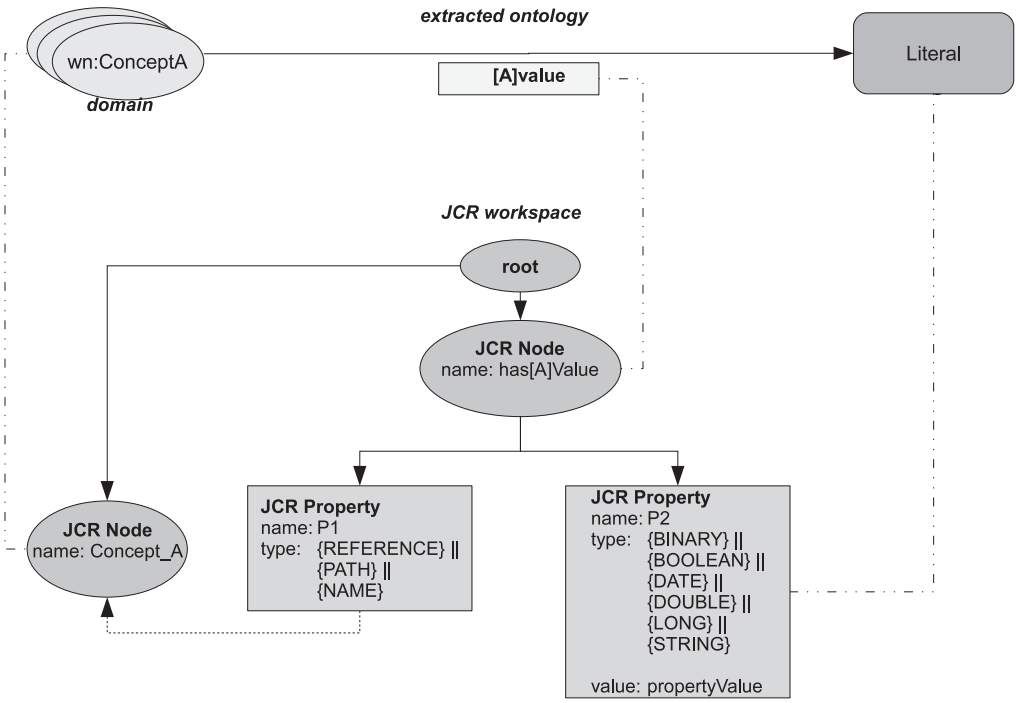

Figure 3.13: Workspace configuration for Mapping Case - 4

for the property construction process; such that the name and the type of the to-be-constructed ontology property are derived from the name and type of the JCR property. If the JCR property is of type REFERENCE, PATH or NAME then an object property is generated whose range is determined by the referenced JCR node. On the other hand, if the JCR property is of type BINARY, BOOLEAN, DATE, DOUBLE, LONG or STRING then a data type property with an appropriate built-in type for its range is constructed.

The conditions are somewhat different for Case 3 and Case 4. In fact, a separate JCR node is used as the basis for the property construction process; accounting for the need to have an additional query. The first or the base query selects exactly these nodes. The second and third queries depend on some relative path expressions to select the two JCR properties to be used for determining the domain and range of the property, respectively. If the JCR property selected by the third query (i.e. query for selecting the range) is of type REFERENCE, PATH or NAME then an object property is generated. The complementary case, that is; when the JCR property is of type BINARY, BOOLEAN, DATE, DOUBLE, LONG or STRING, implies a data type property generation.

Consider the example displayed on Figure 3.14 where the news articles are categorized by their subject codes. Each JCR node with the name “NewsArticle” contains a JCR property “hasSubject” that is of type PATH. The JCR property references a subject code from the tree of subject code categorizations. As one would notice, this is an example of the pattern described in Case 1. What is desired is a way to create the object property named “hasSubject” with the NewsArticle class as its domain and a union of each selected subject code as its range. Based on this information, the JCR queries to select the nodes to be used in the object property construction are as follows:

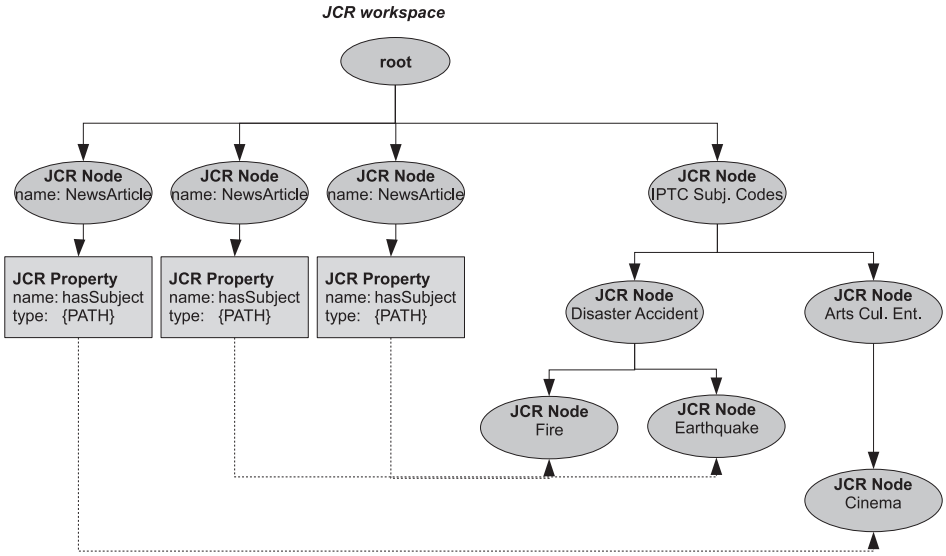

Figure 3.14: A sample workspace configuration for property mappings

Property Domain*:*

SELECT * *from nt:base* **WHERE** *jcr:path LIKE ‘/%/NewsArticle’*

Predicate basis*:*

SELECT *my:hasSubject* **FROM** *nt:base*

WHERE *jcr:path* **LIKE** *‘$SELECTION_PATH’*

The query for locating the property domain selects all nodes named “NewsArticle”, and the inner query is executed for each selected NewsArticle node to obtain its child property named “hasSubject”.

Figure 3.15 provides a representation of the generated object property “hasSubject”. One important point to mention is that the range of the property is set as the union of the classes “Fire”, “Earthquake” and “Cinema”: the aforementioned queries select multiple nodes from the workspace, each of which accounting for the construction of the same object property. Consequently, whenever a new class is encountered for the range of the property, the class is simply added to the union of possible classes in its range.

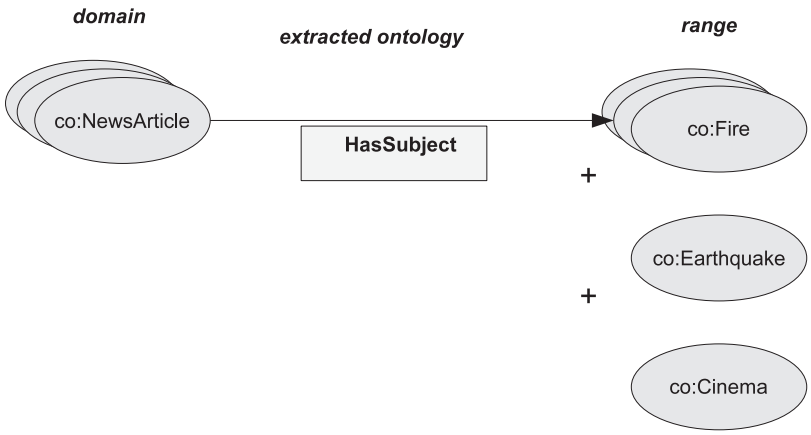

Figure 3.15: The generated “hasSubject” object property

In JCR-to-Onto Bridge, the mapping definition that corresponds to the object (or data type) property construction for the first and second cases is depicted on Figure 3.16.

The two queries “SubjectQuery” and “PredicateQuery” locate (i) the node that is used in setting the domain of the property and (ii) the JCR property from which the name, type and range of the ontology property is extracted, respectively. The “Transformation” element can be used for a possible future extension for modifying the value of the property. Transformations are currently not handled by the framework. Finally, “PropertyAnnotation”s convey additional information about the property to-be-constructed; such as its transitivity, functionality, etc.

Likewise, the mapping definition that corresponds to the object (or data type) property construction for the third and fourth cases is depicted on Figure 3.17. The “PredicateQuery” element points to the node from which the name of the property is obtained. On the other hand, “SubjectQuery” and “ObjectQuery” select the two JCR properties that determine the domain and range of the ontology property, respectively.

```
<xsd:element name="PropertyBridge">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="PredicateQuery"
                type="tns:QueryType" />
            <xsd:element name= "SubjectQuery"
                type="tns:QueryType" minOccurs="0" />
            <xsd:element ref="tns:Transformation"
                minOccurs="0" />
            <xsd:element ref="tns:PropertyAnnotation"
                minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3.16: The XSD for "PropertyBridge"

```
<xsd:element name="EnforcedPropertyBridge">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="PredicateQuery" type="tns:QueryType" />
            <xsd:element name="SubjectQuery" type="tns:QueryType" />
            <xsd:element name="ObjectQuery" type="tns:QueryType" />
            <xsd:element ref="tns:PropertyAnnotation" minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3.17: The XSD for "EnforcedPropertyBridge"

3.4.4 Mapping Cases for Instantiating Individuals

Prior to creating the individuals in the extracted ontology, the following information has to be available in the content repository:

1. Names of the individuals to be created
2. Ontology class(es) for the individual

3. Optionally, object or data type properties associated with the individual and their values

Names of the individuals to be created:

The content nodes in the repository can be used as the basis for all computations related to the construction of an individual in the JCR-to-Onto Bridge framework. As illustrated on Figure 3.18, the node named “B737-800” or equivalently, “C-Skyhawk 172” is a container node for a particular resource, in this case; information about an aircraft model. Such container nodes will be used to derive the name of each individual to be created, along with other useful information.

Ontology classes for the individual:

The motivation behind extracting ontology constructs from node type definitions has already been discussed in Section 3.2.1. Extracting ontology classes from the node type definitions becomes particularly valuable when it comes to assigning classes to an individual. Technically, every JCR node - and so does the base node corresponding to an ontology individual - has a primary node type and optionally a set of mixin node types. The extracted ontology classes from the primary or mixin node type definitions, indeed, form the set of classes for that individual.

Classes inferred from the node types assigned to a base node are not the only candidates for the classes of that individual. In fact, in many cases the base node is directly or indirectly related to another node in the repository, which - from the JCR-to-Onto Bridge’s point of view - may be a categorization node and therefore an ontology class may have already been generated for it. In such a case, it is desirable to include the generated ontology class, too, in the definition of the classes for the individual. For example, based on Figure 3.18, the JCR node named “B737-800” is meant to be the base node for an individual construction process, where the individual is named after the node. The primary node type for the base node is “my:aircraftModelInfoType” and JCR-to-Onto Bridge has already lifted this node type to an ontology class named “co:AircraftModelInfo”. Therefore, “co:AircraftModelInfo” is intuitively a class for the individual “B737-800”. However, the base JCR node also contains a property named “instanceOf” that references another node in the workspace: “CommercialJet”. As you will notice, it belongs to a tree of categorization nodes, and consequently ontology classes have already been generated for the nodes in the tree. If explicitly stated

by the user, the fact that “B737-800” is an individual of the class “co:CommercialJet” should also be asserted in the knowledge base.

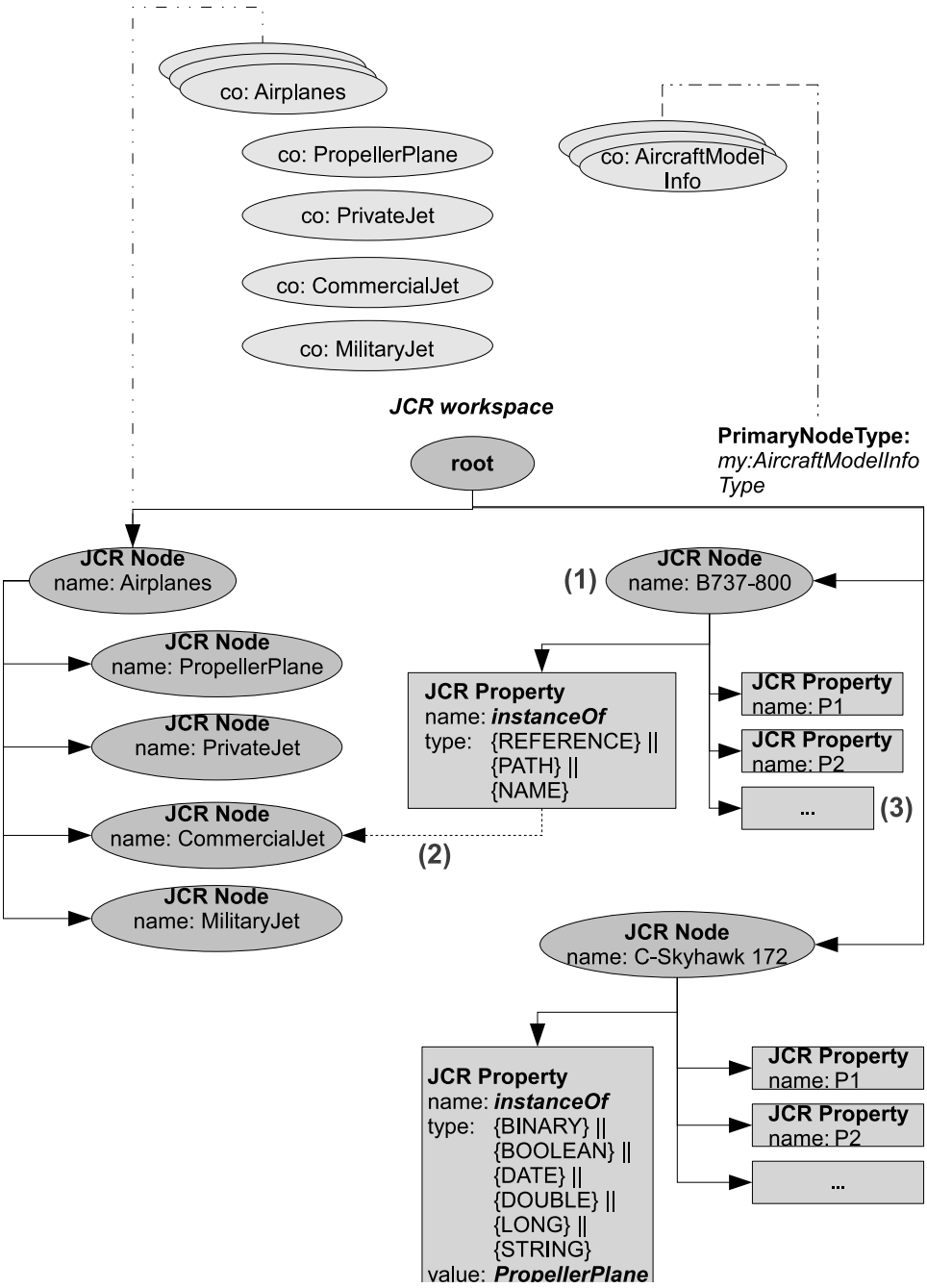

Figure 3.18: A sample workspace configuration for individual generation

Object or data type properties associated with the individual:

Some other valuable information to extract from the workspace is the object or data type properties of an individual and their values. It is important to note that the base node described above may contain other JCR properties that accounts for the generation of new ontology properties or setting up values for already existing ones. If the JCR property matches with that of an already described ontology property in the classes for the individual, then simply its value is set. On the other hand, for a JCR property that cannot be matched with any of the previously formalized ontology properties a new one should be generated. However, one difficulty with this approach is to determine the domain of the property. In most cases, the JCR-to-Onto Bridge ends up generating some auxiliary ontology classes to account for this challenge.

As a conclusion, it is necessary and sufficient to locate the base JCR node to be used for the individual construction stage and optionally a set of JCR properties that will contribute to setting up values of the associated properties. Therefore, two native queries; one for the selection of the base node and the other, whose path is described relative to that of the first, for selecting the JCR properties would suffice.

In JCR-to-Onto Bridge, this is formalized by the “InstanceBridge” mapping definition whose schema is displayed on Figure 3.19.

```
<xsd:element name=“InstanceBridge”>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name=“InstanceName”
                type=“tns:NameType” minOccurs=“0” />
            <xsd:element name=“Query” type=“tns:QueryType” />
            <xsd:element ref=“tns:PropertyBridge”
                minOccurs=“0” maxOccurs=“unbounded” />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3.19: The XSD for “InstanceBridge”

The “Query” element contains the query or the path expression to reach the node representing the individual. If present, the name of the individual is obtained from the element “InstanceName”; otherwise, the individual is named after the JCR node from which it is extracted.

Finally, the values of the object or data type properties are set, as each “PropertyBridge” within the “InstanceBridge” is processed.

3.4.5 Observations

It is worth pointing out that although these mapping definitions are used in an initial ontology extraction phase, they are likely to be valuable when the JCR-to-Onto Bridge is configured to listen to the events in the content repository. It is merely based on the fact that the content repository is in constant growth and that the changes in the repository should be reflected on the knowledge base. In such a scenario, the query forms a template to match against whenever a new event is fired from the content repository. If the particular update matches against any of the mapping definition, then the construction associated with the query should be replayed, if necessary. This is currently a work-in-progress and the details are beyond the scope of this thesis.

3.5 Summary

In this chapter, the motivations behind the ontology extraction framework, namely JCR-to-Onto Bridge have been discussed. JCR-to-Onto Bridge may be considered as a semi-automatic tool for extracting ontologies from JSR-170 compliant content repositories. As discussed in sections 2.6 and 3.4, though, the flexibility of a content repository model - as opposed to the rigid structure in a relational schema - makes it difficult to define in advance, a solid set of heuristics that are guaranteed to work with different repository instantiations. Therefore, in JCR-to-Onto Bridge, ontology extraction techniques are combined with a user-defined mapping strategy to obtain the optimum results. The potential benefits of such an ontology extraction framework for the content management system community is thoroughly discussed in [79].

CHAPTER 4

A HYBRID APPROACH INTEGRATING STRUCTURAL AND FULL-TEXT SEARCH

4.1 Background

One of the driving forces behind enhancing the content management cycle with semantic capabilities is to go beyond ranked keyword search. Currently, most content management systems support document retrieval based on full-text indexing and lexical match of terms. A purely keyword-based search as such has its own limitations, though. On the one hand some relevant results are omitted, on the other hand irrelevant results are returned; simply because the meaning of terms is not taken into account.

Consider the case where the user types in the keywords “home”, “drug”, “factories” to retrieve some news articles related to the illegal production of abusive substances. However, articles such as “Cannabis factory found by police[1]” are omitted because they do not explicitly contain the term “drug” or its lexical derivations. On the contrary, cannabis is a drug and therefore the article should have been included in the result set. A keyword-based query that uses lexical similarities cannot cope with such circumstances.

Now, consider the opposite case. The user executes the query with the keywords “drug”, “collapse”. The following articles appear in the result set:

- “China’s anti-malaria medicine producers face market collapse”: Just three years ago, a global shortage of the anti-malaria medicine artemisinin alarmed medics fighting the killer disease, and spurred scientists who are developing alternative sources of the

[1] http://news.bbc.co.uk/2/hi/uk_news/england/wiltshire/8039171.stm

drug...[2]

- "Arrests after boys' drug collapse": Police have warned people not to take tablets from a possible rogue batch of ecstasy after two teenagers were rushed to hospital from a Swansea nightclub...[3]

- "Donovan embarrassed by drug collapse": Eighties pop star Jason Donovan was embarrassed after he collapsed from a cocaine seizure during Kate Moss' 21st birthday party - because he was desperate to be seen as a wild rocker...[4]

The problem illustrated here is that even though all three articles contain the keywords "drug" and "collapse", the first article is business and finance related whereas the others are related to an event of being poisoned due to overdose. Therefore, it is necessary that the search scope is progressively narrowed based on some input from the user. In this regard, as stated in [48], faceted search enables users to navigate a multi-dimensional information space by combining text search with a progressive narrowing of choices in each dimension.

To overcome these limitations, content management systems support features such as synonym matching, similarity search and "Did you mean this?" suggestions. Synonym matching is based on the idea of repeating the search process with synonyms of a particular keyword. Similarity search deals with removing duplicate or near-duplicate results or once a particular content resource is selected, bringing in similar resources. Finally, "Did you mean this?" suggestions try to resolve incorrectly typed keywords.

However, even these techniques are not interlinked with the domain semantics available in the form of ontologies. Ontologies represent human knowledge explicitly in a form that is suitable for automated processing and if used in combination with full-text search, then a formal mechanism can be devised that provides enhanced search capabilities [6]. There are various works addressing this issue, which will be discussed in detail in Section 6.2. For convenience, here, we provide a brief summary.

Bast et al. [6] describe a semantic search methodology, that first finds concepts and individuals associated with the terms in the documents, and later on uses them as extensions to the original

[2] http://www.innovations-report.com/html/reports/life_sciences/report-100478.html
[3] http://news.bbc.co.uk/2/hi/uk_news/wales/4777812.stm
[4] http://www.contactmusic.com/news.nsf/article/donovan%20embarrassed%20by%20drug%20collapse-_1044537

full-text index. Minack et al. [39] take a somewhat different approach and run full-text search over RDF annotations; therefore extending structural search capabilities of SPARQL [49] with those of Lucene [42]. Finally, QuizRDF [40], described in Davies et al.'s paper, indexes both the document content and its RDF metadata. It lets the user start with some keywords to find and rank the relevant RDF resources. From these RDF resources, the associated concepts are found and presented to the user. The user now selects a concept, provides some additional query parameters based on the properties of the concept and the search scope is narrowed.

As we have seen in Chapter 3, JCR-to-Onto Bridge framework already formulates the node type definitions and patterns in the content repository as ontologies and, if necessary, enriches them with additional domain semantics. Inspired by this, we propose to build a hybrid search mechanism that combines the full-text search capabilities of engines such as Lucene [42] and SOLR [45] with the knowledge accumulated in the extracted ontologies. Our approach is different from the work of Bast et al. since it does not rely on modifying the native full-text indices to provide semantic search capabilities. In [6], the ontology is woven into the documents by adding artificial words into the corpus. On the other hand, we maintain the ontologies separately from the textually indexed documents and combine the search results at a latter step; hence, exploiting both the asserted and the inferred structural and semantic dependencies in the ontology. For a content management system scenario, we cannot utilize an approach similar to the one described by Minack et al. [39] either; since populating the ontologies with the full content of a resource is not practical. In this respect, our hybrid search mechanism complements QuizRDF [40] by providing support for queries around multiple classes. Our aim is twofold:

1. Seamlessly integrate semantic search facilities into keyword-driven interfaces: As argued in [6], keyword-based queries are a natural way to capture human knowledge. Therefore, instead of completely deviating from keyword-driven interfaces, semantic search mechanisms have to be built on top of contemporary paradigms. As a consequence, we should be able to find ontology concepts given a set of keywords and possibly construct complex semantic queries around them.

2. Provide support for faceted-browsing of content resources: Just like ontology browsing is a process that involves first visiting a set of nodes and then iteratively traversing all their connected nodes; so is search that utilizes ontologies in the background. We

employ this paradigm in our framework and therefore provide a multi-level, multi-dimensional interface for browsing search results.

The chapter proceeds as follows: First we discuss how ontologies play a central role in extending keyword-based search: ontology-lookup can be valuable in finding similar content or related terms and enables us to provide a multi-dimensional view over the content resources. In the proceeding section, we present the details of our hybrid search algorithm that combines the power of structural and semantic search with that of the full-text. The discussion proceeds with the complexity analysis of the proposed solution. Rather than providing a complete mathematical analysis for the best, average and worst cases, the discussion is restricted to broadly identifying the parameters that contribute to the complexity. The last but not the least, how the search mechanism is wrapped as a loosely-coupled RESTful [46] service is explained.

4.2 Motivation

4.2.1 Ontology Look-up for Similar Content

Within the context of a content management system, an ontology provides formal means to represent the relationship between content resources. Recall from Chapter 3 that during the lifting process, JCR-to-Onto Bridge represents each content node as an individual in the ontology. Based on the type definition of that particular node and other available information in the workspace, these individuals are associated with one or more ontology classes. Object and data type property values of the individuals are set with values obtained from the child property nodes of the base node in the content repository. Finally, the unique identifier for the content node, which in our case is the path expression to reach the node, is stored as a special data type property of the individual.

Whether or not a semi-automated framework such as JCR-to-Onto Bridge is used in ontologically representing the structure and semantics in a content repository, as long as a mechanism exists to locate the corresponding ontology artifacts for a particular content resource, information available in the ontology may be used to retrieve similar content resources. The idea is outlined as follows:

1. Either a structural query is executed or a look-up on a previously generated index containing all ontology resources is performed to locate the individual that corresponds to the content resource with the given unique identifier.

2. It is likely that the class for the individual and its superclasses contain other individuals that describe similar content resources. Therefore, starting from the direct class and going up to the superclasses, the individuals that belong to these classes are ranked by decreasing similarity. The same procedure applies to the subclasses.

3. An individual may belong to more than one class. In that case, it is fair enough to assume that the class with the least number of individuals is the most prominent one in determining the similarity among its individuals.

4. Individuals that belong to the same class may further be divided into groups of individuals that share same or similar property values or are connected through object properties. In that case, their similarity values should be adjusted accordingly.

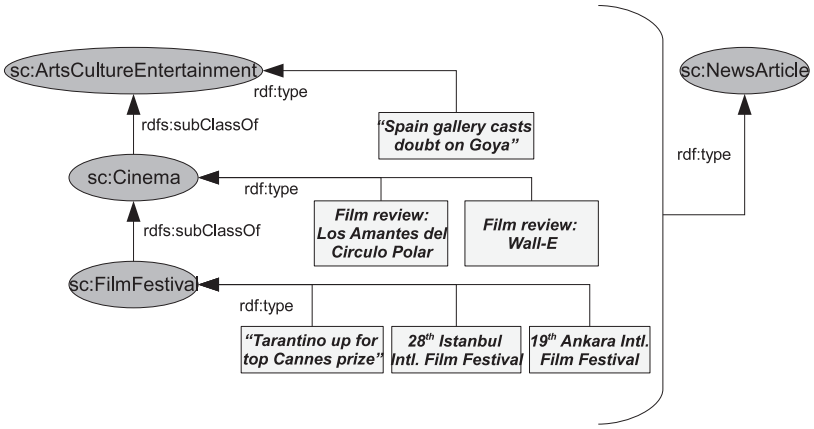

Figure 4.1: Metadata of some news articles in a content repository

Figure 4.1 illustrates a hypothetical ontology that contains classes and individuals representing the metadata about some news articles in a content repository. Assume that we desire to find content resources that are similar to the news article titled "28th Istanbul International Film Festival". Please note that the particular individual is an instance of the class "sc:FilmFestival". Consequently, a place to start looking for similar content would be the individuals of the class "sc:FilmFestival". On the other hand, "sc:Cinema" and "sc:ArtsCultureEntertainment" are all superclasses of "sc:FilmFestival", therefore, their individuals

might as well be candidates for relevancy, provided that their similarity values are lowered. One important point to mention is that “sc:NewsArticle” is not an appropriate selection for categorization; it simply contains far too many individuals.

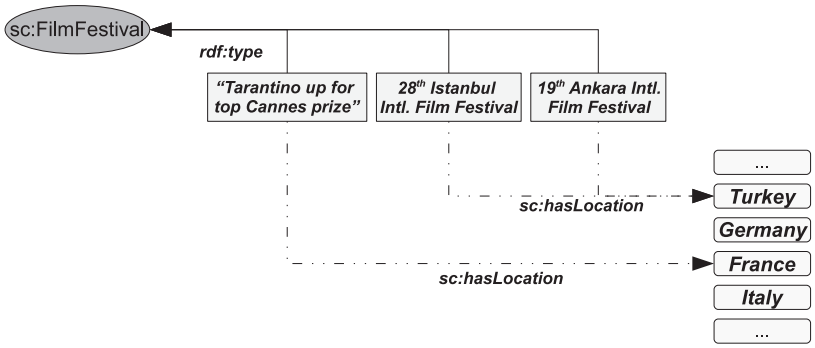

Figure 4.2: Adjusting similarity based on property values

Now assume that an object property labeled “sc:hasLocation” exists for the ontology class “sc:NewsArticle”. As depicted on Figure 4.2, values for this object property are available for the individuals that belong to the class “sc:FilmFestival”. It is now possible to fine-tune the previously calculated similarity values. In this respect, “19th Ankara International Film Festival” is a more relevant article to “28th Istanbul International Film Festival” than “Tarantino up for top Cannes prize” is.

In conclusion, once content resources are associated with the ontology individuals, it is possible to relate these resources based on the structural and semantic dependencies in the ontology. These dependencies appear in the form of inclusion by one or more ontology classes, the class hierarchy and possession of the same property values. In the proceeding sections, we illustrate how this information will be valuable in combining the results received from a full-text search engine such as Lucene [42] or SOLR [45].

4.2.2 Ontology Look-up for Related Terms

An ontology could be a valuable resource to look up terms that are semantically related to those provided in a textual query. Such an approach proves to be useful in cases where keyword-based search fails to return otherwise relevant resources. In this respect, full-text indexing techniques that work on ontologies have already been developed for some triple

stores [39] and offer means to exploit the methodology described in this section.

Given a set of keywords, performing an ontology look-up has two diverse practical implications:

1. A horizontal ontology such as WordNet [80] or dbPedia [81] can be used to discover the names of the semantically related resources. The original set of keywords is then expanded with those retrieved from the ontology look-up. Finally, search proceeds as purely full-text querying of the indexed documents with the expanded keyword set as the new input.

2. A domain ontology, possibly the one extracted by the JCR-to-Onto Bridge framework, is used in the look-up. Rather than using the names of the semantically related resources in a subsequent full-text search process, the structural and semantic dependencies in the ontology are retrieved to find related content resources.

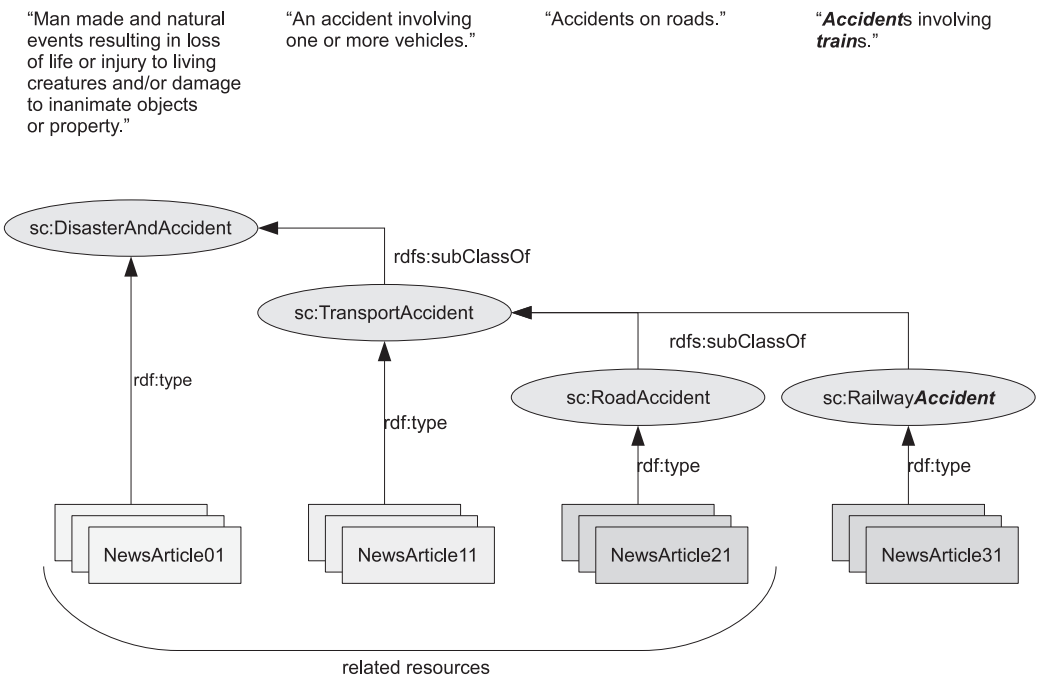

Figure 4.3: Example illustrating ontology look-up for finding related terms

As a possible example for case 2, consider Figure 4.3, which depicts a hypothetical ontology that contains some news articles and their annotations. The articles are categorized by

their IPTC News Subject Codes. Suppose now, that the textual query is around the keywords "train" and "accident". At this stage, an ontology look-up may be performed to find the relevant ontology resources. For this purpose, structural query engines enhanced with full-text search capabilities such as Lucene-SAIL [39] and LARQ [41] can be utilized[5]. Now that the textual annotation for the ontology class "sc:RailwayAccident" contains both of the keywords "train" and "accident", "sc:RailwayAccident" is returned. Based on this information, we conclude that the content resource "NewsArticle31" meets our search criteria. Furthermore, from the subsumption relations among the depicted classes, we conclude that the individuals "NewsArticle21", "NewsArticle11" and "NewsArticle01" also contain references to the relevant articles, in descending order of similarity.

4.2.3 Ontology Look-up for Faceted Browsing of Content

Until now, the discussion has mainly been on the utilization of ontologies in improving keyword-driven methodologies. In this regard, the possibilities for eliminating the weaknesses of full-text search were explored. Consequently, ways in which the result set could be expanded with semantically related results were identified. These improvements have to do with the completeness of the search mechanism: they aim to include in the result set a high percentage of all possible documents intended to be retrieved by the query. As a side effect though, they are likely to introduce some false positives, too, that is; documents that are completely irrelevant to the query context are brought in. Even without these semantic enhancements, full-text search itself is not sound; that is why a full-text search ranks resources based on its perception of relevance.

The fact that the accuracy of the results could be improved, if search proceeds in a faceted fashion, has been discussed in [48]. The idea is that instead of working with just a set of keywords obtained from the user, the search engine forms different views on the results based on some parameters communicated in diverse dimensions: possible content categories and keywords, the semantic links between the documents, etc. Consequently, the search scope is iteratively changed or narrowed until the user is satisfied with the results.

Ontologies play a critical role in enabling such faceted browsing of content and even if not named explicitly, their use has been experimented in various works such as [38], [6] and

[5] Lucene-SAIL and LARQ work on Sesame and Jena2 triple stores respectively.

[40]. In summary, provided that the content resources are annotated with the resources in the ontology, the search scope can iteratively be changed or narrowed simply by executing queries on the ontology and then retrieving the associated results. The problem is in seamlessly constructing these queries based on the parameters communicated by the user. We propose a somewhat similar solution to [40], but relieve the restriction that queries should revolve around a single class and enable queries to be written that join multiple ontology classes. The idea can be outlined as follows:

1. Ontology resources associated with the documents in the result set, including classes, object and data type properties, are retrieved and ranked based on their relevancy to the search criteria.

2. As the user selects from multiple ontology classes and provides values for their object or data type properties, semantic queries are seamlessly formed in the background.

3. Search is repeated with optionally a new set of keywords and the previously formulated query. The new result set that is relevant to the keywords and either expanded, filtered or pruned according to the restrictions imposed by the query is retrieved. The whole process starts over from step 1, until the user is satisfied with the results.

4.3 Implementation Details of the Hybrid Search Algorithm

4.3.1 Overview

In the preceding section, how ontologies play a central role in semantic search has been discussed. This section describes the building blocks used in the implementation of the hybrid search algorithm that combines the power of semantic search on ontologies with that of the full-text search on documents. As opposed to some earlier works [38], [6], [39], [40] and [41] that address the same problem, the two processes, that is; using ontologies as the basis for the semantic search and executing purely text-based queries on documents, are kept separate until the results are finally merged into a scored set. This way, the power of the two different approaches can be fully exploited without a loss of generality. The search mechanism described in this section is wrapped as a loosely-coupled RESTful [46] service so as to encourage its deployment by the content management system development community.

The input to the algorithm is a set of keywords complemented by a structural query part. As outlined in [6], keywords are an easy way to capture the human intention behind search. On the other hand, structural queries are necessary to filter-out unwanted results or reversely to extend the result set with other relevant resources. Structural queries are composed of pattern matching functions and names of ontology constructs and resources. They provide an easy way of encoding information necessary to locate the desired ontology resources.

Given a set of keywords and a structural query, the proposed hybrid search algorithm can be outlined as follows:

1. The set of keywords are used in consulting a full-text search engine such as Lucene [42] or SOLR [45] to retrieve the unique identifiers of the related documents. The unique identifiers, along with the relevancy scores assigned by the full-text search engine, are stored in “Document Set A” as depicted on Figure 4.4.

2. The unique identifiers in “Document Set A” are used for finding the ontology individuals with which the documents are annotated. Based on a degree of flexibility, the ontology is then traversed to find the semantically related set of resources. Traversed ontology classes are pushed into “Resource Set A”. On the other hand, the traversed individuals may indeed be annotations of other documents in the corpus. In that case, these documents are assumed to be semantically related to those in “Document Set A”, hence “Document Set A” is updated as shown on Figure 4.4.

3. Ontology resources and their textual annotations are lexically searched based on the provided set of keywords. The assumption is that the located ontology resources may offer means to find other semantically related documents. Based on a degree of flexibility, the ontology is then traversed and the linked resources in the graph are found. Traversed ontology classes are pushed into “Resource Set B”. On the other hand, the traversed individuals may contain references to other documents in the corpus. In that case, “Document Set B” is constructed with the unique identifiers of these documents and a relevancy score assigned to them by the algorithm.

4. The structural query is executed on the ontology. The ontology is then traversed to find the resources in the graph that are linked to those located by the query. The traversed ontology classes are pushed into “Resource Set C”, whereas the documents corresponding

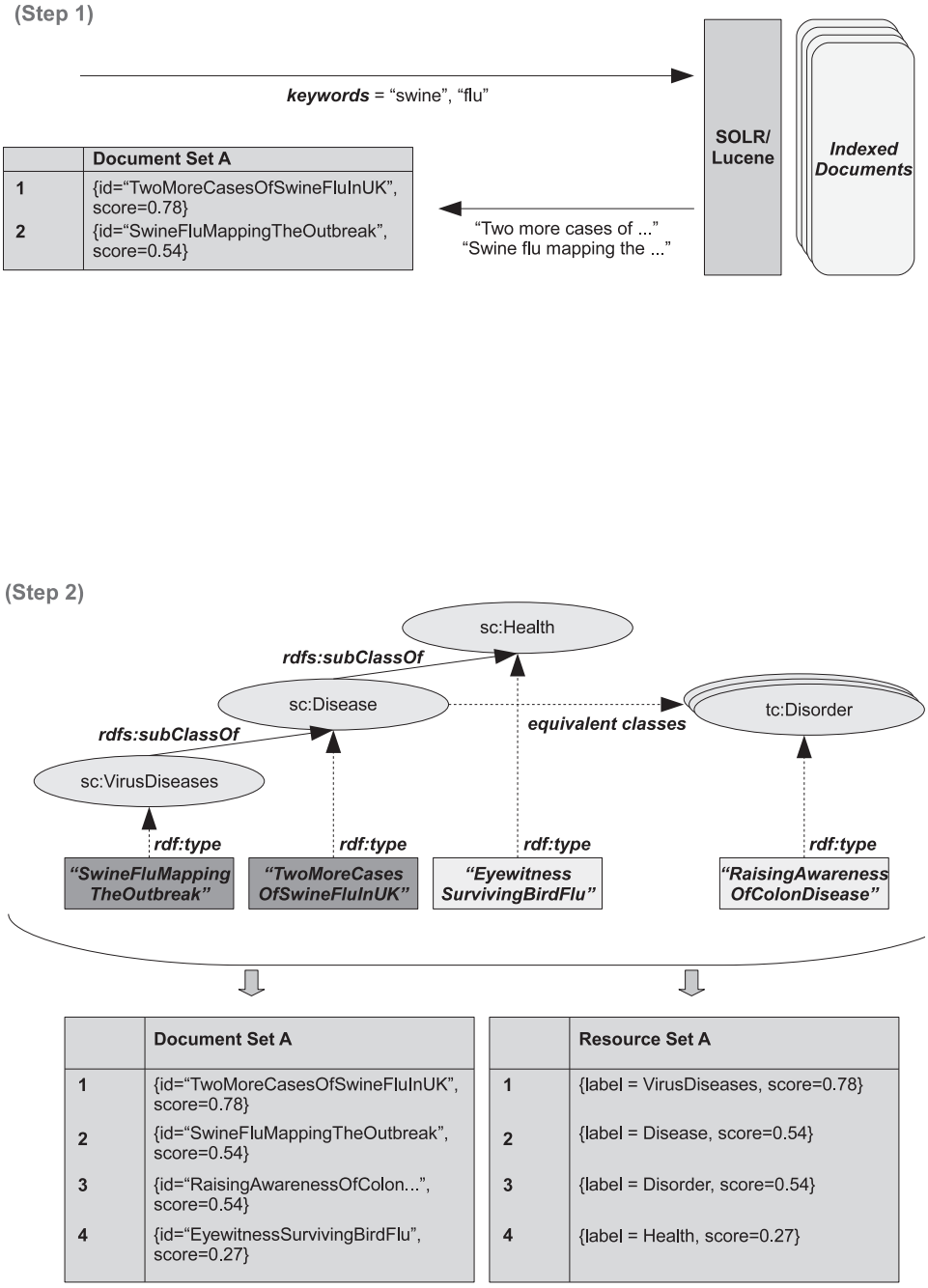

Figure 4.4: Extending full-text results with semantically related documents

to the traversed individuals and their relevancy scores are accumulated in “Document Set C”.

5. “Document Set A”, “Document Set B” and “Document Set C” are normalized and

merged based on some pre-configuration. This constitutes the ranked set of documents returned by the search process.

6. “Resource Set A”, “Resource Set B”, and “Resource Set C” are normalized and merged based on some pre-configuration. The merged set constitutes the ranked set of ontology resources to be used in faceted search.

7. Search is repeated with another set of keywords and a structural query formed by a partial subset of the ontology resources returned in the former step.

The output of the algorithm is a list of documents, sorted in descending order according to their relevancy scores, and a set of ontology resources that are valuable in faceted search and browsing of the documents. A detailed description of the algorithm, the heuristics used in assigning the relevancy scores to the documents as well as the ontology resources, and finally the normalization and merging methodologies are described in the subsequent sections.

4.3.2 Enhancing Full-text Search Results with Related Documents

The first building block of our hybrid search solution exploits the structural and semantic relations in ontologies to locate content resources that are semantically related to those provided as input. Obviously, the assumption is that prior to executing the algorithm, a set of unique identifiers for the documents is retrieved from a full-text search engine. In this respect, the aim is to extend the full-text search result set with possibly other related results.

The problem addressed herein can be reduced to the problem of finding the ontology nodes that are reachable in at most a predefined number of steps from a given base node. The base node is an ontological representation of the document for which we are trying to find its semantically related kinsmen. However, not every link will imply a semantic similarity between a pair of ontologically represented documents. Even if so, the semantic similarities between the pair of documents will vary. Therefore, within the scope of this thesis, various heuristics have been developed so that the traversed resources represent with a high degree of probability, the documents that are semantically related and that their similarity scores are computable. These scores are later on used in combining the results with those obtained from other building blocks of the algorithm. Here, we provide the details of the heuristics developed in finding and scoring the relevant documents.

Pattern matching for finding ontological complements of document resources:

The first problem to address is finding the individuals (or equivalently resources) in the ontology that semantically represent the set of input documents. Even though the outlined methodology works mainly over the ontologies extracted by the JCR-to-Onto Bridge framework, it can easily be customized for other solutions.

JCR-to-Onto Bridge formalizes each content item mapped with an "Instance Bridge" as an individual in the ontology. It automatically assigns the native JCR path of the node as the value of a reserved data type property named "http://www.srdc.com.tr/iks/jcr2ont#path". Based on this information, locating the desired individual is a matter of executing a structural query. In our hybrid search algorithm, we use the Jena framework [27], [28]. For executing structural queries on the RDF triples, we use LARQ [41]. The structural query we use in locating the ontology individuals is:

```
PREFIX pf: <http://jena.hpl.hp.com/ARQ/property#>
PREFIX jcr2ont:<http://www.srdc.com.tr/iks/jcr2ont#>
SELECT ?doc {
        ?lit pf:textMatch $contentRepositoryPath$.
        ?doc jcr2ont:path ?lit
}
```

At run-time, $contentRepositoryPath$ is replaced by the unique identifier of the document whose ontology complement we desire to locate. The query selects the subjects of all triples whose "http://www.srdc.com.tr/iks/jcr2ont#path" predicate has the desired value.

Heuristics for finding and scoring similar content:

Once the ontological complements of the initial set of documents are found, it is reasonable to traverse the ontology in search for complements of other semantically related documents. Documents are annotated by individuals in the ontology; therefore we will limit ourselves to finding the semantically related individuals. Nevertheless, the traversed ontology classes and properties will prove useful for implementing some faceted-browsing features; hence shall be

recorded on-the-fly.

Motivated by the observations outlined in Section 4.2.1, an individual shares a high degree of similarity with those that belong to the same class. On the other hand, an individual may belong to multiple classes. In that case, a mechanism is necessary to rank the classes based on the strength of classification they provide. Finally, the scores have to be fine-tuned based on the distribution of the individuals in the input set; that is, classes that possess more individuals from the input set are likely to contain the individuals that are more relevant. These issues are addressed by the following heuristics:

- The individuals in the input set denoted by $\text{IndividualSet}_{(input)}$, obtain their initial scores from the scores assigned to their complement documents (i.e. $\text{DocumentSet}_{(input))}$ by the full-text search engine;

  $$\forall ind, \text{doc } (\text{ind} \in \text{IndividualSet}_{(input)} \wedge \text{doc} \in \text{DocumentSet}_{(input)})$$
  $$\text{complementOf(ind, doc)} \rightarrow \text{score(ind)} = \text{f(score(doc))}.$$

  "complementOf" is a symmetric relation that implies that an individual in the ontology exists and is identifiable through the unique identifier of the document, and vice versa. The function f transforms the scores from the scale used by the full-text search engine to a native scale used by the proposed hybrid search algorithm.

- The score of an individual in $\text{IndividualSet}_{(input)}$ is conveyed to each class of the individual, in an amount that is inverse exponentially proportional to the total number of individuals the class possesses. This condition does not hold for classes that have explicitly been placed in the ignore list. Their scores are simply taken as zero;

  $$\forall ind, ontClass \ (\text{ind} \in \text{IndividualSet}_{(input)} \wedge \text{ontClass} \in \text{OntologyClasses})$$
  $$\text{type (ind, ontClass)} \rightarrow$$
  $$\text{conveyedScore(ind, ontClass)} \propto \frac{1}{DFACTOR^{rankOf(ontClass,ind)^2}}.$$

  "OntologyClasses" denotes the set containing all of the classes in the ontology. "type" is a relation that asserts that the individual denoted by its first argument has the class denoted by its second argument. DFACTOR is a constant used throughout the hybrid search algorithm and has a value that is greater than 1. "rankOf" is a binary function that returns the order of a class within the set of all direct classes of a particular individual, where the set is sorted in ascending order according to the total number of individuals

the classes possess. Finally, conveyedScore is a binary function that determines the score a class receives from a particular individual.

Suppose that ConcordeCrashFlight4590 $\in$ IndividualSet$_{(input)}$ and score(ConcordeCrashFlight4590) = 0.94 for a particular query. Now, assume that NewsArticleItem and AirTrafficAccident are the only direct classes of ConcordeCrashFlight4590. Furthermore, NewsArticleItem has a total of 14,568 and AirTrafficAccident 136 individuals. In this case, rankOf(NewsArticleItem, ConcordeCrashFlight4590) = 2; since when sorted in ascending order based on their total number of individuals, NewsArticleItem is placed second in the set of all classes of ConcordeCrashFlight4590. Consequently; conveyedScore(NewsArticleItem) $\propto$ 0.05875, provided that DFACTOR = 2.0.

This condition ensures that classes that are least involved in categorizing their individuals are assigned low scores. In determining the classes that are not directly involved in the categorization process, we simply use the individual count of the class as the basis for comparison. We assume that among the set of classes of an individual, the ones that contain the highest number of individuals are least likely to provide any valuable implications for semantic similarity.

- The score of a class is the summation of the scores conveyed by its direct individuals in IndividualSet$_{(input)}$, divided by the total number of individuals the class possesses. There are two direct consequences of this condition:

  1. A class that contains more individuals from IndividualSet$_{(input)}$ is likely to receive a higher score,
  2. The score that a class receives from its individuals is no greater than the score of any of its direct individuals.

After the individuals and their direct classes are scored, it is now possible to traverse the ontology in search for semantically related resources. The scored ontology classes are available in the set ClassSet$_{(directlyRelated)}$. For each ontology class in ClassSet$_{(directlyRelated)}$ the function **computeClosureForOntClass***(ontClass, flexibilityIndividuals, flexibilityClasses, initialScore, dFactor)* is called, which in turn traverses the related set of ontology nodes and assigns their scores. Below are the details of the function.

The function “computeClosureForOntClass” takes as arguments an ontology class and its

score, two separate flexibility factors that determine how far away from the ontology class search can proceed and the value for DFACTOR, whose purpose has been described earlier. The traversal proceeds as follows:

1. A list of equivalent classes for the given input are retrieved and placed in ClassSet-$_{(indirectlyRelated)}$. The score of each class placed in ClassSet$_{(indirectlyRelated)}$ is $\frac{1}{dFactor} \times initialScore$.

2. The list of all superclasses and subclasses and their equivalent classes are retrieved and placed in ClassSet$_{(indirectlyRelated)}$. The flexibilityClasses factor determines how many levels up or down from the base class the algorithm is allowed to traverse. For instance, when flexibilityClasses = 1, only the direct superclasses and subclasses of the base class are retrieved. When flexibilityClasses = 2, the direct superclasses and their direct parents as well as the direct subclasses and their direct children are retrieved. The score of each class placed in ClassSet$_{(indirectlyRelated)}$ is computed as $\frac{1}{dFactor^{distance}} \times initialScore$

Up to now, the algorithm has computed the classes in ClassSet$_{(directlyRelated)}$ and its closure ClassSet$_{(indirectlyRelated)}$; however, it has neither traversed nor scored their individuals. In this respect, first ClassSet$_{(directlyRelated)}$ and ClassSet$_{(indirectlyRelated)}$ are merged into ClassSet$_{(related)}$ while the duplicates are eliminated. Afterwards, the direct individuals of the classes in ClassSet$_{(related)}$, that have been reached by at most flexibilityIndividuals steps, are placed into IndividualSet(related) where the score of each class is assigned directly to its individuals. The set can optionally be expanded with other individuals that are reachable via object properties. In either case, not all of the individuals in the final IndividualSet$_{(related)}$ will have complements in the document corpus. However, from those which do have a complement in the corpus, the unique identifiers for the documents can be retrieved by accessing the reserved data type property values of the individuals. At this step, the algorithm has found the semantically related documents and has placed them in DocumentSet$_{(output)}$. Apart from establishing the grounds for scoring the individuals, the two sets: ClassSet$_{(directlyRelated)}$ and ClassSet$_{(indirectlyRelated)}$ provide means for faceted-browsing of the content resources, that is; they enable keyword search to be enhanced with structural and semantic queries as shall be discussed in Section 4.3.4.

4.3.3 Concept-Driven Retrieval of Results

Using the results obtained from a full-text search engine as the basis for semantic search is a promising, yet, an incomplete solution. Even though the heuristics materialized in Section 4.3.2 offer the ability to extend the result set with semantically related documents, the initial set may be incomplete to work with. To overcome this difficulty, we propose various techniques to incorporate also the results of an ontology look-up before the related documents are traversed and retrieved. This forms the second building block of our hybrid search solution. Our case becomes clearer with an example.

As discussed in Section 4.2.2, alignment with the dbPedia ontology [81] conveys surprisingly good results in terms of what additional semantic search features can be supported. Suppose now, that an ontology extracted from the repository of a news content management system is aligned with the dbPedia ontology. Consequently, the following triple becomes accessible:

Subject: <http://dbpedia.org/resource/Air_Berlin>

Predicate: <http://dbpedia.org/ontology/hubairport>

Object: <http://dbpedia.org/resource/Munich_Airport>.

The user now executes the query with the keywords: “Munich Airport”. However, the full-text search engine cannot find a document that explicitly contains these terms. Under these circumstances, the approach described in Section 4.3.2 will fail as it does not have any input to work with! On the other hand, we have among our ontology resources an individual whose URI matches the keywords. “Munich_Airport” does not have any complementary document in the repository; however “Air_Berlin”, which is connected via the object property “hubairport”, does. To retrieve such resources, a keyword-based look-up on the registered ontologies is necessary.

There are various works that address the problem of full-text search in ontologies. As mentioned earlier, Lucene-SAIL [39] and LARQ [41] are among these. Once a keyword-based look-up on the ontology is performed, the returned ontology classes as well as the ontology individuals could serve as the starting points for the ontology traversal described in Section 4.3.2. The initial scores for these resources can be retrieved directly from the structural query engine or equivalently can be computed via the string similarity metrics utilized in [82]. Con-

sequently, DocumentSet$_{(output)}$, IndividualSet$_{(related)}$ and ClassSet$_{(related)}$ are also populated for this building block.

4.3.4 Iterative Browsing of Results in Multi-Dimensions

The third building block of our hybrid search algorithm is the component that facilitates faceted-search. As outlined in Section 4.2.3, the proposed methodology addresses two diverse issues:

1. retrieving the ontology resources that have somehow been traversed for the purpose of finding the related content,
2. answering structural and semantic queries and using them as the basis for expanding, filtering or pruning the previously obtained results.

ClassSet$_{(related)}$ described earlier in Section 4.3.2, contains all the ontology classes that are identified while the semantic relations present in the ontologies are traversed. Along with the labels and URIs of the classes, these two sets contain individually the score assigned to each class. As one would recall, the score is an indication of how relevant the class is to either a set of documents or keywords. In the former case, the ontology is used in finding the semantically related resources to a given set of documents that are retrieved from a full-text search engine. In the latter case, the keywords are directly looked-up from the ontology. In any case, ClassSet$_{(related)}$ grants means to present the key assets (e.g. possible content categories, related terms, the semantic links between the documents, etc.) useful for constructing the structural query described in the second approach. A demonstration of this feature is provided on Figure 4.5. Here, it is possible to see that ClassSet$_{(related)}$ is directly used in displaying the categories associated with the search results. The user may now select multiple categories to designate the next search direction.

The issue presented in (2) requires more than just structural query handling to empower the desired behavior, that is; to expand, filter or prune the result set. In this respect, together with a set of structural queries, the user is asked to provide the selection criteria that go along with them. The following XML Schema Definition (XSD) [83] illustrated on Figure 4.6 provides a more formal description of the expected input:

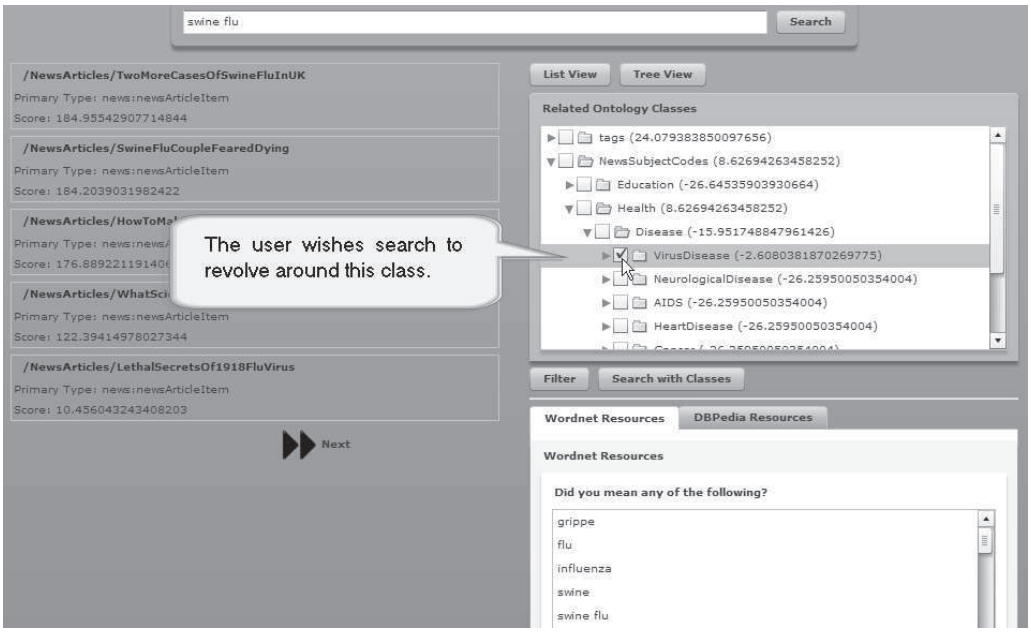

Figure 4.5: Faceted-search facilities of the proposed hybrid search solution

The “ResourceList” element contains 0-unbounded instances of “SelectiveResource” elements. Each selective resource holds either the URI of a particular ontology resource or directly a SPARQL [49] query that selects multiple resources from the ontology. An “Operator” may be defined over the selected resources and the operator possesses one of the following values: “OR”, “NOT_SELECTIVE”, “EXLUDE”. In case an operator is not provided for a particular “SelectiveResource” element, the default implication is “OR”.

The intention behind the operators is to solicit how the user desires the resources to be processed along with the results obtained from the other two building blocks of the algorithm (see sections 4.3.2 and 4.3.3). For convenience, we denote the results obtained from the other two building blocks of the algorithm; namely, $\text{ClassSet}_{(directlyRelated)}$ and $\text{ClassSet}_{(indirectlyRelated)}$, by $\text{Resources}_{(keyword-driven)}$ and those answered as a result of the “SelectiveResource” query by $\text{Resources}_{(selective)}$. Based on this information, the three cases to consider can be summarized as follows:

1. **Operator “OR”**: $\text{Resources}_{(keyword-driven)}$ and $\text{Resources}_{(selective)}$ are merged. In case of any duplicates, the scores are added together.

```
<xsd:element name="ResourceList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="SelectiveResource"
          minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Operator" type="tns:OperatorType"
                minOccurs="0" maxOccurs="1"/>
            <xsd:sequence>
              <xsd:choice minOccurs="1" maxOccurs="1">
                <xsd:element name="ResourceURI"
                    type="tns:non_empty_string"/>
                <xsd:element name="SPARQLQuery"
                    type="tns:non_empty_string"/>
              </xsd:choice>
            </xsd:sequence>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>


<xsd:simpleType name="OperatorType">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="OR"/>
    <xsd:enumeration value="NOT_SELECTIVE"/>
    <xsd:enumeration value="EXCLUDE"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 4.6: The XSD for the "ResourceList" element

2. **Operator "NOT_SELECTIVE"**: $\text{Resources}_{(keyword-driven)}$ and $\text{Resources}_{(selective)}$ are again merged; however, in case of any duplicates the score of the resource in $\text{Resources}_{(keyword-driven)}$ is lowered by the amount attributed to the same resource in $\text{Resources}_{(selective)}$. If the resource does not exist in $\text{Resources}_{(keyword-driven)}$, then its score is subtracted from 0.

3. **Operator "EXCLUDE"**: After $\text{Resources}_{(keyword-driven)}$ - $\text{Resources}_{(selective)}$ is computed, the scores in the final result set are re-normalized. If necessary, some previously omitted results in $\text{Resources}_{(keyword-driven)}$ can now be included. Consequently, the results that the user does not wish to be conveyed are completely omitted and more focus is given to the classes and their relatives that are not excluded.

The order of precedence in executing these operations is EXCLUDE, NOT_SELECTIVE and OR, respectively. When all of the desired operations are executed, we end up with a set of ontology classes denoted by $\text{Resources}_{(merged)}$. It is now possible to find and score the individuals these classes possess and furthermore find the final set of associated documents. These procedures have already been discussed in detail in Section 4.3.2, and therefore will not be presented here. However, as usual, we end up with the following sets: $\text{DocumentSet}_{(output)}$, $\text{IndividualSet}_{(related)}$, $\text{ClassSet}_{(directlyRelated)}$ and $\text{ClassSet}_{(indirectlyRelated)}$, where $\text{Resources}_{(merged)} = \text{ClassSet}_{(directlyRelated)} \cup \text{ClassSet}_{(indirectlyRelated)}$. These sets constitute the final results to be displayed to the user.

4.4 Complexity Analysis of the Hybrid Search Approach

In this section we will determine the computational complexity of our hybrid search algorithm. In doing so, we will follow a bottom-up approach and justify the order of complexity for the innermost functions first; before presenting those of the building blocks and that of the algorithm as whole. It is important to point out that rather than providing a complete mathematical analysis for the best, average and worst cases, we will restrict our study to broadly identifying the parameters that contribute to the complexity of the algorithm. In this respect, our aim is to outline some possible optimizations as future work.

After some internal preprocessing, each of the three building blocks presented in Section 4.3 call the function: "computeClosureForOntClass(ontClass, flexibilityIndividuals, flexibilityClasses, initialScore, dFactor))", whose main role is to return the ontology resources that are linked to the given ontology class (i.e. denoted by the "ontClass" argument) with a distance less than the values provided by the *flexibilityIndividuals* and *flexibilityClasses* parameters. In this respect, it is merely a graph traversal problem; where, in the worst case, the whole graph may need to be visited regardless of the value of "flexibility", except for when flexibilityIndi-

viduals=0 and flexibilityClasses=0. This is especially true if most of the ontology resources are gathered around “ontClass”. Consequently, the worst-case complexity of this function is O(n), where n denotes the number of triples in the ontology.

The function “getPathRelatedResources(searchResults, flexibilityIndividuals, flexibilityClasses)” makes up the core of the first building block described in Section 4.3.2. It simply receives a list of scored documents and tries to find the semantically related ones. It starts out by finding the ontological complement of each document (i.e. individuals with their reserved data type property set to the unique identifier of the document). This process takes O(r) operations, where r denotes the number of documents in “searchResults”. The assumption is that retrieving the subject of a triple for a given predicate and object is a constant-time operation, as the triples are pre-indexed in the triple-store. Obviously, since the process has to be repeated for all r documents, the overall complexity becomes O(r). Once computed, the function proceeds to find the closure of these ontology resources. In a naïve approach, the worst case complexity becomes O(r×n) as the “computeClosureForOntClass” function has to be recalled for every resource retrieved in the former step. Therefore, the overall complexity of the building block becomes O(r) + O(r×n), where r and n denote the size of the input set of documents and number of triples in the ontology, respectively.

The second building block described in Section 4.3.3, first tries to locate the set of ontology resources whose labels and URIs are syntactically similar to the keywords provided. Without any prior indexing, it implies that all ontology resources are to be iterated and that for each ontology resource the string similarity should be computed. The complexity of such an approach is inevitably high, which is O(n) in the worst case. Here n denotes the number of triples in the ontology. The function then proceeds with the best matching c_1 resources found, where c_1 is a predetermined number, and computes their closure by the “computeClosureForOntClass” function. Therefore, the latter step takes O(n) operations in the worst case, provided that c_1 kept constant. Finally, the overall complexity of this building block is O(n).

The third building block enables structural and semantic queries to be executed before their closures are computed. As explained in [50], evaluating graph pattern expressions constructed by using AND, FILTER and UNION operators is an NP-complete task. On the other hand, when restricted to AND and FILTER operators only, the complexity becomes O(n×|P|), where n denotes the number of triples in the ontology and |P| the complexity of the graph pattern. By

allowing only patterns with AND and FILTER operators and restricting the available choice to the most c_2 relevant ontology classes, the complexity can be kept within manageable limits (i.e. |P| becomes constant). Of course, now the closure for the ontology resources that match the provided query needs to be computed. We may still achieve an O(n×|P|) worst-case complexity by keeping flexibilityIndividuals=0 and flexibilityClasses=0, that is; restricting ourselves to only the ontology resources returned by the query. In this case, this procedure has to be repeated for every "SelectiveResource" element described in Section 4.3.4. Again by imposing an upper bound, say c_3, on the number of "SelectiveResource" elements allowed, we limit the overall complexity of the building block to O(n×|P|).

The overall complexity of the proposed hybrid search methodology, with the aforementioned optimizations becomes max(O(r) + O(r×n), O(n×|P|)). As you would recall, r stands for the number of documents retrieved from a full-text search engine, n denotes the number of triples in the ontology and finally, |P| denotes the complexity of the query(-ies) used.

The value of r could be very large if we try to use all of the documents retrieved from a full-text engine as input to the hybrid search algorithm. Whether or not such a naïve approach should be taken is indeed subject to discussion. The intention behind using the full-text search results as input to the algorithm is to find the semantically related documents and possibly a set of assets valuable for forming the queries. On the other hand, in a faceted-search paradigm, the user is not really interested in whether or not the search engine retrieves all of the related resources at once. In other words, it is more important that the user sees only the documents that are related to the currently browsed full-text search results. In this respect, the search results may be processed in chunks in a procedure described as follows:

- Given some keywords, the first set of q_1 results are retrieved from the full-text search engine,
- These results are used as input to the hybrid search algorithm and the related documents and concepts are retrieved,
- Now, the user has the following options:
    1. Finalize search; the document she is looking for is in either the set of results received from the full-text search engine or the related documents found by the hybrid search algorithm.

2. Ask the retrieval of the next set of q1 results from the full-text search engine; a new set of related documents and concepts will be retrieved.
3. Use the returned concepts as the basis of forming queries to change or narrow down the search scope.
4. Repeat the search process with a new set of keywords.
5. A combination of (3) and (4).

With such an optimization, r is now bounded by the value used for q_1, which is a constant, and therefore the complexity of the algorithm is determined by O(n×|P|).

The second parameter that determines the complexity of the algorithm is n, which indicates the number of triples in the ontology. As outlined in [84], contemporary knowledge base systems are capable of handling more than one million triples. In this regard, the performance of the proposed hybrid search methodology degrades linearly as the number of triples asserted in the persistence layer increases. Therefore, some optimizations that would not interfere significantly with the quality of the result set are necessary. The function "computeClosureForOntClass" is too generic in the sense that it treats every ontology resource equally in determining the nodes that are linked to a particular ontology class. On the other hand, it is wise to traverse the classes of the ontology (i.e. TBox) first, before their individuals (i.e. ABox). Based on the scores assigned to the classes, individuals that belong to low scored classes may simply be ignored, if the total number of traversed resources already exceeds a given maximum bound. One would generally expect the ratio of $\frac{|TBox|}{|ABox|}$ to be significantly small for large n, therefore; "computeClosureForOntClass" becomes computationally manageable.

4.5 Wrapping it all as a RESTful Service

The search mechanism described in this section is wrapped as a loosely-coupled RESTful [46] service so as to encourage its use by the content management system development community. The benefit of such an approach is that the community can simply deploy it on top of an existing full-text search engine such as SOLR [45]. In fact, the proposed hybrid search framework does not contain any bundled full-text indexer.

The only prior configuration necessary to initiate semantic search is the registration of the

domain or horizontal ontologies. The global REST resource to register an ontology resides at the following URL:

"http://localhost:8080/PersistenceLayerService/ontologies/"

where the prefix "http://localhost:8080/PersistenceLayerService/" may vary from machine to machine. Registration proceeds with an HTTP POST [47] request on the provided resource with two form parameters; namely, "ontologyURI" and "ontologyContent". "ontologyURI" is a String that represents the base URI of the ontology to be registered. "ontologyContent" is the textual representation of the ontology in RDF/XML syntax [7]. If the ontology is successfully registered, it is assigned a unique identifier by the system and placed at the following location:

"http://localhost:8080/PersistenceLayerService/ontologies/<unique-identifier>/"

Various HTTP GET, POST and DELETE [47] functions are available on these ontology resources each providing a high-level ontology browsing and editing feature. The underlying triple store used is Jena [27]. It has been made database-aware so that the ontologies persist even if the service is stopped. For the prototype developed specifically in this thesis, we use MySQL Community Server v5.1 [85]. The technical details of this Persistence Layer framework and the provided REST interface functions will be discussed at length in the "Interactive Knowledge Stack for small to medium CMS/KMS providers" project deliverable [86]. Within the scope of this thesis, though, we will identify only those that are relevant to our search interface.

An ontology may be deleted by calling the HTTP DELETE operation on the URL (i.e. "http://localhost:8080/PersistenceLayerService/ontologies/<unique-identifier>/") where the ontology resides. Furthermore, its full textual content may be retrieved, again in RDF/XML syntax, by calling the HTTP GET operation with the "Accept" parameter set to "application/rdf+xml".

As you have noticed, multiple ontologies can be registered to the Persistence Layer. In such a case, it is possible to use all registered ontologies or individual ones as the basis for search. For a case that requires a look-up on horizontal ontologies such as dbPedia [81], the former solution would be preferred. The following two URLs are equipped with the REST functions that perform global and local search, respectively:

"http://localhost:8080/PersistenceLayerService/ontologies/search/"

"http://localhost:8080/PersistenceLayerService/ontologies/<unique-identifier>/search/"

The HTTP GET operation is called in either of the two URLs with the form parameter named "query". The response to the operation is of type "application/xml". The following excerpt displayed on Figure 4.7 from the XML schema declaration of the search interface provides a brief overview of the input and output structures used. The full syntax is provided in Appendix B.

"Query" and "Result" are respectively the input and output elements used in the search procedure. The "Query" element consists of the following parts: a set of keywords ("KeywordList"), a structural query ("StructuralQueryPart") and a list of unique identifiers of those documents obtained from the full-text search engine ("FullTextSearchResultList"). The attributes "flexibilityIndividuals" and "flexibilityClasses" determine the values to be used for the "computeClosureForOntClass" function described in Section 4.3.2. Finally, if the attribute "useSynonymsInOntologyLookup" is set to true, then the ontology look-up process described in Section 4.3.3 is repeated with the synonyms of the given set of keywords.

The "Result" element consists of the set of unique identifiers for the relevant documents and their metadata ("ReturnedDocuments"), the ontology resources to be used for faceted browsing ("ReturnedOntologyResources") and finally a set of synonymous and related terms retrieved from WordNet [80] ("ReturnedWordnetResources").

4.6 Summary

The previous sections discuss the various heuristics used in the three main building blocks of the hybrid search algorithm. As you would recall, the intention was to combine the full-text search capabilities of engines such as Lucene [42] and SOLR [45] with the power provided by ontologies. In this respect, we mainly had two concerns.

First of all, we did not wish to deviate extremely from keyword-driven approaches; as argued in [6], keyword-based queries are a natural and a powerful way to capture human knowledge. Consequently, different techniques that could be used to enhance the results of a textual query were explored. Our first argument was that ontologies could be used on an initial set of results

```
<xsd:element name="Query">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="tns:KeywordList"
                minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="tns:StructuralQueryPart"
                minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="tns:FullTextSearchResultList"
                minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="flexibilityIndividuals"
            type="xsd:integer" use="optional"/>
        <xsd:attribute name="flexibilityClasses"
            type="xsd:integer" use="optional"/>
        <xsd:attribute name="maxResults"
            type="xsd:integer" use="optional"/>
        <xsd:attribute name="useSynonymsInOntologyLookup"
            type="xsd:boolean" use="optional"/>
    </xsd:complexType>
</xsd:element>


<xsd:element name="Result">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="tns:ReturnedDocuments"
                minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="tns:ReturnedOntologyResources"
                minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="tns:ReturnedWordnetResources"
                minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 4.7: The XSD for the "Query" and "Result" elements

received from a full-text search engine to find the semantically related documents. The details of this approach were discussed in Section 4.3.2. On the other hand, a textual query could initially be used to perform search on the ontologies, too, in which case, another set of related documents would be retrieved. The details of this second approach were discussed in Section 4.3.3. These techniques aimed to improve the completeness of the search algorithm.

In trying to extend the result set with possibly related documents, we also introduced some noise. Therefore, our second concern was to eliminate the false positives in the search results. It was argued that providing a faceted-browsing over the results would solve the problem. The details of how the results could be expanded, filtered or pruned based on a selection of ontology resources were finally discussed in 4.3.4.

Our approach is different from the work of Bast et al. since it does not rely on modifying the native full-text indices to provide semantic search capabilities. In [6], the ontology is woven into the documents by adding artificial words into the corpus. On the other hand, we maintain the ontologies separately from the textually indexed documents and combine the search results at a latter step; hence, exploiting both the asserted and the inferred structural and semantic dependencies in the ontology. For a content management system scenario, one cannot utilize an approach similar to the one described by Minack et al. [39] either; since populating the ontologies with the full content of a resource is not practical. In this respect, our hybrid search mechanism complements QuizRDF. Finally, the clear separation of full-text and structural search mechanisms until the merging step enables input to be processed in chunks. This is one of the areas that can be exploited to increase the performance of the hybrid search solution. We plan to explore it as part of our future work.

CHAPTER 5

EVALUATION OF JCR-TO-ONTO BRIDGE AS AN ENABLER FOR SEMANTIC SEARCH

The motivation behind the JCR-to-Onto Bridge framework is to facilitate the use of semantic technologies within the context of a content management system. As argued in Chapter 3, JCR-to-Onto Bridge frees the semantics that would otherwise be locked up in the content repository. The repository itself does not provide the means for the development of semantic services; as it does not have any reasoning power. Consequently, applications have to deal with this problem internally, and for most of the time by impromptu solutions [87]. On the other hand, JCR-to-Onto Bridge formalizes the semantic information in the repository in an ontology, which may later on be enhanced with additional horizontal or domain knowledge. Inevitably, ontologies represent the knowledge explicitly in a form that is suitable for automated processing [6]; hence enabling such semantic features to be built and integrated with ease.

To demonstrate the value of the JCR-to-Onto Bridge approach, we have developed a hybrid search methodology that combines the power of semantic search on ontologies with that of the full-text search on documents. The hybrid search mechanism complements QuizRDF [40] by providing more complex faceted-search behavior and support for extended queries. It is highly decoupled from a full-text search engine and the service can be invoked anytime, anywhere through various REST operations.

In this chapter, we will demonstrate the power of the JCR-to-Onto Bridge framework through the value-added semantic search features enabled when the proposed hybrid search engine is executed on top of the ontology extracted from a content repository. The chapter proceeds as follows: first, the content repository used in the ontology extraction process is introduced.

Next, the ontology that JCR-to-Onto Bridge produces is presented. Following that, the search service is invoked with queries that demonstrate its value-added features including faceted-search.

5.1 Content Repository Used

The content repository used in this experiment is Apache Jackrabbit [51], which is a reference implementation for the JSR-170 specification [54]. The repository is populated with more than 50 news articles assembled in different categories from the BBC News site (http://news-.bbc.co.uk/). In addition to the news articles, the workspace contains a tree of categorization nodes whose values are obtained from the IPTC News Codes, specifically the subject codes [71]. The three level hierarchy depicted by the subject codes is expressed as a tree hierarchy in the content repository. Furthermore, an example "taxonomy" using namespaced tags [88] is again represented as a tree of JCR nodes. Each news article is tagged with numerous resources from this taxonomy.

A node type namely, "news:newsArticle" is registered to account for the type of all nodes that contain the news articles. Its supertype is "nt:base", from which all built-in and custom node types are inherited. Three properties, namely "title", "categorizedBy" and "content", are defined over each node of type "news:newsArticle". The three properties have the type declarations STRING, PATH and BINARY, respectively. The title of the news article is stored in the "title" property of the node and the full HTML [89] page in its "content" property. On the other hand, the "categorizedBy" property is used to give reference to a number news subject category or taxonomy nodes described earlier.

The news articles, with which the content repository is populated, are also added to the local SOLR [45] server for use during the semantic search process. SOLR creates indices on the textual content of these documents. For loading content into SOLR, an XML document, whose structure is outlined below, is sent to the server through an HTTP POST operation:

```
<add>
    <doc>
        <field name="id">/NewsArticles/GermanShipHijackedByPirates</field>
```

```
        <field name="text">
            Textual content of the news article.
        </field>
    </doc>
    <doc>
    </doc>
    ...
</add>
```

A special Java archive file (jar) has been prepared by the SOLR community to ease the invocation procedure. In this respect, registering the news articles to SOLR can simply be achieved by the following command-line invocation, where "allDocuments.xml" is the filename of the XML document described just recently:

```
java -jar post.jar allDocuments.xml
```

To delete a particular resource, the same library is used but instead with a different document[1]. The following command-line invocation clears all pre-built indices:

```
java -Ddata=args -jar post.jar "<delete><query>id:[* TO *]</query></delete>"
```

One important point to mention is that the field named "id" associated with a particular document is the native JCR Path expression used to locate the resource in the repository. As discussed earlier in Section 4.2.1, the hybrid search algorithm receives a set of unique identifiers for the retrieved documents from the full-text search engine. In this respect, we use the JCR Path as the unique identifiers of the documents we want SOLR to index.

5.2 The Extracted Ontology

The full JCR-to-Onto Bridge mapping definition used for extracting the domain ontology from the content repository is presented in Appendix C. Here, we provide a brief overview of the extracted ontology:

[1] In this particular example, the document is provided as an inline argument.

- All of the built-in and custom node types in the content repository are represented as ontology classes. The native inheritance hierarchy in the content repository is reflected as super-class, sub-class relations (see Section 3.2.1).

- The nodes in the workspace that belong the IPTC News Codes categorization tree and the "taxonomy" of namespaced tags are all represented as ontology classes. Since the nodes in question all fit to the "hierarchical pattern" described in Section 3.4.1, the super-class, sub-class relations are set accordingly. For example, "Cancer" and "NeurologicalDisease" are both subclasses of "Disease", and "Disease", in turn, is a subclass of "Health".

- For each JCR node of type "news:newsArticle", a class individual is generated. The individuals are named after the value of the "title" property in the workspace. By default, all of these individuals belong to at least the previously generated ontology class "http://www.srdc.com.tr/news#newsArticle". However, through the value referenced by the "categorizedBy" property additional container classes are asserted (see Section 3.4.4).

- The object property "http://iks-project.org/companyname/repositoryname/workspace-name#categorizedBy" is also asserted in the knowledge base based on the guidelines described in Section 3.4.3.

- The data type property "http://www.srdc.com.tr/iks/jcr2ont#path" is a reserved property used by the hybrid search algorithm to associate the actual documents indexed by the external full-text search engine with the individuals in the knowledge base. The value of the data type property is set to the native path expression used in the repository to reach the node.

Once the ontology is extracted by the JCR-to-Onto Bridge, it is aligned with a subset of the MeSH ontology described in [90] by asserting equivalence of certain classes through the Protégé-OWL framework. The ontology extracted by the JCR-to-Onto Bridge framework is loaded on to the search service in a manner described in Section 4.5. For ease of use, this process is handled as an event in a custom built Protégé plug-in.

5.3 Pilot Use Cases for Search

Here, we evaluate the value-added features of the proposed hybrid search methodology in the following dimensions:

1. To what extent keyword-based look-up of concepts enhances search results,
2. Given a set of full-text search results, to what extent the semantically related documents can be retrieved,
3. What faceted-browsing capabilities are provided.

To perform this demonstration, we have implemented a Java client that invokes the RESTful functions described in Section 4.5. On top of the Java client, a Rich-Internet Application (RIA) [91] implemented in Flex v.3 [92] is deployed.

Concept look-up may either be used for discovering the list of semantically related terms or for finding documents through the concepts in the ontology.

Case 1: The simplest case is suggesting related terms by scanning the horizontal ontologies such as WordNet [80] or dbPedia [81]. When queried with the keyword “hijack”, the following are the top results retrieved from the search service:

PiratesHijackShipOffSomalia, **Score**: 118

GermanShipHijackedByPirates, **Score**: 77

OperaExaminesScientistsDeath, **Score**: 8

SydneyOperaHouseArchitectDies, **Score**: -23

WorldMiddleEastTelAvivOperaDropsPlansForWagner, **Score**: -23

Most of these documents are directly obtained from the full-text search engine. On the other hand, articles such as “SydneyOperaHouseArchitectDies” and “WorldMiddleEastTelAviv-OperaDropsPlansForWagner” are present simply because they are semantically related to one of the articles that explicitly contain the term “hijack” , namely; “OperaExaminesScientists-Death”.

Along with these resources, various synonymous and related terms are also presented to the user. These terms include, but are not restricted to the following: “pirate”, “take over”, “seize”, “crime”. The user may now double click on any of these terms, say “seize”, and repeat the whole search process. In such a case, the new result set becomes:

PiratesSeizeShipOffSomalia, **Score**: 73

MaoistRebelsSeizeIndianTrain, **Score**: 16

PiratesHijackShipOffSomalia, **Score**: 1

GermanShipHijackedByPirates, **Score**: -32

GunmenSeizeDarfurAidWorkers, **Score**: -58

It is important to note that some articles that have not been included in the former result set are now available.

Case 2: Another use of concept look-up is for finding the ontology classes whose labels or URIs are syntactically similar to the given keywords.

As an example, consider the case when the search service is queried with the term “vinicultural”. The domain ontology that the hybrid search engine uses in the background contains the class “Viniculture”. Even though none of the indexed documents can be retrieved by full-text querying, the hybrid search engine responds by some quite surprising results:

PlantingBeginsOnANewVineyard, **Score**: 74

UKFarmsToastRecordWheatCrop, **Score**: -73

The algorithm first traverses the TBox of the ontology to find the class “Viniculture”, whose label portrays a high syntactic similarity with the given keyword. Consequently, “PlantingBeginsOnANewVineyard”, which is categorized by the news subject code “Viniculture”, is placed at the top of the list. Finally, the second article is categorized by the news subject code “Agriculture”. Since the class label has a lower degree of syntactic similarity with the keyword “vinicultural”, “UKFarmsToastRecordWheatCrop” has a lower score.

Case 3: A more complex case is a two-level look-up; where first, a set of related terms are retrieved from a horizontal ontology such as WordNet [80] and then the domain ontology used

in the background is scanned for the syntactically related concepts.

For our particular example, neither the documents nor the concepts in the domain ontology contain the term “lymphoma”. In fact, when queried with the option “useWordNetLookupForSynonyms” disabled, no results are retrieved. On the other hand, WordNet [80] asserts that “cancer” is its direct hypernym. Now, a WordNet enabled query results in the retrieval of the following classes and documents even when the keyword is “lymphoma”.

Classes:

Cancer, **Score**: 60

Health, **Score**: -36

NewsSubjectCodes, **Score**: -36

Disease: **Score**: -36

Documents:

MysteryDiseaseKillsHomosexuals, **Score**: 60

CancerBrakeCouldHaltDisease, **Score**: 60

The only documents that are categorized by the news subject code “Cancer” are indeed the two brought by the search service. Here, it is possible to see that once the ontology class “Cancer” is located, the algorithm traverses - based on a predefined degree of flexibility - its super and sub classes.

Up to now, various use cases have been presented that demonstrate the power of concept lookup on semantic search. On the other hand, expanding the result set with semantically related documents is another promising feature of the hybrid search mechanism. For this purpose, the result set is first populated with the documents retrieved from the full-text search engine. Later on, the set is expanded with semantically related documents whose similarity is inferred from the relations in the domain ontology.

Case 4: The simplest case where ontology look-up for similar content yields to good results is when the individuals that share a common class are treated as similar. Consider that the search service is experimented with the keywords “swine” and “flu”.

*TwoMoreCasesOfSwineFluInUK,

Score: 200, **categories**: Disease; **tags**: medical

*SwineFluCoupleFearedDying,

Score: 200, **categories**: VirusDisease; **tags**: health_care,

science_and_medicine

*HowToMakeASwineFluVaccine,

Score: 191, **categories**: HealthTreatment; **tags**: research

*WhatScientistsKnowAboutSwineFlu,

Score: 133, **categories**: Medicine, PrescriptionDrugs; **tags**: research, medical

*LethalSecretsOf1918FluVirus,

Score: 15, **categories**:; **tags**: science_and_medicine,

medical, recreational_drugs, research

*EyewitnessSurvivingBirdFlu,

Score: 15, **categories**: Disease, Medicine; tags:

SwineFluMappingTheOutbreak,

Score: 8, **categories**: VirusDisease; **tags**: medical

DrugFirmsInventingDiseases,

Score: 0, **categories**: HealthTreatment, Medicine;

tags: recreational_drugs, research

CancerBrakeCouldHaltDisease,

Score: 0, **categories**: HealthTreatment, Cancer;

tags: research, science_and_medicine

*LegionnairesDisease,

Score: -10, **categories**: Education; **tags**:

*MapPinpointsDiseaseHotspots,

Score: -13, **categories**: Disease, Ecosystem; **tags**: animals

RaisingAwarenessOfColonDisease,

Score: -21, **categories**: Disease; tags:

ExSchoolsChiefRevealsDisease

Score: -21, **categories**: Disease; **tags**: lifestyle

MuscleDiseaseCareFallsShort

Score: -21, **categories**: Disease; **tags**:

LymeDisease

Score: -21, **categories**: Education; **tags**: research

For convenience, the documents retrieved from the full-text search engine are marked with an asterisk. The others are retrieved due to their semantic similarity. For example, "CancerBrakeCouldHaltDisease" is semantically related to the individual "HowToMakeASwineFluVaccine" through the classes "HealthTreatment" and "research", and to the individual "SwineFluCoupleFearedDying" through the class "science_and_medicine". These concepts are displayed to the user as on Figure 5.1.

Case 5: An improved version of Case 4 is when the individuals that share a common class - and therefore are assumed to be related - are identified through ontology reasoning. The keywords for this example will be "atopic" and "eczema":

PyschiatryCanCureSkinDisorders, **Score**: 107,

GeneticCluesToEatingDisorders, **Score**: -88,

The only document that contains these keywords is the one titled "PyschiatryCanCureSkinDisorders"; and it is categorized by an arbitrary class named "Psychiatry_Psychology". One of

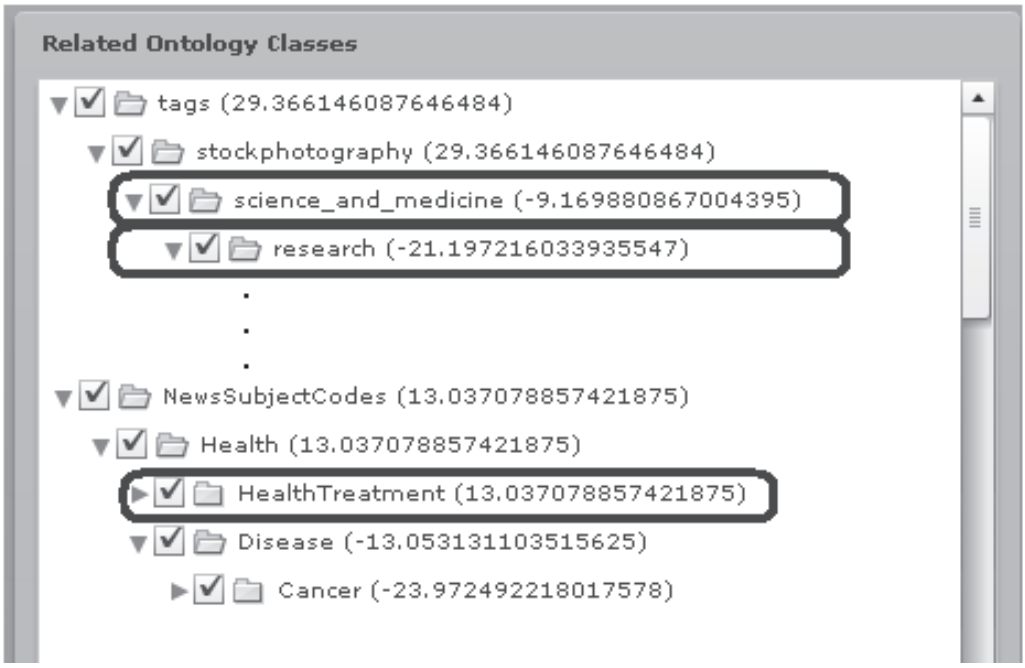

Figure 5.1: Concepts associated with the search results

the indirect subclasses of “Psychiatry_Psychology” is “Eating_Disorders”. On the other hand, the second document is categorized by a third class named “EatingDisorder”. Equivalence of the two classes “Eating_Disorders” and “EatingDisorder” is inferred through ontology reasoning. Consequently, “GeneticCluesToEatingDisorders” becomes an indirect individual of “Psychiatry_Psychology”; and that is why it is in the result set.

Case 6: The most complex case is when individuals that share the same property values, or equivalently those that are related through some object properties are identified as similar. Consider the case with the keywords “female” and “chancellor”:

UKsaysMerkelbacksFiscalBoost, **Score**: 127

MerkelOffersStateAidForOpel, **Score**: 66

GermanyAgreesBadBankScheme, **Score**: 16

EntertainmentStingScoopsMusicHonour, **Score**: 8

KylieSweepsAussieMusicAwards, **Score**: 8

Even though the article “GermanyAgreesBadBankScheme” does not contain the keywords, as illustrated on Figure 5.2, the two individuals in the ontology are connected via their object properties.

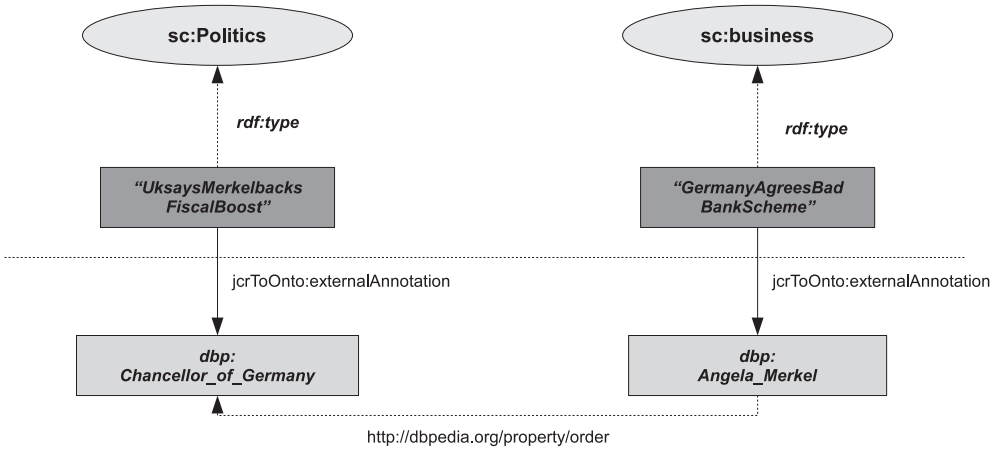

Figure 5.2: Utilization of external ontologies in search

Finally, the real value of the hybrid search solution is in its support for faceted search. As you would recall, faceted search enables users to navigate a multi-dimensional information space by combining text search with a progressive narrowing of choices in each dimension [48].

Case 7: The initial set of results received from the search service may be too broad; consequently, they have to be narrowed down to those that are related to a selected set of ontology resources.

In our example, a query with the keyword “disease” will return far too many results. However, as showng on Figure 5.3, with a progressive selection of classes, the user narrows down the choices to only those categorized by “Cancer” and “NeurologicalDisease”.

The set of returned documents now becomes:

MysteryDiseaseKillsHomosexuals, **Score**: 60

CancerBrakeCouldHaltDisease, **Score**: 60

MotorNeuroneDiseaseGeneClue, **Score**: -140

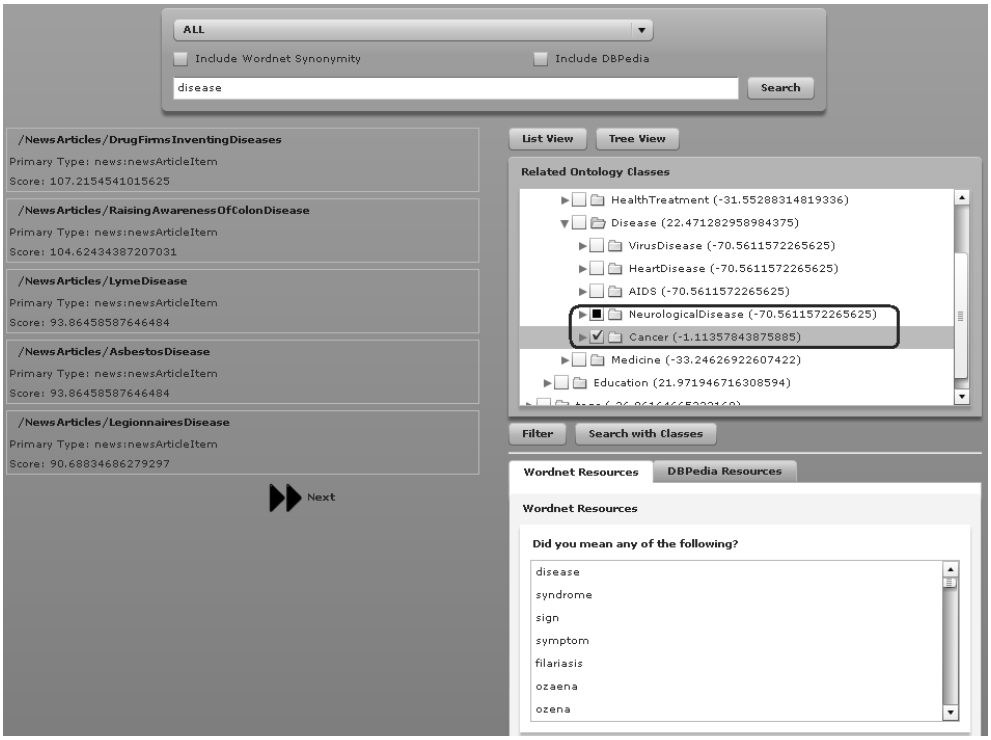

Figure 5.3: Faceted-search with a progressive selection of classes

Case 8: As opposed to Case 7, the initial set of keywords may be insufficient to retrieve all related documents; however, the suggested choices provide means to change the search scope such that omitted results are now included.

Suppose search is initiated with the keyword “viniculture”. As illustrated on Figure 5.4, only two documents: “PlantingBeginsOnANewVineyard” and “UKFarmsToastRecordWheatCrop” are returned. “PlantingBeginsOnANewVineyard” is categorized by the news subject code “Viniculture” whereas “UKFarmsToastRecordWheatCrop” by “ArableFarming”. However, the search service lists the names of the indirectly related classes, too.

The user explicitly excludes the two classes “NewsSubjectCodes”, “Viniculture” and “ArableFarming” from the selection list as shown on Figure 5.4. “EconomyBusinessFinance” is included, whereas “Agriculture” is set to “not selective”. Furthermore, the keyword is changed from “viniculture” to “wine”. As explained in Section 4.3.4, this condition implies that first the EXLUDE operator will be applied on the classes “NewsSubjectCodes”, “Viniculture”,

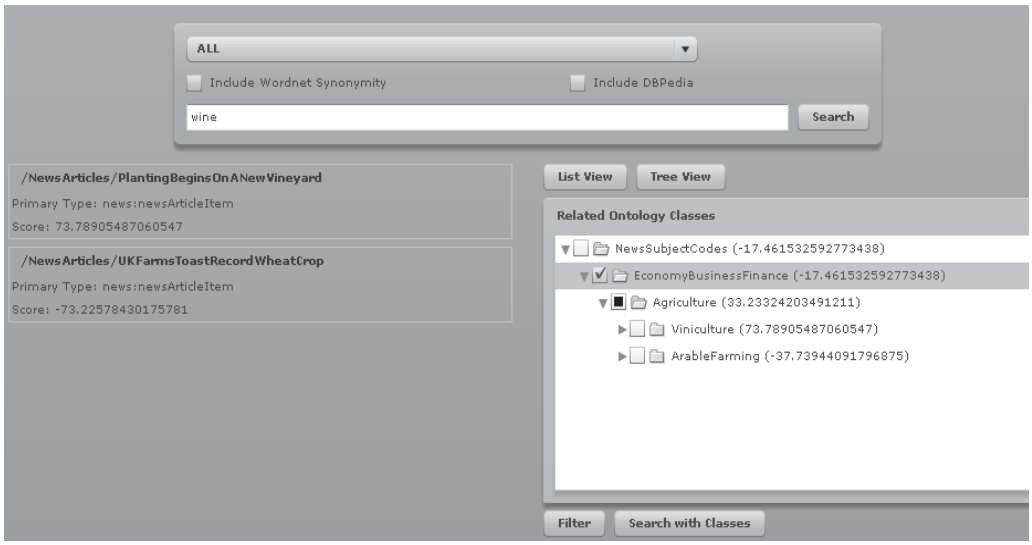

Figure 5.4: Faceted-search refined with new keywords and concepts

"ArableFarming"; therefore, results that belong to these categories will be pruned. Documents categorized by "Agriculture" will be returned; however, their scores will be kept low. Finally, articles that are categorized by "EconomyBusinessFinance" will be returned together with those whose text contains the term "wine". As a result of this iteration, the classes and the received synonyms will change. The process will continue until the user is satisfied with the results.

CHAPTER 6

RELATED WORK

In the preceding chapters, first an ontology extraction framework from JSR-170 compliant content repositories has been presented. Then, the implementation details of the hybrid search mechanism that works on the extracted ontologies have been thoroughly discussed. Finally, the value of the ontology extraction framework and the hybrid search mechanism combined has been evaluated from a content management system's point of view. Consequently, our discussion in this chapter will mainly revolve around three topics; that is, ontology extraction frameworks, enhancements over full-text search and finally the use of the semantic web technologies for content management.

6.1 Ontology Extraction from Relational Databases

Making the semantic relations implicit in the schema of a database available to other semantic web applications has been a popular research problem. In this respect, various papers have been published [64], [65], [66], [67], [68] and some prototype applications [93], [94] have been built.

Based on the context, the process described in the aforementioned works fall into either of the two categories:

1. Automatically or semi-automatically generating ontologies from the available entity-relationship model, the database schemas and the instances in the database,

2. Defining explicit mappings between the database schema and parts of an existing ontology.

Although these approaches can be exploited as they are; they, too, have their own limitations. The former approach is particularly valuable in its ability to automate the ontology extraction process. Some works [68] even utilize heuristic rules to increase the performance. However, the generated ontologies may be of poor practical significance to the user community if they are not properly enhanced with the domain knowledge. Nevertheless, the process provides an initial means for semantisizing the database schema and acts as a gap-bridge between existing database applications and the semantic web [68].

The latter approach eliminates the need for alignment with further domain knowledge as it starts working with domain ontologies already from the beginning. The user simply maps the views of the source database to terminologies in an existing target ontology. The obvious advantage over (1) is that in the end, an ontology whose T-Box assertions describe well the particular domain and whose A-Box is populated by the instances in the database can be obtained. On the other hand, it assumes that such domain ontologies will be readily available to work with. Furthermore, mappings are defined by hand, whereas in (1) the whole process is automated.

Below we provide a brief summary of some selected work:

VisAVis: "An Approach to an Intermediate Layer between Ontologies and Relational Database Contents"

The motivation behind the VisAVis approach comes from the need to make information that is locked in the relational databases on the web, which is often referred to as the deep web, be properly retrieved by web search engines [64]. The proposed solution relies on capturing any combination of datasets from the database and mapping them onto ontology classes. Consequently, the work falls into the latter category, that is; explicit mappings between the database and an existing ontology are defined. However, the authors argue against populating the ontologies by the data in the database. Therefore, the mapping is defined between the database schema and the T-Box of the ontology, whereas the A-Box contains only references to the database instances. In other words, every class in the original ontology is extended with a special data type property that stores the actual SQL query that returns the mapped dataset.

A prototype implementation; "VisAVisTab", is provided as a Protégé [29] plug-in. After opening the ontology that contains the classes of interest and establishing the database connection,

the graphical user interface lets the users select the datasets to be used in the mapping process. The procedure works seamlessly, that is; the user maps columns from possibly multiple tables in the database to ontology classes, while the corresponding SQL queries are generated in the background automatically. Intuitively, VisAVis treats each row selected by the SQL query as an individual of the mapped ontology class when queries are executed on the knowledge base. However, in that respect, the mapping process may still be considered coarse-grained because mapping to lower level constructs than ontology classes is not possible.

R_2O, an Extensible and Semantically Based Database-to-ontology Mapping Language

R_2O is a declarative language to map an existing database to an appropriate ontology implemented in RDF(S) or OWL [66]. The authors consider the work as an extension to some similar mapping approaches like D2R MAP [95] and D2R [96]. Since the work provides a mapping methodology rather than an ontology extraction framework, it falls into the same category as VisAVis. However, R_2O enables mappings to be defined in a greater depth: correspondences between both the components of the database schema (i.e. tables, columns, primary and foreign keys, etc.) and those of the ontology (concepts, relations, attributes, etc.) can be captured. Unlike VisAVis though, in R_2O, the ontologies are populated with instances from the database. In fact, ODEMapster - a reference mapping engine for R_2O - processes the mapping definitions to populate the ontology with the individuals.

As depicted on Figure 6.1, based on the different levels of overlap between the domain of the database and that of the ontology, three mapping cases have been identified by R_2O:

1. **A single database table maps to a single ontology class**: In this case, the columns of the table are mapped to the object or data type properties of the class. For each row in the table an individual is created where the property values are also set based on the aforementioned mapping.

2. **A single database table maps to multiple ontology classes**: Each column of the table is mapped to the object or data type properties of the same or different classes. However, for each table record, only a single individual per concept is generated.

3. **A single database table maps to multiple ontology classes, where multiple instances per concept may be generated**: Each column of the table is mapped to the object or data type properties of the same or different classes. For each table record, multiple

individuals per concept may be generated.

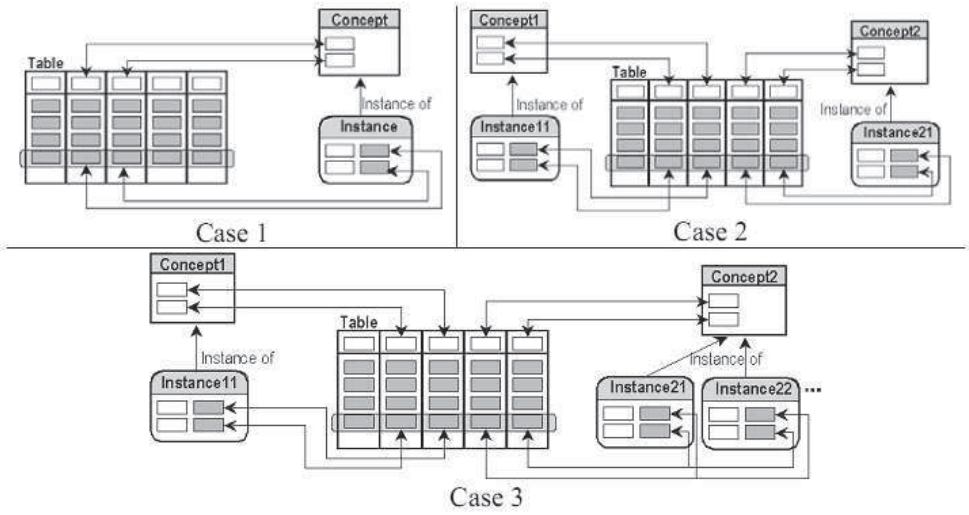

Figure 6.1: Mapping cases in R_2O [66]

The novelty of R_2O is in its strength to support mappings even in case of low similarity between the database and the target ontology. It is primarily due to the fact that different views can be generated on the database by applying any combination of joins, unions, projections and selections. Furthermore, the language allows conditions to be placed on the mapping definitions so as to control how and when the ontology population takes place. Finally, transformations can be defined that let the data to be modified before it is individualized.

Migrating data-intensive Web Sites into the Semantic Web

In their paper [65], Stojanovic et al. describe a methodology that uses the information in the relations, attributes, attribute types, primary and foreign keys and inclusion dependencies of a relational schema for ontology construction. Some predefined mapping rules are executed to construct ontology classes, establish the inheritance hierarchy among the generated classes and form the ontology properties. Finally, after the data migration process, the ontology is populated with instances from the database. The work falls into the former category, that is; semi-automatic generation of ontologies from the available database schema. As argued in [66], it is rather a lifting than a mapping process, therefore, situations that require fine-tuned adjustments will have to be dealt with manually at the “evaluation, validation and refinement of the generated ontology” step.

The RDBToOnto Tool

RDBToOnto [94] is an automatic ontology extraction framework that, like [65], exploits the structural implications of a database schema. On the other hand, RDBToOnto uses various mining techniques to identify patterns that reside particularly in the data but not in its schema [97], hence allowing more accurate ontologies to be derived. To be more specific, RDBToOnto allows categorization patterns in the database content to be automatically mined. In cases where the automatic categorization process fails, users are given the option to manually control the database attributes to be used in categorization.

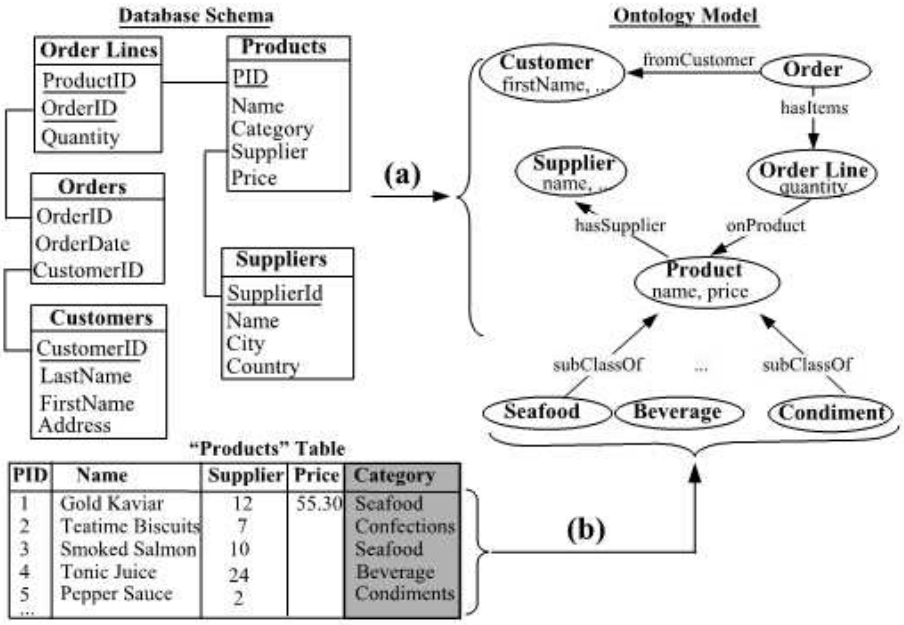

Figure 6.2: RDBToOnto uses various mining techniques for further classification [94]

As argued in [97], the fact that "Seafood", "Beverage" and "Condiment" are all subclasses of "Product" simply has to do with how frequently each item occurs in the actual dataset. As one may notice, this information is not available in the database schema; therefore, simply a flat ontology extraction environment that makes use of the schema alone would not have been able to detect such taxonomical implications.

Up to now, we have seen how the problem of ontology extraction from available database schemas has been dealt with in literature. The problem has been addressed mainly in two dimensions, that is; either mappings between the database schema and parts of an existing ontology are defined or the target ontology is semi-automatically generated from the schema alone. R_2O and VisAVis fall into the former category, whereas both Stojanovic et al.'s solution and the RDBToOnto framework fall into the latter. In R_2O, a complete mapping language is described that takes into account the different combinations of mapping schemes with conditional operators and transformations. R_2O populates the target ontology based on the mapping constructs provided as input. On the other hand, VisAVis chooses not to populate the ontology with actual instances but rather with references to the mapped datasets. Stojanovic et al.'s approach is considered as an automatic ontology extraction framework where the target ontology is constructed from scratch. Predefined heuristics help extract the semantic relations present in the database schema that are later on expressed as ontology building blocks. Finally, the knowledge engineer is given the ability to refine the generated ontology. The RDBToOnto framework takes it one step further and mines instances in the datasets in search for patterns that are not explicitly described by the schema.

To the best of our knowledge, the problem of ontology extraction from the JCR model has not been addressed before. However, the works proposed in the field of ontology extraction from relational database schemas have particularly been inspiring to our JCR-to-Onto Bridge framework. Even though content repositories are quite different in structural means from relational databases, the principal idea of using heuristics to capture the semantics in a content repository remains the same.

In that respect, JCR-to-Onto Bridge may be considered as a semi-automatic tool for extracting ontologies from JSR-170 compliant content repositories. As discussed in sections 2.6 and 3.4, though, the flexibility of a content repository model - as opposed to the rigid structure in a relational schema - makes it difficult to define in advance, a solid set of heuristics that are guaranteed to work with different repository instantiations. Therefore, JCR-to-Onto Bridge combines both of the aforementioned ontology extraction and mapping approaches and provides a scheme by which the user maps patterns in the repository to predefined ontology construction processes, along with a set of heuristics to extract ontological constructs from node type definitions. Since JCR-to-Onto Bridge employs a hybrid approach - where the extraction process can be custom-tailored by applying different mappings on the repository -

the resulting formalisms are more suitable for alignment with additional domain information. Now the user has full control over the produced ontology as opposed to the case where only some built-in heuristics are utilized. Finally, the mapping definitions enable content repository updates to be processed and be reflected on the generated ontologies, hence resolving possible synchronization problems.

6.2 Approaches Integrating Structural and Full-text Search

The idea of using structural queries in combination with full-text search to provide semantic search capabilities has been addressed by various works in literature: [38], [6], [39], [40] and [41]. Each work differs in the way it deals with the resources it uses for indexing, the indexing methodology, at which level (i.e. content, ontology, keyword) it merges the power of full-text and semantic search and finally the degree of complexity its supports through its structural queries. However, the process described in the aforementioned works broadly falls into any of the following three categories:

1. RDF literals and RDF annotations are indexed so that full-text search facilities may be built over structural queries,
2. An ontology look-up is performed to find semantically relevant terms to those in the documents and full-text indices are built on the expanded set of terms,
3. Given a set of keywords, the documents as well as their ontological annotations are searched through and presented to the user in a faceted fashion.

Although these approaches can be exploited as they are; they, too, have their limitations. The first approach enhances structural query languages with the power of full-text indexing. As argued in [39], structural query languages such as SPARQL [49] are not always powerful enough because search is simply based on the traversal of a directed graph where the RDF resources are represented by the graph's nodes and the RDF predicates by its edges. Furthermore, nodes and edges are matched based on either complete string matching techniques or by using regular expressions, which turns out to be a slow operation. Therefore, works such as [39] and [41] build full-text indices over the RDF graph and use them in answering the textually enhanced queries. This approach has the drawback that it simply assumes all

information will be available in the form of ontologies. On the other hand, in the context of a content management system application, it may be most desirable to store the metadata and only the reference to the content resource in the ontology. In that case, queries can only be run on the indexed metadata rather than the full textual content of the documents. Consequently, a document that contains the query term but has not explicitly been annotated with it will not be retrieved.

Works studied in the second category propose a different methodology in the level at which full-text and structural search is merged. Prior to executing textual queries, the frequent terms in the documents are looked-up from the ontologies and all relevant concepts, individuals and literals are indexed along with the terms in the document. Consequently, as illustrated in [6], a document that refers to "Tony Blair" will be retrieved when queried with the term "politician", provided that an ontology exists which relates the two terms. Without a doubt, this approach eliminates the drawbacks of a purely keyword-based search by bringing in relevant results that would otherwise have been omitted. On the other hand, support for arbitrary structural queries is still limited.

Similar to those in the second one, works that fall into the third category use terms in the documents as well as their ontological annotations in search. Contrary to [6] though, QuizRDF [40] enables the users to form some semantic queries, hence providing a more focused view of the search results. As outlined in [40], one limitation of the methodology proposed in QuizRDF is that queries can only be made around one class. Obviously, a far better solution would be to incorporate structural and semantic dependencies among concepts and properties in the ontologies when forming the queries.

Before going into the details of how these approaches have been complemented within the scope of this thesis, below we provide a brief summary of some selected work:

The Sesame LuceneSail: RDF Queries with Full-text Search

It is argued in [39], that structural search mechanisms lack support for full-text search. Some structural query languages like SPARQL [49] support simple string matching functions; however, they are observed to have negative effects on speed. Therefore, LuceneSAIL aims at integrating the full-text indexing capabilities into structural search without a loss in performance. The proposed methodology is built as an extension to the Sesame triplestore [10],

where Lucene [42] is used as the full-text indexer.

LuceneSAIL exploits “SPARQL extensions” so that native queries can be conveyed in two parts: textual and structural. When it receives the full query, LuceneSAIL separates the textual part from its structural complement. The textual query is executed on the virtual properties built over the literals in the graph. On the other hand, the structural query is executed on the whole triples. The results are merged in the final step.

Semantic Full-Text Search with ESTER: Scalable, Easy, Fast

In their paper titled “Semantic Full-text Search with ESTER: Scalable, Easy, Fast” [6], Bast et al. describe a search engine that combines full-text search with the powerful semantic capabilities of ontologies. As argued in [38], the novelty of the approach is in its speed and scalability compared to other works that implement such hybrid solutions. The reason is that for semantic search, instead of performing a direct ontology look-up, ESTER uses the semantically enhanced full-text indices. In other words, already at the indexing stage, the ontologies are woven into the document corpus and therefore the semantic relations are retrievable through keywords.

As described in [6], the relationships between the words in the documents and concepts in the ontology are established by the entity recognizer. The entity recognizer adds to the beginning of each document a set of annotations that describe the document content. For example, the following artificial words are added to an article about Tony Blair:

0 entity:tony blair

0 person:tony blair

1 is a:2

1 politician of:3

2 class:politician

3 country:united kingdom

Intuitively, these set of artificial words provide not only the annotations for some specific terms in the document but also the values of various object or data type properties and even

superclasses of the associated entities. Based on this information, ESTER is able to answer semantic queries such as “audience pope person:*” and “class:politician - is a - person:*” or even provide joins of both.

Even though such an approach scales well to large corpora, complex queries require all ontological facts and relations to be encoded in every single document for efficient processing. Consider now that the ontology contains further information about United Kingdom; e.g. that it is located in Europe. Now, a query that requests all documents related to the politicians in Europe cannot be answered unless multiple joins are performed. First of all, the entities that are related to Europe with an object property named “located in” should be retrieved. Later on the query has to be repeated for all such entities and the results have to be joined. In case of multiple joins, there is no clear advantage of representing the ontological annotations as artificial words since the same behavior can be achieved through ontology traversal.

QuizRDF: Search Technology for the Semantic Web

The motivation behind QuizRDF [40] is that only a very small proportion of the WWW resources are annotated semantically and therefore it is preferable to preserve the power of traditional free text search engines while providing search facilities that exploit the power of ontologies.

In QuizRDF, both the content resources and their RDF annotations are indexed. The indexing mechanism works as follows: First, the content descriptors are retrieved. QuizRDF obtains the content descriptors either by performing a full text analysis on the content or by processing the annotations that are directly related to the resource through a datatype or object property as shown in Figure 6.3. Then the ontological index is created, that is; for each content descriptor, the ontology class it belongs to, the names of its properties and the RDF resource itself is indexed as shown in Figure 6.4.

QuizRDF assembles keyword-based search and semantic browsing of content in a single user interface. On start-up, a text box and a drop-down menu is presented. The user enters arbitrary keywords into the text box. QuizRDF returns a list of RDF resources ranked based on their relevance to the query. The ranking mechanism is based on a variation of the well-known “tf.idf” vector product scheme [44]. While presenting the ranked resources to the user, QuizRDF also computes the classes associated with each resource and updates the drop-down

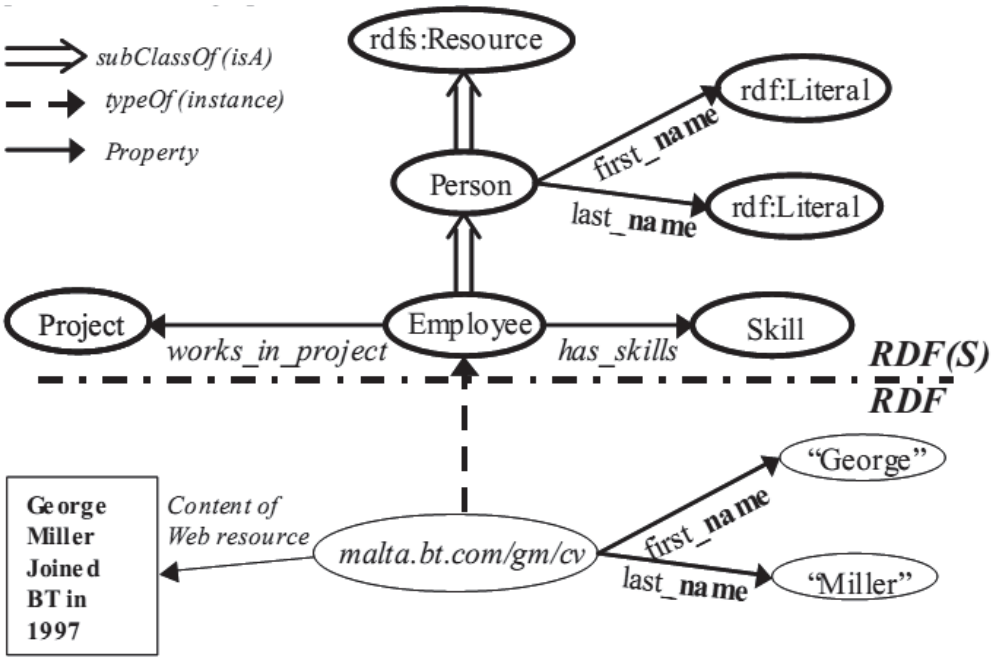

Figure 6.3: Ontology-based indexing in QuizRDF [40]

Descriptor	Class	Property	Resource
Miller	Employee	Ø	malta.bt.com/gm/cv
joined	Employee	Ø	malta.bt.com/gm/cv
BT	Employee	Ø	malta.bt.com/gm/cv
1990	Employee	Ø	malta.bt.com/gm/cv
George	Employee	first_name	malta.bt.com/gm/cv
Miller	Employee	last_name	malta.bt.com/gm/cv

Figure 6.4: Indices created on content descriptors [40]

list. When the user selects a certain class from the drop-down list, the results that are not instances of the selected class are filtered out. Furthermore, QuizRDF allows users to narrow down the search scope by providing values for the properties of the displayed classes. However, queries around multiple classes that require unions are currently not supported. The authors have identified it as future work.

Even though the hybrid search approach presented in this thesis complement works described in [38], [6], [39], [40] and [41]; there are some differences. First of all, [40] assumes that the

content resources reside on the RDF graph and therefore are indexed together with their annotations. However, to avoid overcrowded ontologies that no reasoner is able to process, it is not our intention neither with JCR-to-Onto Bridge nor an alternative ontology extraction scheme to populate the domain ontology with the document content. We keep the metadata in the domain ontologies, while the documents reside in the content repository. Consequently, our hybrid search mechanism uses an external full-text search engine and works with the domain ontologies instead. The results from the ontological and the full-text search components are merged at a latter step. In this respect, our approach is different from [6], too, in that we are still able to exploit the reasoning power of the knowledge base while answering the queries. As for the concept driven retrieval of results presented earlier in Section 4.3, we strictly exploit the power provided by the extended structural and semantic queries discussed in [39] and [41]. Finally, we build our faceted-search interface in a similar way to [40], where search scope can iteratively be narrowed or extended.

6.3 Semantic Web Applications in the News Domain

The use of the semantic web technologies in the news domain has been addressed by various papers [98], [99] and research projects [100]. The domain is suitable for such experimentation as the semantic web technologies can facilitate the management of large volumes of news documents [98]. Even though the works discussed in this section are domain-specific, the proposed methodology is applicable to other domains, hence; are of relevance to this thesis.

The works addressed herein attack the problem by: (i) developing domain ontologies that enable semantic services to be built, (ii) implementing annotation and information retrieval tools that use natural language processing or statistical methods to extract metadata from news articles and finally, (iii) building semantic features on top of existing search paradigms. Below we discuss the novelty of each approach.

An experience with Semantic Web technologies in the news domain

In their paper titled “An experience with Semantic Web technologies in the news domain” [98], Fernandez et al. describe the motivation and the proposed methodology behind the NEWS project [100]. NEWS is an information society technologies project funded by the European Commission in the sixth framework programme. As stated in [98], news agencies

produce content in the form of news items describing an event. Most of this content is text, but they also produce multimedia content in different human languages. In this regard, managing all this heterogeneous information in an efficient manner becomes problematic. Fernandez et al. propose a framework that automatically annotates content with the concepts in a domain ontology. The ontology is engineered beforehand, and it covers the main concepts required in the news domain. It is a lightweight RDFS [63] ontology and provides the basic classes, properties and instances for news item categorization and content annotation. The ontology resources have labels in multiple languages, too.

Bringing the IPTC News Architecture into the Semantic Web

It is argued in [99] that the abundance of different metadata formats in the news production process leads to interoperability problems. Therefore, the paper discusses how an OWL [13] ontology for the IPTC News Architecture [72] is designed and linked with other multimedia metadata standards such as EXIF [101], DIG35 [102] and XMP [103]. Initially, an OWL ontology has been engineered from the IPTC News Architecture framework. It is argued in the paper that some preprocessing (i.e. flattening of the XML structure, reification of statements, code resolution, etc.), was necessary. The IPTC News Subject codes taxonomy [71] has also been OWLized. The links with the other multimedia metadata standards have been established through asserting class or property equivalences. Finally, the news documents are annotated with these ontologies.

The correctness of the engineered ontology infrastructure has been evaluated through a semantic search and browsing interface. A rather interesting example demonstrated in the paper is that when searched with “Lyon”, a photo is retrieved whose annotations do not explicitly contain the search term. In fact, the caption of the photo mentions Juninho Pernambucano, recognized by SPROUT as a football player, who is later on identified through dbPedia as a member of the soccer club “Lyon”.

As you would recall, the motivation behind the JCR-to-Onto Bridge framework is similar to that of the works described in this section, that is; to facilitate the use of semantic web technologies. However, it is different in the sense that the scope is not restricted to the news domain but generalized for all types of content management systems. In this respect, JCR-to-Onto Bridge provides the semi-automated means for extracting ontologies from JSR-170 compliant content repositories, whereas the domain ontologies in [98], [99] and [100] are

engineered. Furthermore, to demonstrate the value of the JCR-to-Onto Bridge approach, we have developed a hybrid search methodology that combines the power of structural and semantic search on ontologies with that of the full-text search on documents. The proposed solution also uses external ontology lookup (i.e. WordNet [80] and dbPedia [81]), however, complements [99] with its support for structural and semantic queries.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Our discussion in this thesis revolved around an ontology extraction framework, namely the JCR-to-Onto Bridge, and a semantic search mechanism built on top of it. In Chapter 3, we outlined the various issues related to the ontology extraction process, the content modeling patterns we have exploited and finally the mapping schemes. In the following chapter, namely Chapter 4, the algorithmic details of our hybrid search mechanism were presented. Finally, the power of the JCR-to-Onto Bridge framework was demonstrated in Chapter 5 through the value-added semantic search features of the proposed hybrid search engine.

The motivation behind the JCR-to-Onto Bridge framework has been to facilitate the use of semantic technologies within the context of a content management system. JCR-to-Onto Bridge, which may be considered as a semi-automatic tool for extracting ontologies from JSR-170 compliant content repositories, frees the semantics that would otherwise be locked up in the repository. The framework combines both ontology extraction and mapping approaches; hence, it works around a set of built-in heuristics and at the same time provides the freedom to map patterns in the repository to different construction processes.

The semantic search engine built on top of the JCR-to-Onto Bridge framework combines the power of structural and semantic search on ontologies with that of the full-text search on documents. It uses an external full-text search engine and works with the domain ontologies instead. Our approach is different from [6], in that we are still able to exploit the reasoning power of the knowledge base while answering the queries. The hybrid search mechanism complements QuizRDF [40] by providing more complex faceted-search behavior and support for extended queries. Finally, the service can be invoked anytime, anywhere through various REST operations.

The work produced in this thesis is merely an initial attempt towards making the content management lifecycle semantically enabled if the whole spectrum of features is taken into consideration. First of all, the mapping scheme of the JCR-to-Onto Bridge framework currently does not have support for transformations; that is, data in the repository is directly translated into the ontology attributes. Furthermore, updating the knowledge base when the content repository grows is left as a future challenge. However, we argue that the queries used in the mappings will serve as templates to match against whenever a new event is fired from the content repository. As outlined in Section 4.4, the hybrid search mechanism may not be suitable for large ontologies, either. In fact, further optimizations such as “processing in chunks”, “first n results retrieval”, “T-Box traversal” and inevitably building appropriate indices on the ontology resources may be necessary.

REFERENCES

[1] S. McKeever, "Understanding Web content management systems: evolution, lifecycle and market," *Industrial Management & Data Systems*, vol. 103, no. 9, pp. 686-692, 2003.

[2] "Effective Web Content Management: Empowering the Business User While IT Maintains Control," Ektron, Inc., Amherst, NH, USA, Tech. Rep., 2001.

[3] J. Alpert, and N. Hajaj, "We knew the web was big...," *Google Inc.*, 2008. [Online]. Available: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html. [Accessed: Jun. 6, 2009].

[4] D. M. Le, and L. Lau, "An Open Architecture for Ontology-Enabled Content Management Systems: A Case Study in Managing Learning Objects," in *On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE*, Vol. 4275/2006, pp. 772-790, 2006.

[5] N. Guarino, R. Poli, Kluwer Academic Publishers, In Press Substantial, and T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *in Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, in press. Substantial revision of paper presented at the International Workshop on Formal Ontology*, 1993.

[6] H. Bast, F. M. Suchanek, and I. Weber, "Semantic Full-Text Search with ESTER: Scalable, Easy, Fast," *in Proceedings of ICDM Workshops*, 2008, pp. 959-962.

[7] D. Beckett, and B. McBride, "RDF/XML Syntax Specification (Revised)," *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar/. [Accessed: Jun. 6, 2009].

[8] "World Wide Web Consortium," *W3C*, 2009. [Online]. Available: http://www.w3.org/. [Accessed: Jun. 6, 2009].

[9] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," *The Internet Engineering Task Force*, 1998. [Online]. Available: http://tools.ietf.org/html/rfc2396. [Accessed: Jun. 6, 2009].

[10] J. Broekstra, A. Kampman, and F. v. Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," in *Proceedings of the First International Semantic Web Conference Sardinia*, 2002, pp. 54-68.

[11] F. Manola, E. Miller, and B. McBride, "RDF Primer," *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/rdf-primer/#rdfmodel. [Accessed: Jun. 6, 2009].

[12] D. Brickley, R.V. Guha, and B. McBride, "RDF Vocabulary Description Language 1.0: RDF Schema," *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/rdf-schema/. [Accessed: Jun. 6, 2009].

[13] D. L. McGuinness, and F. v. Harmelen, “OWL Web Ontology Language,” *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/owl-features/. [Accessed: Jun. 6, 2009].

[14] M. Pagels, “DAML: The DARPA Agent Markup Language Homepage,” *Defense Advanced Projects Agency*, 2006. [Online]. Available: http://www.daml.org/. [Accessed: Jun. 6, 2009].

[15] J. Heflin, “OWL Web Ontology Language: Use Cases and Requirements,” *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/webont-req/#onto-def. [Accessed: Jun. 6, 2009].

[16] L. Quin, “Extensible Markup Language (XML),” *W3C*, 2009. [Online]. Available: http://www.w3.org/XML/. [Accessed: Jun. 6, 2009].

[17] “openRDF.org,” 2009. [Online]. Available: http://www.openrdf.org/. [Accessed: Jun. 6, 2009].

[18] “The Sesame library,” *openRDF.org*. [Online]. Available: http://www.openrdf.org/doc-/sesame/users/ch01.html#d0e69. [Accessed: Jun. 6, 2009].

[19] J. Bock, P. Haase, Q. Ji, and R. Volz, “Benchmarking owl reasoners,” in *ARea2008 - Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, 2008.

[20] “User Guide for Sesame: An Overview of the Sesame Architecture,” *openRDF.org*, [Online]. Available: http://www.openrdf.org/doc/sesame/users/userguide.html#d0e129. [Accessed: Jun. 6, 2009].

[21] G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki, “Querying Community Web Portals,” Institute of Computer Science, FORTH, Heraklion, Greece, Tech. Rep., 2000.

[22] “The SeRQL query language (revision 1.2),” *openRDF.org*. [Online]. Available: http://-www.openrdf.org/doc/sesame/users/ch06.html. [Accessed: Jun. 6, 2009].

[23] A. Kiryakov, D. Ognyanov, and D. Manov, “OWLIM - A Pragmatic Semantic Repository for OWL,” in *Proceedings of Web Information Systems Engineering 2005 International Workshops*, 2005, Vol. 3807, pp. 182-192.

[24] “User Guide for Sesame: Repositories and Inferencing,” *openRDF.org*, [Online]. Available: http://www.openrdf.org/doc/sesame/users/userguide.html#d0e115. [Accessed: Jun. 6, 2009].

[25] “User Guide for Sesame: Custom inferencing,” *openRDF.org*, [Online]. Available: http://www.openrdf.org/doc/sesame/users/userguide.html#d0e803. [Accessed: Jun. 6, 2009].

[26] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, “Description logic programs: combining logic programs with description logic,” in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 48-57.

[27] “Jena - A Semantic Web Framework for Java,” *sourceforge.net*. [Online]. Available: http://jena.sourceforge.net/. [Accessed: Jun. 6, 2009].

[28] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds, “Efficient RDF Storage and Retrieval in Jena2,” in *Proceedings of the 1st International Workshop on Semantic Web and Databases*, 2003, pp. 131-151.

[29] Stanford Center for Biomedical Informatics Research, “Protégé,” *Stanford University*, 2009. [Online]. Available: http://protege.stanford.edu/. [Accessed: Jun. 6, 2009].

[30] “SWOOP: Semantic Web Ontology Editor,” *Google Code*, 2009. [Online]. Available: http://code.google.com/p/swoop/. [Accessed: Jun. 6, 2009].

[31] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, “The protégé owl plugin: An open development environment for semantic web applications,” in *Proceedings of Semantic Web - ISWC*, 2004, pp. 229-243.

[32] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, “A practical guide to building OWL ontologies using the Protege-OWL plugin and co-ode tools edition 1.0,” August 2004. [Online]. Available: http://www.co-ode.org/resources/tutorials-/ProtegeOWLTutorial.pdf. [Accessed: Jun. 6, 2009].

[33] “KAON2,” 2006. [Online]. Available: http://kaon2.semanticweb.org/. [Accessed: Jun. 6, 2009].

[34] Information Systems Group, “Hermit OWL Reasoner.” [Online]. Available: http://-www.hermit-reasoner.com/. [Accessed: Jun. 6, 2009].

[35] V. Haarslev, R. Moller, and M. Wessel, “Querying the semantic web with racer + nrql,” in *Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL‘04)*, 2004.

[36] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, “Pellet: A practical OWL-DL reasoner,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51-53, June 2007.

[37] “The new DIG interface standard (DIG 2.0),” 2006. [Online]. Available: http://dl.kr.org-/dig/interface.html. [Accessed: Jun. 6, 2009].

[38] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber, “Ester: efficient search on text, entities, and relations,” in *Proceedings of SIGIR*, 2007, pp. 671-678.

[39] E. Minack, L. Sauermann, G. Grimnes, C. Fluit, and J. Broekstra, “The Sesame Lucene Sail: RDF Queries with Full-text Search,” NEPOMUK Consortium, Tech. Rep. 2008-1, 2008.

[40] J. Davies, and R. Weeks, “QuizRDF: search technology for the semantic Web,” in *the 37th Annual Hawaii International Conference on System Sciences* , 2004.

[41] “LARQ - Free Text Indexing for SPARQL,” *sourceforge.net*. [Online]. Available: http://-jena.sourceforge.net/ARQ/lucene-arq.html. [Accessed: Jun. 6, 2009].

[42] The Apache Software Foundation, “Apache Lucene,” 2009. [Online]. Available: http://-lucene.apache.org/java/docs/. [Accessed: Jun. 6, 2009].

[43] E. Hatcher, and O. Gospodnetic, *Lucene in Action* (In Action series). Manning Publications, December 2004.

[44] G. Salton, “Developments in automatic text retrieval,” *Science*, vol. 253, pp. 974-979, 1991.

[45] The Apache Software Foundation, “Apache Solr,” 2009. [Online]. Available: http://-lucene.apache.org/solr/. [Accessed: Jun. 6, 2009].

[46] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. dissertation, University of California, Irvine, 2000.

[47] R. T. Fielding et al., “Hypertext Transfer Protocol - HTTP/1.1: Method Definitions,” *W3C*. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. [Accessed: Jun. 6, 2009].

[48] “SIGIR’2006 Workshop on Faceted Search: Call for Participation,” Aug. 10, 2006. [Online]. Available: http://facetedsearch.googlepages.com/. [Accessed: Jun. 6, 2009].

[49] E. Prud’hommeaux, and A. Seaborne, “SPARQL Query Language for RDF,” *W3C*, 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/. [Accessed: Jun. 6, 2009].

[50] J. Perez, M. Arenas, and C. Gutierrez, “Semantics and Complexity of SPARQL,” in *Proceedings of the 5th International Semantic Web Conference*, 2006, Vol. 4273/2006, pp. 30-43.

[51] The Apache Software Foundation, “Apache Jackrabbit,” 2009. [Online]. Available: http://jackrabbit.apache.org/. [Accessed: Jun. 6, 2009].

[52] B. Chapuis, “JCR or RDBMS? Why, when, how?,” *Day Software AG*, 2008. [Online]. Available: http://dev.day.com/microsling/content/blogs/main/jcrrdbmsreport.html. [Accessed: Jun. 6, 2009].

[53] S. Patil, “What is Java Content Repository,” *O’Reilly Media, Inc.*, 2006. [Online]. Available: http://www.onjava.com/pub/a/onjava/2006/10/04/what-is-java-content-repository.html. [Accessed: Jun. 6, 2009].

[54] D. Nuescheler, “JSR 170: Content Repository for Java technology API,” *Java Community Process*, 2006. [Online]. Available: http://jcp.org/en/jsr/detail?id=170. [Accessed: Jun. 6, 2009].

[55] D. Nuescheler et al., “Content Repository API for Java Technology Specification,” Day Software AG, Tech. Rep., 2004.

[56] “CRX 1.4.1 Setup Guide,” Day Software AG, Tech. Rep., 2009.

[57] J. Clark, and S. DeRose, “XML Path Language (XPath) Version 1.0,” *W3C*, 1999. [Online]. Available: http://www.w3.org/TR/xpath. [Accessed: Jun. 6, 2009].

[58] “Information technology - Database languages - SQL,” *International Organization for Standardization*, 1999. [Online]. Available: http://www.iso.org/iso/iso_catalogue-/catalogue_tc/catalogue_detail.htm?csnumber=26196. [Accessed: Jun. 6, 2009].

[59] D. Nuescheler, “What is JCR/JSR-170/JSR-283?.” [Online]. Available: http://-www.slideshare.net/uncled/introduction-to-jcr. [Accessed: Jun. 6, 2009].

[60] Sun Microsystems, Inc., “Java Transaction API (JTA),” 2001. [Online]. Availabe: http://-java.sun.com/javaee/technologies/jta/. [Accessed: Jun. 6, 2009].

[61] D. Nuescheler, “JSR 283: Content Repository for JavaTM Technology API Version 2.0,” *Java Community Process*, 2009. [Online]. Available: http://jcp.org/en/jsr/detail?id=283. [Accessed: Jun. 6, 2009].

[62] D. Choy, E. Guresh, A. Brown, and M. McRae, “OASIS Content Management Interoperability Services (CMIS) TC,” *OASIS*, 2008. [Online]. Available: http://-www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis#technical. [Accessed: Jun. 6, 2009].

[63] D. Choy, and E. Guresh, “Content Management Interoperability Services - Domain Model Version 0.62a,” *OASIS CMIS TC*, Tech. Rep. draft, 2009. [Online]. Available: http://www.oasis-open.org/committees/download.php/32774/CMIS%20Part%20I%20–%20Domain%20Model%20v0.62a%20with%20ACLs.doc. [Accessed: Jun. 6, 2009].

[64] N. Konstantinou, D. E. Spanos, M. Chalas, E. Solidakis, and N. Mitrou, “VisAVis: An Approach to an Intermediate Layer between Ontologies and Relational Database Contents,” in *Proceedings of International CAiSE Workshop on Web Information Systems Modeling (WISM)*, 2006, pp. 1050-1061.

[65] L.Stojanovic, N. Stojanovic, and R. Volz, “Migrating data-intensive Web Sites into the Semantic Web,” in *Proceedings of the 2002 ACM symposium on Applied computing*, 2002, pp. 1100 - 1107.

[66] J. Barrasa, O. Corcho, and A. Gómez-Pérez, “R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language,” in Bussler, C., Tannen, V., Fundulaki, I. (eds.) SWDB, 2004, vol. 3372.

[67] N. Cullot, R. Ghawi, and K. Yétongnon, “DB2OWL: A Tool for Automatic Database-to-Ontology Mapping,” in *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD*, 2007, pp. 491-494.

[68] Z. Xu, S. Zhang, and Y. Dong, “Mapping between Relational Database Schema and OWL Ontology for Deep Annotation,” in *Proceedings of the International Conference on Web Intelligence*, 2006, pp. 548-552.

[69] “Apache Jackrabbit - Node Type Notation,” *The Apache Software Foundation*. [Online]. Available: http://jackrabbit.apache.org/node-type-notation.html. [Accessed: Jun. 6, 2009].

[70] S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel, and L. A. Stein, “OWL Web Ontology Language Reference: Enumerated datatype,” *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/owl-ref/#EnumeratedDatatype. [Accessed: Jun. 6, 2009].

[71] “IPTC News Subject Codes,” *IPTC*, 2008. [Online]. Available: http://cv.iptc.org-/newscodes/subjectcode/. [Accessed: Jun. 6, 2009].

[72] “The International Press Telecommunications Council,” *IPTC*, 2009. [Online]. Available: http://www.iptc.org/cms/site/index.html?channel=CH0086. [Accessed: Jun. 6, 2009].

[73] “NewsCodes: Metadata Taxonomies for the News Industry,” *IPTC*. [Online]. Available: http://www.iptc.org/cms/site/index.html;jsessionid=-a00DKRfpdvW_?channel=CH0088. [Accessed: Jun. 6, 2009].

[74] “NewsCodes: View any of them...,” *IPTC*. [Online]. Available: http://www.iptc.org-/cms/site/index.html;jsessionid=a00DKRfpdvW_?channel=CH0103. [Accessed: Jun. 6, 2009].

[75] Y. Kalfoglou, and M. Schorlemmer, “Ontology mapping: the state of the art,” *The Knowledge Engineering Review*, vol. 18, no. 1, pp. 1-31, 2003.

[76] “flickr,” *Yahoo! Inc.*, 2009. [Online]. Available: http://www.flickr.com/. [Accessed: Jun. 6, 2009].

[77] Y. Hassan-Montero, and V. Herrero-Solana, “Improving Tag-Clouds as Visual Information Retrieval Interfaces,” in *InScit2006: International Conference on Multidisciplinary Information Sciences and Technologies*, 2006.

[78] “Wordnet in RDFS and OWL,” *W3C*, 2004. [Online]. Available: http://www.w3.org-/2001/sw/BestPractices/WNET/wordnet-sw-20040713.html. [Accessed: Jun. 6, 2009].

[79] A. Dogac, G. B. Laleci, G. Aluc, A. A. Sinaci, W. Behrendt, B. Delacretaz, and J. M. Pittet, “A semantically Enriched Persistence Mechanism for Interactive Knowledge Stack”, Accepted for *eChallanges Conference*, October 2009.

[80] Cognitive Science Laboratory, Princeton University, “WordNet: a lexical database for the English language,” 2006. [Online]. Available: http://wordnet.princeton.edu/. [Accessed: Jun. 6, 2009].

[81] “DBpedia,” 2009. [Online]. Available: http://dbpedia.org/. [Accessed: Jun. 6, 2009].

[82] M. Ehrig, and Y. Sure. “foam: Framework for Ontology Alignment and Mapping,” *Institut AIFB, Universität Karlsruhe*, 2005. [Online]. Available: http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/. [Accessed: Jun. 6, 2009].

[83] C. M. Sperberg-McQueen, and H. Thompson, “XML Schema,” *W3C*, 2000. [Online]. Available: http://www.w3.org/XML/Schema. [Accessed: Jun. 6, 2009].

[84] Y. Guo, Z. Pan, and J. Heflin, “Lubm: A benchmark for owl knowledge base systems,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158-182.

[85] “MySQL 5.1 Reference Manual,” *Sun Microsystems, Inc.*, 2009. [Online]. Available: http://dev.mysql.com/doc/refman/5.1/en/. [Accessed: Jun. 6, 2009].

[86] “Semantic CMS Reasoning and Data Persistence Components,” SRDC Ltd., Tech. Rep., 2009.

[87] “Description of Work: Interactive Knowledge Stack for small to medium CMS/KMS providers (IKS),” Salzburg Research Forschungsgesellschaft m.b.H. et al., Tech. Rep., Seventh Framework Programme, ICT-2007-4.4, Intelligent Content and Semantics, 2008.

[88] B. Delacretaz, "CQ5 Content Models: the Tags," *Day Software AG*, 2009. [Online]. Available: http://dev.day.com/microsling/content/blogs/main/cq5tags.html. [Accessed: Jun. 6, 2009].

[89] D. Raggett, A. L. Hors, and I. Jacobs, "HTML 4.01 Specification," *W3C*, 1999. [Online]. Available: http://www.w3.org/TR/html401/. [Accessed: Jun. 6, 2009].

[90] J. S. Nelson, D. Johnston, and B. L. Humphreys, "Relationships in Medical Subject Headings," in *Relationships in the Organization of Knowledge*, pp.171-184, A. C. Bean, and R. Green, Ed. New York: Kluwer Academic Publishers; 2001.

[91] J. Farrell, and G. Nezlek, "Rich Internet Applications The Next Stage of Application Development," in *29th Int. Conference on Information Technology Interfaces*, 2007, pp. 413-418.

[92] "Adobe Flex 3," *Adobe Systems Inc.* [Online]. Available: http://www.adobe.com-/products/flex/?promoid=DINEZ. [Accessed: Jun. 6, 2009].

[93] F. Cerbah, "RDBToOnto User Guide, Version 1.2 beta, beta From Relational Databases to Fine-Tuned Populated Ontologies," TAO/2008/D7.2a1/v1.2, 2009.

[94] F. Cerbah, "Learning highly structured semantic repositories from relational databases," in *Proceedings of ESWC*, Vol. 5021 of LNCS, pp. 777-781, 2008.

[95] C. Bizer, "D2R MAP - A DB to RDF Mapping Language," in *12th International World Wide Web Conference*, May 2003.

[96] J. Barrasa, O. Corcho, and A. Gómez-Pérez, "Fund Finder: A case study of database-to-ontology mapping," in *ISWC2003 Workshop on Semantic Integration*, Sanibel Island, Florida, 2003.

[97] F. Cerbah, "Mining the Content of Relational Databases to Learn Ontologies with Deeper Taxonomies," in *Proceedings of ACM International Conference on Web Intelligence*, 2008, pp. 553-557.

[98] N. Fernandez et al., "NEWS: Bringing Semantic Web Technologies into News Agencies," in *Proceedings of the 5th International Semantic Web Conference*, ISWC 2006, pp. 778-791.

[99] R. Troncy, "Bringing the IPTC News Architecture into the Semantic Web," in *Proceedings of the 7th International Conference on The Semantic Web*, 2008, pp. 483-498.

[100] "NEWS: News Engine Web Services," *DFKI GmbH - Ansgar Bernardi*, 2006. [Online]. Available: http://www.dfki.uni-kl.de/ bernardi/News/. [Accessed: Jun. 6, 2009].

[101] Technical Standardization Committee on AV & IT Storage Systems and Equipment, "Exchangeable image file format for digital still cameras: Exif Version 2.2," *Japan Electronics and Information Technology Industries Association*, 2002. [Online]. Available: http://www.digicamsoft.com/exif22/exif22/html/exif22.htm. [Accessed: Jun. 6, 2009].

[102] "DIG35 Initiative Group," *International Imaging Industry Association*, 1999. [Online]. Available: http://www.i3a.org/technologies/metadata/. [Accessed: Jun. 6, 2009].

[103] "Extensible Metadata Platform (XMP)," *Adobe Systems Inc.* [Online]. Available: http://www.adobe.com/products/xmp/. [Accessed: Jun. 6, 2009].

Appendix A

SAMPLE OWL CONSTRUCTS EXTRACTED FROM NODE TYPE DEFINITIONS

```
Object Properties:

  <owl:ObjectProperty rdf:about="#hasImages">
    <rdfs:domain rdf:resource="#CulturalHeritageItem"/>
    <rdfs:range rdf:resource="#Images"/>
  </owl:ObjectProperty>

Data Properties:

    ...
    ...

  <owl:DatatypeProperty rdf:about="#location">
    <rdfs:domain rdf:resource="#CulturalHeritageItem"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#popularity">
    <rdfs:domain rdf:resource="#CulturalHeritageItem"/>
    <rdfs:range>
      <rdf:Description>
        <rdf:type rdf:resource="&owl;DataRange"/>
        <owl:oneOf>
           <rdf:Description>
            <rdf:type rdf:resource="&rdf;List"/>
            <rdf:first rdf:datatype="&xsd;string">Not Popular</rdf:first>
            <rdf:rest>
              <rdf:Description>
                <rdf:type rdf:resource="&rdf;List"/>
                <rdf:first rdf:datatype="&xsd;string">Popular</rdf:first>
                <rdf:rest>
                  <rdf:Description>
```

```
                    <rdf:type rdf:resource="&rdf;List"/>
                    <rdf:first rdf:datatype="&xsd;string">Very Popular</rdf:first>
                    <rdf:rest rdf:resource="&rdf;nil"/>
                  </rdf:Description>
                </rdf:rest>
              </rdf:Description>
            </rdf:rest>
          </rdf:Description>
        </owl:oneOf>
      </rdf:Description>
    </rdfs:range>
  </owl:DatatypeProperty>

    ...
    ...

Classes:

  <owl:Class rdf:about="#AncientStructureAndBuilding">
    <rdfs:subClassOf rdf:resource="#CulturalHeritageItem"/>
  </owl:Class>

  <owl:Class rdf:about="#CulturalHeritageItem">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>

  <owl:Class rdf:about="#Images">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>

  <owl:Class rdf:about="#Monument">
    <rdfs:subClassOf rdf:resource="#CulturalHeritageItem"/>
  </owl:Class>

  <owl:Class rdf:about="&owl;Thing"/>
```

Appendix B

FULL XML SCHEMA DECLARATION OF THE SEARCH INTERFACE

```
<xsd:schema
  targetNamespace="search.model.rest.persistence.iks.srdc.com.tr"
  attributeFormDefault="qualified"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="search.model.rest.persistence.iks.srdc.com.tr">

  <xsd:element name="Query">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:KeywordList" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="tns:StructuralQueryPart" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="tns:FullTextSearchResultList" minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
      <xsd:attribute name="useSynonymsInOntologyLookup" type="xsd:boolean" use="required"/>
      <xsd:attribute name="usedbPediaForFindingSimilarContent" type="xsd:boolean" use="required"/>
      <xsd:attribute name="flexibilityClasses" type="xsd:integer" use="optional" />
      <xsd:attribute name="flexibilityIndividuals" type="xsd:integer" use="optional" />
      <xsd:attribute name="maxResults" type="xsd:integer" use="optional" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Result">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:ReturnedDocuments" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="tns:ReturnedOntologyResources" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="tns:TopRelatedOntologyResources" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="tns:ReturnedWordnetResources" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="tns:ReturnedDBPediaResources" minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
```

```
  </xsd:complexType>
</xsd:element>

<xsd:element name="TopRelatedOntologyResources">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:ClassResource" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ReturnedDBPediaResources">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:DBPediaResource" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="DBPediaResource">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Property" type="tns:non_empty_string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Object" type="tns:non_empty_string" minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ReturnedWordnetResources">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:WordnetResource" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="WordnetResource">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="tns:non_empty_string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Score" type="xsd:float" minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ReturnedDocuments">
  <xsd:complexType>
```

```
    <xsd:sequence>
      <xsd:element ref="tns:Document" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Document">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="DocumentXPath" type="tns:non_empty_string" minOccurs="1"
        maxOccurs="1" />
      <xsd:element name="PrimaryType" type="tns:non_empty_string" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="Score" type="xsd:float" minOccurs="1" maxOccurs="1" />
      <xsd:element name="RelatedTo" minOccurs="0" maxOccurs="1" >
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ClassURI" type="tns:non_empty_string" minOccurs="1"
              maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Metadata" minOccurs="0" maxOccurs="1" >
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="NameValuePair" minOccurs="1" maxOccurs="unbounded" >
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="Name" type="xsd:string" />
                  <xsd:element name="Value" type="xsd:string" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ReturnedOntologyResources">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:ClassResource" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="ClassResource">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ClassURI" type="tns:non_empty_string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Score" type="xsd:float" minOccurs="1" maxOccurs="1" />
      <xsd:element ref="tns:ClassResource" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="StructuralQueryPart">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="1">
        <xsd:element name="SPARQLQuery" type="tns:non_empty_string"/>
        <xsd:element ref="tns:ResourceList"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="KeywordList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Operator" type="tns:OperatorType" minOccurs="0" maxOccurs="1" />
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="tns:KeywordList"/>
        <xsd:element name="Keyword" type="tns:non_empty_string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ResourceList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="SelectiveResource" minOccurs="0" maxOccurs="unbounded" >
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Operator" type="tns:OperatorType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="ResourceURI" type="tns:non_empty_string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

```
  </xsd:element>

  <xsd:element name="FullTextSearchResultList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:FullTextSearchResult" minOccurs="0"
          maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="FullTextSearchResult">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="DocumentXPath" type="tns:non_empty_string" minOccurs="1"
          maxOccurs="1" />
        <xsd:element name="Score" type="xsd:float" minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:simpleType name="non_empty_string">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name = "OperatorType">
    <xsd:restriction base = "xsd:NMTOKEN">
      <xsd:enumeration value = "EXACT"/>
      <xsd:enumeration value = "AND"/>
      <xsd:enumeration value = "OR"/>
      <xsd:enumeration value = "NOT_SELECTIVE"/>
      <xsd:enumeration value = "EXCLUDE"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Appendix C

JCR-TO-ONTO BRIDGE MAPPING DEFINITION USED FOR EXTRACTING THE DOMAIN ONTOLOGY

```
<BridgeDefinitions xmlns="model.jcr2ont.persistence.iks.srdc.com.tr">
  <ConceptBridge>
    <Query>/tags/tags/%</Query>
    <SubsumptionBridge>
      <PredicateName>child</PredicateName>
    </SubsumptionBridge>
    <PropertyBridge>
      <PredicateName>jcr:title</PredicateName>
    </PropertyBridge>
  </ConceptBridge>
  <ConceptBridge>
    <Query>/NewsSubjectCodes/%</Query>
    <SubsumptionBridge>
      <PredicateName>child</PredicateName>
    </SubsumptionBridge>
  </ConceptBridge>
  <InstanceBridge>
    <Query>/NewsArticles/%</Query>
    <PropertyBridge>
      <PredicateName>categorizedBy</PredicateName>
      <PropertyAnnotation>
        <Annotation>instanceOf</Annotation>
      </PropertyAnnotation>
    </PropertyBridge>
    <PropertyBridge>
      <PredicateName>title</PredicateName>
    </PropertyBridge>
  </InstanceBridge>
</BridgeDefinitions>
```