

Configuration of Semantic Web Applications using Lightweight Reasoning

Stuart Taylor

BSc (Hons.), Computing Science, University of Aberdeen, 2007

A thesis presented for the degree of
Doctor of Philosophy
at the
University of Aberdeen.



Department of Computing Science

2014

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date:

Abstract

The web of data has continued to expand thanks to the principles of Linked Data outlined by Tim Berners-Lee, increasing its impact on the semantic web both in its depth and range of data sources. Meanwhile traditional web applications and technologies, with a strong focus on user interaction, such as blogs, wikis, folksonomies-based systems, and content management systems have become an integral part of the World Wide Web. However the semantic web has not yet managed to fully harness these technologies, resulting in a lack of linked data coming from user-generated content. The high level aim of this thesis is to answer the question of whether semantic web applications can be configured to use existing technologies that encourage user-generated content on the Web.

This thesis proposes an approach to reusing user-generated content from folksonomy-based systems in semantic web applications, allowing these applications to be configured to make use of the structure and associated reasoning power of the semantic web, but while being able to reuse the vast amount of data already existing in these folksonomy-based systems. It proposes two new methods of semantic web application development: (i) a reusable infrastructure for building semantic mashup applications that can be configured to make use of the proposed approach; and (ii) an approach to configuring traditional web content management systems (CMS) to maintain repositories of Linked Data.

The proposed approach allows semantic web applications to make use of tagged resources, while also addressing some limitations of the folksonomy approach by using ontology reasoning to exploit the structured information held in domain ontologies. The reusable infrastructure provides a set of components to allow semantic web applications to be configured to reuse content from folksonomy-based systems, while also allowing the users of these systems to contribute to the semantic web indirectly via the proposed approach. The proposed Linked Data CMS approach provides a configurable tools for semantic web application developers to develop an entire website based on linked data, while allowing ordinary web users to contribute directly to the semantic web using familiar CMS tools. The approaches proposed in this thesis make use of lightweight ontology reasoning, which is both efficient and scalable, to provide a basis for the development of practical semantic web applications.

The research presented in this thesis shows how the semantic web can reuse both folksonomies and content management systems from Web 2.0 to help narrow the gap between these two key areas of the web.

Acknowledgements

Firstly I would like to thank to all of my colleagues in the KT and DL groups in the Department of Computing Science for their feedback and encouragement over the years and for the memorable time spent working at the University of Aberdeen.

I would like to express my most sincere thanks to my supervisor Dr Jeff Z. Pan for giving me the opportunity to study a PhD degree. The research presented in this thesis would not have been possible without Jeff's constant support and enthusiasm over the years. I would also like to thank Jeff for his suggestions and improvements to this thesis.

I am also extremely grateful to Prof Chris Mellish for serving as my second supervisor during the majority of my PhD studies, and would like to thank him for all the advice, inspiration and many valuable comments on this work, which have resulted in many improvements.

I would also like to thank Dr Dave Reynolds for acting as my second supervisor during the early part of my PhD and for the opportunity of working with him and Chris Dollin at Hewlett Packard Labs, Bristol. In addition I would like to thank both Dave and Jeff for the opportunity to contribute to the W3C OWL and RIF working groups.

Finally I would like to express my thanks to my parents Donald and Christine for their continual support and incredible patience throughout my studies at the University of Aberdeen.

Contents

List of Tables	9
List of Figures	10
List of Algorithms	12
List of Definitions	13
1 Introduction	16
1.1 Motivation	18
1.1.1 Problem I: Bridging the Web of Data	19
1.1.2 Problem II: Building with Semantic Tools	20
1.1.3 A Next Step for the Web	20
1.2 Thesis Objectives	21
1.3 Reader's Guide	23
2 Background	25
2.1 The Semantic Web	25
2.1.1 Related Standards	27
2.1.2 Linked Data	28
2.2 Description Logic, Ontologies and Reasoning	30
2.2.1 Lightweight Ontology Reasoning	32
2.2.2 Approximating OWL Ontologies	33
2.2.3 Fuzzy Description Logics	35
2.3 TrOWL and ONTOSEARCH2	36
2.3.1 Semantic Approximation	37
2.3.2 Searching Ontologies	37
2.4 Web 2.0 and Tagging	39
2.4.1 Folksonomy-based Tagging Systems	40
2.4.2 Searching Tagging Systems	42
2.5 Query Expansion	43
2.5.1 Measures of Precision, Recall and Relevance	44
2.6 Web Content Management Systems	45

3	Existing Work	47
3.1	Semantic Mashup	47
3.2	Folksonomies and Tagging	48
3.3	Information Retrieval and Structured Data	51
3.4	Query Expansion and Linked Data	53
3.5	Semantic Web Content Management	56
3.6	Our Approach	59
4	Folksonomy Search Expansion	62
4.1	Limitations of Folksonomy Search	63
4.1.1	How Can Ontologies Help?	64
4.2	Folksonomy Search Expansion	64
4.2.1	How Keywords Relate to Ontologies	65
4.2.2	Search Expansion Methods	66
4.2.3	Applications of Folksonomy Search Expansion	67
4.2.4	Example Scenarios	68
4.3	Formalisation of Expansion Methods	74
4.3.1	Tag Relations	74
4.3.2	Search Function	74
4.3.3	Expansion Methods	74
4.3.4	Practical Considerations	77
4.4	Reasoning Requirements	77
4.4.1	OWL 2 DL Search Expansion	78
4.4.2	Non-Reasoning Search Expansion	79
4.4.3	Lightweight Reasoning Search Expansion	81
4.5	Implementation	82
4.5.1	Folksonomy Search Expansion with Taggr	82
4.6	Case Study and Experiments	87
4.6.1	Case Study: MusicMash2	87
4.6.2	Proof of Concept Evaluation: MusicMash2	88
4.6.3	Usefulness of Folksonomy Search Expansion	88
4.6.4	Scalability of Expansion Methods	90
4.7	Summary	91
5	Configurable Semantic Mashup Applications	93
5.1	Reusable Infrastructure for Semantic Mashup	94
5.1.1	Three Layers of Semantic Tools	94
5.2	Layer 3: Ontology Querying and Storage	94
5.2.1	Ontology Query Answering	96
5.2.2	Fuzzy Query Answering	96
5.2.3	Fuzzy Ontology Searching	96

5.3	Layer 2: Folksonomy Search Expansion	97
5.3.1	Common Folksonomy Interface	97
5.3.2	Evaluating Search Expansion	98
5.4	Layer 1: Semantic Mashup	99
5.4.1	Integrating Folksonomies and Semantic Mashup	100
5.4.2	MusicMash2 Architecture	101
5.4.3	Configuring Folksonomy Search Expansion	102
5.5	Case Study: Mashing Linked Music Data	103
5.5.1	Example User Scenario	106
5.6	Summary	107
6	Configuration of Linked Data Content Management Systems	108
6.1	Formalisation of CMS Entities	109
6.1.1	CMS Entity Example	110
6.2	Linked Data Content Management	111
6.2.1	Class Based Browsing	112
6.2.2	Linked Data CMS mapping	113
6.2.3	Class Based Browsing Example	113
6.3	Drupal Linked Data CMS Implementation	117
6.3.1	Implementation using SPARQL Views	118
6.3.2	Configuration and Maintenance	120
6.4	Case Study: Cultural Heritage Linked Data CMS	121
6.4.1	Cultural Heritage Ontology	121
6.4.2	Drupal Linked Data CMS Configuration	122
6.4.3	Querying and Reasoning Requirements	124
6.5	Summary	125
7	Discussion, Conclusions and Future Work	126
7.1	Thesis Overview and Discussion	126
7.1.1	Folksonomy Search Expansion	126
7.1.2	Reusable Infrastructure for Semantic Mashup	128
7.1.3	Linked Data Content Management Systems	130
7.2	Future Work	131
7.2.1	Framework for Folksonomy Search Expansion	131
7.2.2	Lightweight Rule Extension for RIF	132
7.2.3	Configuration Meta Ontology	133
A	Acronyms and Abbreviations	135
B	Namespaces	137
C	OWL 2 Functional Syntax Mapping	138

D	Tag Ontology Overview	140
E	Taggr API Overview	141
E.1	Overview	141
E.2	Dependencies	141
E.3	Class Summary	141
E.3.1	Tagged Resource	141
E.3.2	Keyword Set	142
E.3.3	Expansion Query	142
E.3.4	Search Query	143
E.3.5	Sub Class Expansion	143
E.3.6	Individual Property Value Expansion	143
F	Walkthrough: Drupal Linked Data CMS	145
F.1	Overview	145
F.2	Requirements and Dependencies	145
F.3	Front End User Interface	147
F.3.1	Listings View	147
F.3.2	Details View	148
F.3.3	Update Views	151
F.4	Back End User Interface	156
F.4.1	SPARQL Views Resource Types	156
F.4.2	Views Specification	158
F.4.3	Panels Layouts	160
F.4.4	Panels Pages	161
G	Example Drupal Linked Data CMS Configuration	163

List of Tables

4.1	Music Videos: Artists (Example 1)	69
4.2	Music Videos: Genres (Example 2)	70
4.3	Music Images: Band Members (Example 3)	70
4.4	Music Images: Music-related Keywords I (Example 4)	71
4.5	Music Images: Music-related Keywords II (Example 5)	71
4.6	Film Clips: Associated People (Example 6)	72
4.7	Pet Photos: Breeds of Dog (Example 7)	72
4.8	Sports Car Videos: Sports Car Models (Example 8)	73
4.9	Related Resources: Sibling Sports Cars (Example 9)	73
4.10	Artist names used in the evaluation.	89
4.11	Specifications of test computer	91
4.12	Performance of Individual Property Value Expansion (IPVE)	91
C.1	OWL 2 class and property expressions	138
C.2	OWL 2 axioms	139

List of Figures

1.1	Dog ontology example.	17
1.2	Dog ontology with expressive schema.	18
2.1	RDF Example	25
2.2	RDF Schema Example	26
2.3	OWL Example	26
2.4	SPARQL Example	28
2.5	RIF Rule Example	28
2.6	RDFa Example	29
2.7	Linked Data Cloud Diagram (2011-09-19)	29
2.8	Description Logic Example	30
2.9	Dog ontology example in DL syntax	34
2.10	Fuzzy SPARQL Example	36
2.11	Keyword search with f-SPARQL	39
2.12	Example of an image and its tags.	41
2.13	Dog ontology using OWL vocabulary.	44
4.1	Dog ontology example in DL syntax	77
4.2	Taggr Beta: http://www.taggr.org/	83
4.3	Taggr Architecture	84
4.4	MusicMash2 Alpha: http://www.musicmash.org/	88
4.5	Result set precision for <i>top k</i> search.	90
5.1	Resuable Infrastructure for Semantic Mashup	95
5.2	Querying music data with f-SPARQL	96
5.3	Keyword search with f-SPARQL	97
5.4	ONTOSEARCH2 Search and Query Interface	98
5.5	Taggr Web Interface	99
5.6	Expansion Method keyword retrieval with f-SPARQL	100
5.7	MusicMash2: Architecture	101
5.8	MusicMash 1.0 Screenshot	103
5.9	MusicMash2 Screenshot	106
6.1	Example CMS page displaying details of a person	111

6.2	Animal ontology with complex class hierarchy.	112
6.3	Example person ontology instances	115
6.4	Drupal SPARQL Views specification	119
6.5	Drupal Page Template	120
6.6	Drupal Linked Data CMS Configuration Entities	121
6.7	Listings View in Drupal Linked Data CMS	122
6.8	Details View in Drupal Linked Data CMS	123
6.9	A Page Template Instance in Drupal Linked Data CMS	124
7.1	SPARQL BGP template for IPVE expansion	131
7.2	OWL 2 Punning: Linked Data CMS Configuration	134
F.1	Searching <i>people</i> using full-text search.	147
F.2	Filtering <i>boats</i> by sub-type.	148
F.3	SPARQL queries required to generate a page	149
F.4	Details view for an archaeological site.	150
F.5	Details view of a person page instance.	150
F.6	Creating a new <i>person</i> record.	151
F.7	SPARQL query generated to create a new person record.	151
F.8	Updating the newly created person instance.	152
F.9	SPARQL query to update the relevant fields for the new person instance.	152
F.10	Updating field values for a person page instance.	153
F.11	Updating relationships for a person page instance.	153
F.12	Viewing the updated person page instance.	154
F.13	Deleting a person record from the edit view.	154
F.14	SPARQL query to delete the person record.	155
F.15	Listing of all resource types (browsing classes).	156
F.16	Listing of fields and relationships for the Archaeological Site browsing class.	157
F.17	RDF mapping for <i>english name</i> field.	157
F.18	Listing of all views (page templates).	158
F.19	Views configuration interface for the summary area.	159
F.20	Views configuration interface for the properties area.	159
F.21	Views configuration interface for a relationship area.	160
F.22	Panels layout template for details pages.	160
F.23	List of all panels pages for the browsing classes.	161
F.24	Panel page configuration	161
F.25	Details display configuration	162

List of Algorithms

4.1	Taggr Search Function	86
4.2	Taggr Expansion Function	86
6.1	Linked Data CMS Mapping Algorithm	114

List of Definitions

2.1	Definition (Core Ontology Relationships)	27
2.2	Definition (Lightweight Reasoning)	33
2.3	Definition (Database-style query)	37
2.4	Definition (Logical properties of OWL 2 QL semantic approximation)	37
2.5	Definition (Keyword)	38
2.6	Definition (Keyword Set)	38
2.7	Definition (Keyword Group)	38
2.8	Definition (Tag)	40
2.9	Definition (Tag to Keyword Set Function)	40
2.10	Definition (Tag Set to Keyword Set Function)	41
2.11	Definition (Folksonomy)	41
2.12	Definition (Tagged Resource)	42
2.13	Definition (Relevance)	44
2.14	Definition (Precision)	45
2.15	Definition (Recall)	45
4.1	Definition (Ontology-keyword Association)	65
4.2	Definition (Search Function)	74
4.3	Definition (Retain Mode Expansion Function)	75
4.4	Definition (Sub Class Expansion (SCE))	75
4.5	Definition (Class Sibling Expansion (CSE))	75
4.6	Definition (Class Instance Expansion (CIE))	76
4.7	Definition (Individual Property Value Expansion (IPVE))	76
4.8	Definition (Individual Class Expansion (ICE))	76
4.9	Definition (Individual Property Expansion (IPE))	76
5.1	Definition (Folksonomy Search Expansion Configuration)	102
6.1	Definition (Content Management System (CMS))	109
6.2	Definition (Field)	109
6.3	Definition (Relationship)	109
6.4	Definition (Page Template)	110
6.5	Definition (Page Instance)	110

6.6	Definition (Linked Data CMS Configuration)	112
6.7	Definition (Browsing Class)	112
6.8	Definition (Linked Data CMS)	113

Publications

Taylor, S., Jekjantuk, N., Mellish, C., and Pan, J. Z. (2013). Reasoning Driven Configuration of Linked Data Content Management Systems. In *the Proc. of the 3rd Joint International Semantic Technology Conference (JIST2013)*

Pan, J. Z., Thomas, E., Ren, Y., and Taylor, S. (2012). Exploiting Tractable Fuzzy and Crisp Reasoning in Ontology Applications. In *IEEE Computational Intelligence Magazine*

Thomas, E., Pan, J. Z., Taylor, S., and Ren, Y. (2010b). Lightweight Reasoning and the Web of Data for Web Science. In *the Proc. of the 2nd International Conference on Web Science (WebSci 2010)*

Pan, J., Taylor, S., and Thomas, E. (2009a). MusicMash2: Mashing Linked Music Data via An OWL DL Web Ontology. In *the Proc. of the Web Science Conference 2009 (WebSci09)*

Pan, J., Taylor, S., and Thomas, E. (2009b). Reducing Ambiguity in Tagging Systems with Folksonomy Search Expansion. In *the Proc. of the 6th European Semantic Web Conference (ESWC2009)*

Thomas, E., Pan, J. Z., Taylor, S., Jekjantuk, N., and Ren, Y. (2009). Semantic Advertising for Web 3.0. In *the Proc. of the 2nd Future Internet Symposium (FIS2009)*

Pan, J. Z., Stamou, G., Stoilos, G., Taylor, S., and Thomas, E. (2008). Scalable Querying Services over Fuzzy Ontologies. In *the Proc. of the 17th International World Wide Web Conference (WWW2008)*

Pan, J. Z., Taylor, S., and Thomas, E. (2007). MusicMash2: Expanding Folksonomy Search with Ontologies. In *the Proc. of SAMT2007 Workshop on Multimedia Annotation and Retrieval enabled by Shared Ontologies (MARESO)*

Section 1.3 provides a description of how the publications listed above relate to the chapters in this thesis.

Chapter 1

Introduction

The semantic web is an evolution of the Web into a machine understandable web of data (Berners-Lee et al., 2001). In recent years the web of data has continued to expand thanks to the principles of Linked Data outlined by Tim Berners-Lee (Berners-Lee, 2009), which provide a set of conventions for publishing structured data on the web, encouraging reuse of data and interconnection between datasets. These principles have aided Linked Data in increasing its impact on the semantic web, both in its depth and range of data sources. The Linked Data cloud currently contains approximately 31.6 billion *machine-understandable* Resource Description Framework (RDF) statements, spanning 295 distinct datasets and covering topics from life sciences to government data.^{1,2} For example, the UK Government has played a major role in this effort by exposing a large amount of its data as Linked Data (Shadbolt et al., 2012). However out of these 295 datasets, only 20 datasets covering 0.42% of the RDF statements in the linked data cloud come from user-generated content.

Meanwhile traditional web applications and technologies, with a strong focus on user interactions (often referred to as Web 2.0), such as blogs, mashup, folksonomies, and content management systems have become an integral part of the World Wide Web. For example in August 2011, Flickr announced that it had a total of 6 billion photo uploads alone (softpedia.com, 2011).³ Currently WordPress.com hosts almost 57 million individual blogs, producing a vast amount of user-generated content (wordpress.com, 2012).⁴

The semantic web provides a mass of machine-understandable data available in the form of Linked Data, and Web 2.0 provides the technologies and tools that allow user participation and user-generated content. This observation suggests that the tools and user-generated content from Web 2.0 could be reused in the semantic web to help encourage participation from ordinary web users. However it has been observed that although these two aspects of the web could greatly benefit each other (Ankolekar et al., 2007; Greaves, 2007; Benjamins et al., 2008), the associated technologies are often incompatible. On one hand the semantic web has a firm underpinning of highly structured data and automated reasoning; most notably in the form of the

¹Based on statics published 2011-09-19: <http://www4.wiwiwiss.fu-berlin.de/lodcloud/state/>.

²Appendix A contains a list of acronyms and abbreviations used throughout this thesis.

³Flickr is a social photo sharing website launched in 2004: <http://www.flickr.com/>.

⁴WordPress.com is a blog hosting service: <http://wordpress.com/>.

OWL Web Ontology Language (Bechhofer et al., 2004).⁵ While on the other hand, traditional web content, often organised using folksonomy-based systems, can have little or no structure, which can result in difficulty when trying to reuse content from different sources, or when attempting to make inferences about the resources contained within them.

Ontology reasoning allows for greater value to be inferred from linked data, by allowing applications to make use of the implicit information contained in these datasets. In the example shown in Figure 1.1, the first `SubClassOf` axiom asserts that the class `Dog` is a type of `Animal`. The second and third axioms assert that the classes `Collie` and `Pug` are types of `Dog`, and in the fourth axiom `EnglishShepherd` is asserted to be a type of `Collie`. However it cannot be guaranteed that all of the data in a particular dataset is in a form that can be retrieved easily for a particular application. For example, the fourth axiom asserts that `EnglishShepherd` is a type of `Collie`, but without knowledge of the semantics of `SubClassOf`, it cannot be automatically inferred that `EnglishShepherd` is also a type of `Dog`, or even a type of `Animal`.

```
SubClassOf(Dog Animal)
SubClassOf(Pug Dog)
SubClassOf(Collie Dog)
SubClassOf(EnglishShepherd Collie)
```

Figure 1.1: Dog ontology example.

In the example shown in Figure 1.1, we could use graph traversal techniques to materialise the implicit `SubClassOf` axioms; the transitive nature of the sub class relationship make a graph traversal approach feasible. However on the web where more expressive schemas are present, and often defined using OWL, a straightforward graph traversal approach cannot be applied in general.

Consider the example shown in Figure 1.2, which uses a more expressive OWL schema. In this example, the first axiom asserts that the `eats` property relates members of the class `Terrier` *only* to instances of `DogFood`. The second axiom asserts that all things which are related via the `eats` property to a type of `DogFood` are a type of `Dog` (in other words, all things that eat dog food are dogs). From these two axioms an OWL reasoner can automatically infer that `Terrier` is a type of `Dog` (more formally, we can say that the axiom `SubClassOf(Terrier Dog)` is entailed by the ontology). The third axiom asserts that `Dalmatian` instances are related via the `eats` property *only* to things that are `PremiumDogFood` (Dalmatians only eat premium dog food). The fourth axiom asserts that `PremiumDogFood` is also a type of `DogFood`. An OWL reasoner is able to automatically infer that `Dalmatian` is a type of `Dog` from the relationships between the classes and properties in the last three axioms.

⁵OWL is a web standard for defining ontologies and provides a rich set of class and property constructors. Chapter 2 covers the OWL Web Ontology Language in more detail.

```

SubClassOf(Terrier ObjectAllValuesFrom(eats DogFood))
SubClassOf(ObjectAllValuesFrom(eats DogFood) Dog)
SubClassOf(Dalmatian ObjectAllValuesFrom(eats
    PremiumDogFood))
SubClassOf(PremiumDogFood DogFood)

```

Figure 1.2: Dog ontology with expressive schema.

These two examples are intended to illustrate that by using an ontology reasoner supporting a sufficiently expressive ontology language such as OWL, applications can be supported that make use of arbitrary linked data datasets, regardless of the expressivity of their schema or how the relevant part of the dataset has been asserted.

The main disadvantage of the reasoning approach is that OWL reasoning is known to be computationally expensive (Motik et al., 2009a), and in general is not considered tractable and therefore unsuitable for efficient or scalable web applications.

This criticism suggests that ontology reasoners may not be suitable tools when reusing linked data in a hybrid Web 2.0 and semantic web application, where scalability would be desirable due to potentially large amounts user-generated content. However an answer has been emerging within the semantic web community that *lightweight reasoning*, based on tractable OWL sub-languages (known as “profiles” (Motik et al., 2009a)) and approximation techniques, could be applied to overcome the limitations of traditional ontology reasoning (Ankolekar et al., 2007; Thomas et al., 2010b; Pan et al., 2012). Lightweight reasoning could therefore be used to help form a practical basis for this type of hybrid application.

In addition there has already been significant interest on combining semantic web and Web 2.0. Ankolekar et al. (2007) present the potential for combining Web 2.0 and semantic web technologies in a weblog scenario, illustrating that semantic web and Web 2.0 are not in fact competing visions of the Web and with the right focus can be combined to overcome each other’s limitations. Greaves (2007) also discuss the combination of Web 2.0 and semantic web technologies, concluding that the most crucial area of semantic web technologies that can be of benefit to Web 2.0 lie in its query and reasoning capability. Benjamins et al. (2008) believe that an integration between the technologies used in the semantic web and in Web 2.0 can form the basis for the future generation of semantic-service based computing infrastructure.

1.1 Motivation

In this section we introduce the two main problems that motivate the work in this thesis: (i) how to reuse unstructured, user-generated content from Web 2.0 in semantic web applications (Section 1.1.1); (ii) how to exploit existing tools and methodologies for user-generated content on the web to encourage user-generated content on the semantic web (Section 1.1.2).

1.1.1 Problem I: Bridging the Web of Data

Recently the semantic web and Web 2.0 visions for the web have started to converge. In June 2011 the three major web search engines, Google (Guha, 2011), Yahoo! (Yahoo!, 2011) and Bing (Bing, 2011) united to announce the release of `schema.org`; a set of schemas that allow webmasters to annotate elements in HTML documents with microdata (Hickson, 2012), to provide additional metadata to search engines.⁶ The Linked Data community responded with `schema.rdfs.org` (`schema.rdfs.org`, 2011), in an effort to map `schema.org` to the Resource Description Framework (RDF) (Manola and Miller, 2004), so that it can be used in RDFa annotations (Adida et al., 2012a) in an effort to integrate `schema.org` with the semantic web. Similarly, the popular Drupal Content Management System can already automatically generate `schema.org` markup in RDFa from Content Management System (CMS) pages (`drupal.org`, 2011).⁷ Meanwhile the World Wide Web Consortium (W3C) have been working toward standardising the integration between RDFa and HTML (Adida et al., 2012b).⁸

Despite this recent move to narrow the gap between the semantic web and traditional web content, there is still a large amount of user-generated, annotated web content that is not considered by any existing W3C standardisation effort. For instance, more and more web resources are now being annotated using popular folksonomy-based tagging systems (or social tagging systems), such as Flickr, YouTube⁹ and `del.icio.us`¹⁰; where users are invited to annotate resources with free-text labels, known as tags. These folksonomy-based systems allow great malleability and adaptability, and the simplicity of adding and using tags allows them to be more accessible to users than well defined (more complex) classification systems.

Microdata and tagging can be seen as a way of allowing users to annotate web resources with additional metadata. Thanks to the efforts of the linked data community, `schema.rdfs.org` provides a bridge between microdata and the semantic web. However it is not clear how the tags that have already been annotated using the folksonomy-based approach could also be mapped to RDF and RDFa.

These folksonomy-based tagging systems also suffer from inherent problems (Passant, 2007), such as ambiguity in the meaning of tags and flat organisation of tags. It has been suggested that semantic web ontologies can help address these problems (Hotho et al., 2006; Lin et al., 2009). This points to the conclusion that some form of integration between folksonomies and ontologies could help both improve the characteristics of folksonomies by addressing their limitations, and provide a platform to build applications based on both tagged resources and linked data. However, to be widely applicable on the semantic web, such a platform must also support expressive ontologies (such as the example in Figure 1.2), so that it can be applied to a wide range of ontologies on the web. Additionally to avoid sacrificing the benefits of

⁶Schema.org documentation is available at: <http://schema.org/>.

⁷A web Content Management System (CMS) allow users to author web pages without requiring detailed knowledge of the underlying web technologies (Section 2.6).

⁸The World Wide Web Consortium (W3C) is the primary international web standards organisation: <http://www.w3.org/>.

⁹YouTube is a video-sharing and hosting service: <http://www.youtube.com/>

¹⁰Del.icio.us is a social bookmarking service: <http://del.icio.us/>

folksonomies, a key question remains open: how to exploit the benefits of ontologies without bothering the untrained users of folksonomy-based systems with their rigidity.

1.1.2 Problem II: Building with Semantic Tools

Reusing content from traditional web applications in the semantic web can be a time consuming task, since the proprietary data models used elsewhere in the web are often incompatible with semantic web standards. There are however tools for building these hybrid types of applications. The BBC's Dynamic Semantic Publishing (DST) architecture (Rayfield, 2012) has been used to build the BBC World Cup 2010 website,¹¹ and more recently the 2012 London Olympics website.¹² The DST architecture allows journalists to link news and sports articles to ontological classes and individuals, to provide additional metadata to users reading articles on the BBC website. Semantic MediaWiki (SMW)¹³ allows users to annotate wiki pages with OWL classes and properties (Krötzsch et al., 2006). Semantic Drupal (Corlosquet et al., 2009) also allows users to provide a mapping between CMS pages and fields and RDF classes and properties.

These systems extend traditional types of web applications, which have previously been successfully deployed on the web in similar “non-semantic” applications, by providing some form of integration with semantic web data. These approaches build on tried and tested tools for facilitating user participation in the web. However, these approaches don not directly manage existing linked data, but rather provide a mapping between their own data model to export data using an RDF or OWL vocabulary.

This sort of integration can be seen as a *read or write only* approach, where linked data is either imported or exported from the system. The next step in this evolution of content management systems for the semantic web is a full integration with linked data: allowing ontology instances, already published as linked data, to be managed using these widely used web content management platforms. In addition any approach to reuse linked data within this new type of content management system must also consider expressive ontologies (such as the example shown in Figure 1.2) so that it can be applied to any ontology, regardless of the expressivity of its schema.

The semantic web could greatly benefit from the reuse of traditional web content management systems to encourage user participation, however integrating these existing systems directly into the web of linked data has not yet been fully realised.

1.1.3 A Next Step for the Web

An ontology entity in OWL is identified by a URI reference, which uniquely and unambiguously identifies that resource on the web. The URI reference is used when describing the resource and can be used to link that resource to further related resources. In a tagging system, resources

¹¹The BBC World Cup website was built using their DST architecture: http://news.bbc.co.uk/sport1/hi/football/world_cup_2010/default.stm

¹²The BBC 2012 London Olympics website also built with DST: <http://www.bbc.co.uk/sport/0/olympics/2012/>

¹³Semantic MediaWiki is a semantic extension of the MediaWiki wiki-engine, which is used to power <http://www.wikipedia.org/>

often have local identifiers, which are not usually described by the tagging system in way that is currently compatible with linked data. Tagging systems describe resources with textual annotations, known as tags. These tags provide a set of keywords that can be used when users are searching the tagging system. However, RDF and OWL could be used to describe tagged resources by giving them their own URI, thereby allowing them to be linked to ontology entities. However, this linking would be a manual process performed by human users, which would be time consuming when performed on a large scale.

As discussed in Section 1.1.1, there has been significant interest in combining technologies from Web 2.0 with semantic web applications. A next step for semantic web applications is to develop new types of hybrid application that can make use of both the structure and associated reasoning power of the semantic web and Linked Data, but while being able to reuse the vast amount of user-generated content already existing in tagging systems. Therefore to address the problems outlined above, the reuse of not only the resources held in folksonomy-based tagging systems, but also the paradigm of Web 2.0 user participation and user-generated content in semantic web applications is required. This observation introduces an important research question based on Problem I: can semantic web applications be configured to reuse user-generated content from folksonomy-based tagging systems?

The standard ontology language for expressing schema on the web is OWL 2 (Hitzler et al., 2009), and any approach that can make use of linked data in general must consider expressive ontology schemas (such as the example shown in Figure 1.2) to be widely applicable. However reasoning tasks using these expressive schemas comes at a high computational cost and may be considered unsuitable efficient or scalable web applications. It has been argued that in the context of user participation the web needs lightweight ontologies (Mika, 2005; Ankolekar et al., 2007). However even with lightweight ontologies, when combined with the scale of data on the web, the reasoning tasks can be computationally expensive, suggesting that the semantic web also needs lightweight reasoning to help form a basis for practical reasoning-driven applications (Ankolekar et al., 2007; Thomas et al., 2010b).

As identified in Section 1.1.2, there are many tools that are well suited to Web 2.0, that could greatly benefit linked data applications. In particular, content management systems can help encourage user-contribution, without bothering ordinary web users with the structure of the underlying web technologies. In addition, to hide much of the structure from the users it may be possible to utilise the power of lightweight reasoning to automatically configure these tools with respect to domain ontologies. This poses another important research question based on Problem II: can existing web content management systems be exploited by to encourage user-generated content on the semantic web?

1.2 Thesis Objectives

The high level goal of the research presented in this thesis is to investigate whether semantic web applications can be configured to use existing technologies that encourage user-generated content on the Web.

Firstly, we aim to make a contribution to solving *Problem I: Bridging the Web of Data* (Section 1.1.1) by presenting a method of combining semantic web technologies and the user-generated content of Web 2.0. To achieve this aim we propose a method to reuse content from folksonomy-based tagging systems in semantic web applications.

Secondly, we aim to make a contribution to solving *Problem II: Building with Semantic Tools* (Section 1.1.2) by presenting an approach to reuse existing tools designed to encourage user-participation on the web to allow users to contribute linked data. To achieve this aim we propose an approach to reuse traditional web content management systems to maintain repositories of linked data.

In order to achieve both these aims we must consider solutions that make use of ontology reasoning, so ontologies that make use of expressive schemas can be supported. Finally, we aim to achieve these contributions in a way that does not burden ordinary web users, allowing these users to continue to use familiar tools but while benefiting from semantic web technologies.

To realise the goal and aims of this thesis we present a set of objectives to answer the following specific research questions, based on the questions raised in Section 1.1.3.

1. Objective 1 is to investigate whether semantic web applications can be configured to reuse user-generated content from folksonomy-based tagging systems, without sacrificing the benefits of folksonomies by limiting the ease of user contribution (Problem I). An approach to achieve this objective should allow users to continue to use existing folksonomy tools to annotate web resources, and also address the following two questions.
 - 1.1 Can the approach proposed in Objective 1 be configured to have high precision? An approach to achieve this objective should make use of the structured data in ontologies to help address the limitations of folksonomy searching in our approach.
 - 1.2 Can the proposed approach in Objective 1 make use of ontology reasoning to support a wide range of ontologies, regardless of the expressivity of their schema? An approach to achieve this objective should make use of lightweight reasoning to provide support for practical semantic web applications.
2. Objective 2 is to investigate whether a reusable infrastructure can be configured to building applications that make use of the approach proposed in Objective 1 (Problem I). An approach to achieve this objective should make use of lightweight reasoning to allow the development of efficient and scalable applications and the infrastructure should be configurable to support the large variety of domains covered by Linked Data.
3. Objective 3 is to investigate whether traditional web content management systems can be configured to maintain repositories of Linked Data (Problem II). An approach to achieve this objective should make use of domain ontologies to help automate the configuration of these tools, and must make use of ontology reasoning so that any domain ontology could be used with the approach, regardless of the expressivity of its schema.

1.3 Reader's Guide

The remainder of the thesis is organised as follows.

Chapter 2 introduces the background of relevant areas of research in further detail. These include the semantic web, ontologies, description logics, reasoning and their related standards, Web 2.0 and folksonomies, query expansion, and Web Content Management Systems.

Chapter 3 reviews the existing approaches most closely related to the objectives of the thesis. This chapter also clarifies how we will address the limitations of existing work.

Chapter 4 presents our Folksonomy Search Expansion approach for reusing resources from folksonomy-based tagging systems in ontology-based semantic web applications. We also show by means of a proof of concept how the proposed search expansion approach can be used to increase search precision in tagging systems.

Chapter 5 proposes a reusable infrastructure for building semantic mashup applications. The infrastructure consists of three layers of semantic tools which have been designed to facilitate the development of semantic mashup applications which make use of our folksonomy search expansion approach.

Chapter 6 presents our Linked Data Content Management Systems (Linked Data CMS) approach as a method to configure existing state of the art web content management systems to manage repositories Linked Data. We present an illustrative case study in the cultural heritage domain using our Linked Data CMS approach.

Chapter 7 concludes the thesis by reviewing the work presented and discussing the extent to which the objectives have been met. Finally we discuss some future work motivated by this thesis.

Appendix A contains a list of acronyms and abbreviations used throughout this thesis.

Appendix B contains a list of namespaces used in examples throughout.

Appendix C presents a mapping between the features of OWL 2 Functional syntax and Description Logic (DL) syntax used in this thesis.

Appendix D provides a description of the tag ontology used by Taggr in Section 4.5.

Appendix E contains an overview of the Taggr API described in Section 5.1.

Appendix F contains a walkthrough of the Linked Data CMS implementation described in Section 6.2.

Appendix G provides an example of a Linked Data CMS configuration that can be defined using our Drupal-based implementation.

Some of the work in this thesis has previously been published: a preliminary version of our Folksonomy Search Expansion approach (Chapter 4) has been published in Pan et al. (2009b); details of our case study semantic mashup application MusicMash2 (Chapter 4, Chapter 5) are published in Pan et al. (2007, 2009a); an initial version of our three layer infrastructure for semantic mashup (Chapter 5) has been published in Pan et al. (2009a), and an updated version of the reasoning infrastructure is presented in Pan et al. (2012); additional details of the lightweight reasoning infrastructure TrOWL, are presented in Thomas et al. (2010b); some details of our application of fuzzy ontology reasoning in our folksonomy search expansion implementation Taggr (Chapter 4, Chapter 5), have been published in Pan et al. (2012), along with the fuzzy DL-Lite query engine in Pan et al. (2008); our Linked Data CMS approach (Chapter 5) has been published in Taylor et al. (2013), and some details of the cultural heritage case study (Chapter 5) have been presented in Beel et al. (2013).

Chapter 2

Background

In this chapter we give a brief background of some of the key research areas investigated in this thesis.

2.1 The Semantic Web

Berners-Lee (1998) outlined the architecture for the semantic web as the web of data and a global semantic database. In this architecture, resources on the web and the links between them are given meaning and are not simply documents with presentational information (Berners-Lee, 1997). In other words, the content that people regularly describe on the web in natural language, should be also described in not just machine processable, but in a machine understandable language, with a well defined meaning and semantics. This architecture for the semantic web depends on three core layers: the assertional layer, the schema layer and the logic layer.

RDF (Klyne and Carroll, 2004) is the assertional layer, it allows us to make assertions about resources on the web, and how they relate to other things. For example, using RDF we can say that *Stuart knows Jeff*; *Stuart is a PhD-Student*; and that *Jeff's homepage-is `http://www.csd.abdn.ac.uk/~jpan`*; using a *subject-predicate-object* triple pattern (Figure 2.1).

```
ex:Stuart ex:knows ex:Jeff ;  
    rdf:type ex:PhDStudent .  
ex:Jeff ex:homepage <http://www.csd.abdn.ac.uk/~jpan> .
```

Figure 2.1: RDF Example

This language provides only the expressivity to assert metadata, and exchange this metadata between applications. However it does not provide the semantics of the terms used in the assertions, nor does it allow us to assert that, e.g., `foaf:knows` is a property connecting two resources rather than a property relating a resource to a literal value.

RDFS (RDF Schema) (Brickley and Guha, 2000) is the schema layer of the semantic web, providing a basic set of classes and properties for defining the schema of RDF documents, which forms a basic ontology language. For example, RDFS provides classes such as `rdfs:Resource`, the class of all things described by RDF; `rdfs:Class`, a metaclass for describing new classes of

RDFS resources; and properties such as `rdfs:range` for defining the *range* of an `rdf:Property`. With RDFS we can now assert that the range of the `ex:knows` property is a resource rather than a literal value (Figure 2.2).

```
ex:knows rdfs:range ex:Person .
ex:Person rdf:type rdfs:Class .

ex:PhDStudent rdfs:subClassOf ex:Person .
ex:name rdf:type rdf:Property ;
    rdfs:range rdfs:Literal .
```

Figure 2.2: RDF Schema Example

The RDFS semantics (Hayes, 2004) allows some useful inferences to be made based on an RDFS document. For example, by using the RDFS Entailment Rules (Hayes, 2004, Sec. 7.2), we can infer that `ex:Stuart` is a `Person`. It is worth noting that it is not possible to check the consistency of an RDF/RDFS documents. Since RDFS does not include negation, it is not possible to assert a contradiction.

OWL Web Ontology Language (Bechhofer et al., 2004) provides the logical layer of the semantic web, providing a powerful ontology language on which inference and consistency checking reasoning services can be established. The OWL ontology language provides a set of class and property constructors, allowing the creation of complex class and property descriptions based on atomic ones. For example, using OWL we can express the axiom: *PhD Students have an Undergraduate Degree* and that *Undergraduate Degrees and Persons are disjoint* (Figure 2.3).

```
ClassAssertion(ex:Stuart ex:PhDStudent)
SubClassOf(ex:PhDStudent ObjectSomeValuesFrom(
    ex:hasDegree ex:UndergraduateDegree))
SubClassOf(ex:PhDStudent ex:Person)
ClassAssertion(ex:Stuart ex:UndergraduateDegree)
DisjointClasses(ex:Person ex:UndergraduateDegree)
```

Figure 2.3: OWL Example

OWL semantics (Patel-Schneider et al., 2004) allow us to infer that `ex:Stuart` has an undergraduate degree because of the following two axioms:

```
ClassAssertion(ex:Stuart ex:PhDStudent)
SubClassOf(ex:PhDStudent ObjectSomeValuesFrom(
    ex:hasDegree ex:UndergraduateDegree))
```

It also allows us to detect an inconsistency in the data:

```
ClassAssertion(ex:Stuart ex:PhDStudent)
```

```
ClassAssertion(ex:Stuart ex:UndergraduateDegree)
SubClassOf(ex:PhDStudent ex:Person)
DisjointClasses(ex:Person ex:UndergraduateDegree)
```

The three core layers of the semantic web’s architecture are now in place thanks to the standardisation efforts of the W3C community.

Ontologies have a close link with controlled vocabularies and thesauri (Garshol, 2004; Roussey et al., 2011), and also have a close link with conceptual graphs (Sowa, 1984) and semantic networks (Sowa, 1987). These formalisms can be used to represent a structured vocabulary using a set of five core relationships: generalisation (“is-a”), instance, partitive (or part-of), associative, and equivalence (Amann and Fundulaki, 1999). These core relationships correspond to typical topological or graph structures found in OWL and RDFS schemas: sub class/-property axioms (generalisation), individual assertions (instance), property assertions (partitive and associative), and equivalent classes/same-as (equivalence). In this thesis we define the core relationships used in OWL ontologies as follows.

Definition 2.1 (Core Ontology Relationships). *The core relationships of OWL ontologies are: generalisation, instance, partitive, associative, and equivalence.*

OWL 2 Web Ontology Language

Since the standardisation of OWL (Patel-Schneider et al., 2004), a new version of the OWL Web Ontology Language has been standardised by the W3C (OWL Working Group, 2009). OWL 2 extends OWL with a range of new expressive features: keys; property chains; richer datatypes, data ranges; qualified cardinality restrictions; asymmetric, reflexive, and disjoint properties; and enhanced annotation capabilities.

It has been recognised that OWL 2 is a very expressive language (both computationally and for users) and thus can be difficult to implement well and to work with (Hitzler et al., 2009). The OWL 2 specification addresses the issue by introducing a set of three tractable sub-languages, the OWL 2 Profiles (Motik et al., 2009a). These profiles are defined by placing restrictions on OWL 2 DL in such a manner that efficient reasoning mechanisms can be implemented. Each of the profiles are suited to expressing a particular type of ontology; such as those with a rich class hierarchy, or those with a large number of individuals. In Section 2.2.1 we discuss the OWL 2 Profiles and their relationship with lightweight reasoning in further detail.

2.1.1 Related Standards

SPARQL (Prud’hommeaux and Seaborne, 2008) (SPARQL Protocol and RDF Query Language) is an RDF query language. It is currently the W3C recommendation for querying the web of data; in a similar fashion to the SQL (Structured Query Language) for relational databases (SQL:2011, 2011). SPARQL allows triple patterns to be matched against RDF data sources and returned as a table of results, or an RDF graph (constructed using a template specified in the query). For example, the following SPARQL query returns all people who have an undergraduate degree (Figure 2.4).

```

SELECT ?person WHERE {
  ?person a ex:Person ;
    ex:hasDegree [ a ex:UndergraduateDegree ] .
}

```

Figure 2.4: SPARQL Example

SPARQL 1.1 (SPARQL, 2012) adds some further features to SPARQL, including new expressive features to the query language, an update language and entailment regimes, where the level of inference can be specified (e.g., RDFS entailment or OWL entailment).

RIF (Rule Interchange Format) (Kifer and Boley, 2010) is a W3C standard that provides both a level of interoperability between different rule systems and compatibility with key semantic web standards such as RDF and OWL. Being an interchange format, rulesets are intended to be translated from their original syntax to RIF XML format and exchanged via the web. However it is also possible to author rules in RIF syntax directly. RIF also provides a standard for compatibility with OWL and RDF (de Bruijn, 2010).

```

forall ?x ?y (
  ?x[ex:dislikes -> ?y] :-
  And( ?x[rdf:type -> ex:MetalBand]
    ?y[rdf:type -> ex:BoyBand] ) )

```

Figure 2.5: RIF Rule Example

The rule shown in Figure 2.5 would be difficult to express in OWL, but can be easily expressed using a straightforward RIF rule. Consequently, the combination of rules and ontologies is seen as an important next step for the semantic web.

RDFa (Adida et al., 2012a) is an extension to HTML (Hyatt and Hickson, 2012) and XHTML (Altheim and McCarron, 2001) allowing RDF assertions to be embedded directly into the markup of web pages. This allows a variety of possibilities for data integration and exchange on the web. Since traditional (X)HTML documents provide only structural elements, such as paragraphs, headings and lists, it is difficult for machines to extract the *meaning* of a document. However, since RDFa can use OWL vocabulary, it is possible for software to extract structured *machine understandable* information from web pages which use RDFa. Figure 2.6 shows an example HTML5 document excerpt with RDFa annotations.¹

2.1.2 Linked Data

Linked Data was coined by Berners-Lee (2009), and refers to a set of conventions for publishing RDF datasets on the web. The main ideas are to use HTTP URIs to refer to things, allowing

¹HTML5 is the latest version of the HTML Hypertext Markup Language currently under development by the W3C HTML Working Group: <http://www.w3.org/html/wg/>.

```

<body
  about="http://musicmash.org/artist/Metallica"
  typeof="http://musicmash.org/MusicArtist">
  <h1>Artist Page:
    <span property="foaf:name">Metallica</span></h1>
  <p property="dc:description">
    Metallica is an American heavy metal band from Los
    Angeles, California.
  </p>
</body>

```

Figure 2.6: RDFa Example

these URIs to be dereferenced in order to discover further related information, in W3C standard formats such as RDF and OWL. These datasets should also refer to further datasets where necessary so that other related resources can be found. This has led to a vast increase in the number of RDF Linked Data datasets being published on the web (Bizer et al., 2009a).

To give an idea of the current scale of this data, Cyganiak (2011) has produced a number of linked data cloud diagrams over the past years.² The current Linked Data cloud diagram (published 2011-09-19) is shown in Figure 2.7, totalling approximately 31.6 billion RDF triples.

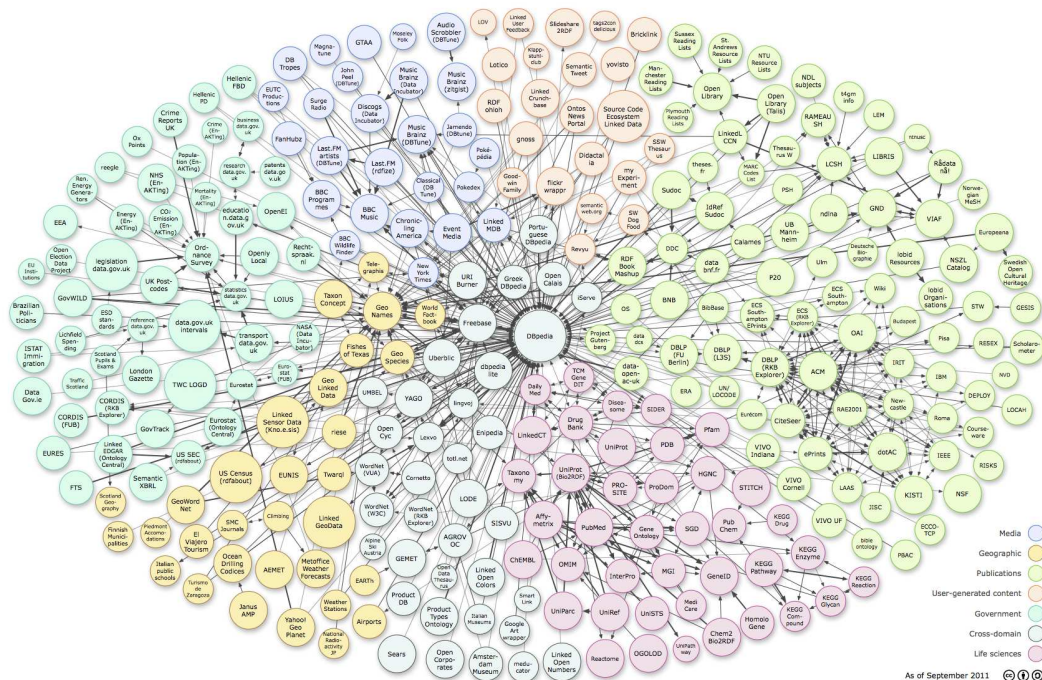


Figure 2.7: Linked Data Cloud Diagram (2011-09-19)

The datasets in the cloud diagram are divided into the following categories (Bizer et al., 2011):

²Linked Data cloud diagrams 2008—present: <http://richard.cyganiak.de/2007/10/lod/>.

- Media (25 datasets; 5.82% of the total RDF triples in the cloud)
- Geographic (31 datasets; 19.43%)
- Government (49 datasets; 42.09%)
- Publications (87 datasets; 9.33%)
- Cross-domain (41 datasets; 13.23%)
- Life sciences (41 datasets; 9.60%)
- User-generated content (20 dataset; 0.42%)

It is interesting to note that 42% of the published datasets (13.3 billion triples) are from government data.

2.2 Description Logic, Ontologies and Reasoning

Description Logic (DL) (Baader et al., 2002) is a family of knowledge representation languages, which form the logical under-pinning of the OWL Web Ontology Language (Motik et al., 2009b). An OWL DL ontology \mathcal{O} consists of a set of classes \mathbf{C} , object properties \mathbf{OP} , datatype properties \mathbf{DP} , individuals \mathbf{I} , and the axioms describing the relationships between them. For example, the OWL ontology in Figure 2.3 can be expressed in DL syntax as shown in Figure 2.8.³

```

PhDStudent  $\sqsubseteq$  Student  $\sqcap \exists hasDegree. UndergraduateDegree$ 
PhDStudent  $\sqsubseteq$  Person
Person  $\sqcap UndergraduateDegree \sqsubseteq \perp$ 
PhDStudent(Stuart)
UndergraduateDegree(Stuart)

```

Figure 2.8: Description Logic Example

The axioms in an OWL DL ontology are divided into the TBox \mathcal{T} (terminological box; or *schema*) and the ABox \mathcal{A} (assertional box; or *data*). TBox axioms make statements about how classes and properties relate to each other, e.g.,

$$\text{Person} \sqcap \text{UndergraduateDegree} \sqsubseteq \perp$$

While ABox axioms make statements about class and property membership, e.g.,

$$\text{Person}(\text{Jeff}), \text{knows}(\text{Stuart}, \text{Jeff})$$

In this thesis we consider an OWL DL ontology as a tuple $\mathcal{O} = \langle \mathbf{C}, \mathbf{OP}, \mathbf{DP}, \mathbf{I}, \mathcal{A}, \mathcal{T} \rangle$; we use these names to refer to the sets of ontology entities throughout the remainder of the thesis.

³For a mapping between OWL and DL syntax we refer the reader to Appendix C.

In this section we have introduced DL syntax as an alternative syntax for expressing OWL ontologies. Appendix C provides a reference for the mapping between OWL 2 and DL syntax. For the reasoning examples presented in this chapter, and later in the thesis, we make use of the more compact DL syntax.

Ontology Expressivity

DLs are typically named with respect to the expressive features provided by the language. Class constructors (class expressions) in the typical base DL ontology language \mathcal{ALC} (Attributive Language with Complements) are defined as follows:

$$C := \perp \mid \top \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$$

where C and D are class expressions, A is an atomic class and R is a property.

\mathcal{ALC} also contains the class \top , which represents the class of all individuals in \mathbf{I} , and the class \perp which represents the empty class.

Adding further expressivity, e.g., property inclusion axioms \mathcal{H} and inverse property axiom \mathcal{I} results in the DL \mathcal{ALCHI} . The original OWL DL language corresponds to the DL $\mathcal{SHOIN}(\mathcal{D})$ (Horrocks et al., 2000), whereas OWL 2 DL corresponds to $\mathcal{SROIQ}(\mathcal{D})$ (Horrocks et al., 2006), by extending OWL DL with a set of expressive features including property chains \mathcal{R} , qualified number restrictions \mathcal{Q} , and the universal role U which relates all individuals in \mathbf{I} (Golbreich and Wallace, 2010).

Ontology Reasoning

The expressivity of a DL has an impact on the complexity of the core reasoning tasks, and therefore on performance of software systems that make use of DL reasoning. The combined complexity of core reasoning tasks in OWL DL rises from NEXPTIME-complete to 2NEXPTIME-complete in OWL 2 DL due to the interaction of the additional language features.⁴ In a tractable DL, such as \mathcal{EL}^{++} (Baader et al., 2005) (which forms the basis for the OWL 2 EL profile), the complexity of these reasoning tasks is PTIME-complete; DLs with complexity classes greater than PTIME, are considered to be *intractable*.

The core reasoning tasks for a Description Logic knowledge base (or OWL ontology) \mathcal{O} are as follows:

Ontology Consistency Checking is the task of checking if the ontology contains a logical inconsistency; i.e., whether $\mathcal{O} \models \top \sqsubseteq \perp$.

Class Satisfiability Checking is the task of checking if a class expression can have instances; i.e., whether $\mathcal{O} \not\models C \sqsubseteq \perp$.

Class Subsumption Checking is the task of checking if one class expression is subsumed by another; i.e., whether $\mathcal{O} \models C \sqsubseteq D$.

⁴The combined complexity of OWL reasoning tasks is the complexity measured with respect to both the size of the axioms (TBox), the size of the assertions (ABox).

Instance Checking is the task of checking if an individual is an instance of a class; i.e., whether $\mathcal{O} \models C(i)$.

2.2.1 Lightweight Ontology Reasoning

Generally there are two approaches to achieve efficient and/or scalable reasoning for OWL ontologies. The first, is to use a lightweight ontology language (such as those defined in the OWL 2 Profiles specification (Motik et al., 2009a)), which is tractable for the desired reasoning tasks. Each profile poses a set of restrictions on OWL 2 DL in order to ensure that reasoning tasks are tractable. The disadvantage of these profiles is that the user must understand the restrictions used to define the profile, as a single misplaced class expression could easily lead to intractability. The second approach is to use approximation methods (Hitzler and Vrandečić, 2005; Pan and Thomas, 2007) to reduce an ontology written in an expressive ontology language to an approximation in a lightweight ontology language. The advantage of this approach is that users can enjoy the freedom of an expressive ontology language, with the efficient reasoning characteristics of the lightweight language. However, the particular approximation technique and target lightweight language must be carefully chosen so as not to sacrifice too much in terms of soundness and completeness.

The lightweight profiles of OWL 2 are briefly described below:

OWL 2 EL is intended for high performance reasoning with large ontologies, with a reasonably expressive TBox. This language allows for a large variety of class constructors and property axioms. However it is not specifically tailored to reasoning about individuals. The combined complexity of OWL 2 EL reasoning tasks is PTIME-complete. This profile is based on the \mathcal{EL} family of Description Logics (Baader, 2003).

OWL 2 QL is designed for large scale ABox query answering. Both class and property expressions are heavily restricted in this profile. However OWL 2 QL reasoning can scale to an extremely large number of individuals, where the ontology only requires a relatively simple TBox. This profile allows for a natural integration with relational database management systems (RDBMS), meaning that a QL reasoner can exploit existing RDBMS technology to provide high performance query answering for ontologies with large ABoxes. The combined complexity of OWL 2 QL reasoning tasks is NLOGSPACE-complete, and conjunctive query answering is AC^0 in terms of data complexity. This profile is based on DL-Lite family of Description Logics (Artale et al., 2009).

OWL 2 RL provides all of the property expressions available in OWL 2 DL, but makes strict restrictions on the expressivity of class constructors. These features make this profile particularly suited to ontologies with a rich ABox, requiring reasoning about object property assertions. Additionally RL reasoners can be built on top of rule based systems, thereby

allowing an integration with rule extended DBMSs and allowing reasoners to exploit existing rule systems. The combined complexity of OWL 2 RL reasoning tasks is PTIME-complete. This profile is based on the intersection between OWL 2 and Horn rules (Dowling and Gallier, 1984); in particular Description Logic Programs (Grosz et al., 2003b) and pD^* (ter Horst, 2005).

The OWL 2 RL semantics can be expressed by RIF as a set of entailment rules (Reynolds, 2010). It is therefore possible to ‘instantiate’ an OWL 2 RL reasoner using a compatible RIF rules engine. pD^* (ter Horst, 2005) is another lightweight ontology language closely related to OWL 2 RL, which extends RDFS with OWL vocabulary rather than placing restrictions on OWL 2 DL semantics.

It has been argued that the web needs lightweight ontologies (Mika, 2005). For example, the OWL 2 QL profile (which was based on DL-Lite (Calvanese et al., 2004)) can express most features in UML class diagrams but still has a low reasoning overhead; worst case polynomial time, compared to worst case exponential time in the case of most widely used DL-based ontology languages. When we refer to lightweight reasoning we mean this class of tractable ontology languages, or approximated expressive ontology languages that have tractable computational properties (Thomas et al., 2010b).

In this thesis we define *lightweight reasoning* as follows:

Definition 2.2 (Lightweight Reasoning). *Lightweight reasoning is the class of ontology reasoning tasks with tractable computational properties (PTIME or better).*

2.2.2 Approximating OWL Ontologies

The core idea of approximation is to reduce an ontology written in an expressive ontology language to an approximation in a lightweight ontology language. Usually the goal of the approximation process is to reduce the computational cost of reasoning with an expressive ontology language to that of reasoning with a lightweight ontology language.

There are generally two approaches to approximation with OWL ontologies. The first approach is known as syntactic approximation (Hitzler and Vrandečić, 2005; Ren et al., 2010), where the syntax of the ontology written in an expressive ontology language is approximated using a less expressive language. The second approach, known as semantic approximation (Pan and Thomas, 2007), makes use of a reasoner for the expressive ontology language to pre-compute the required entailments of an ontology \mathcal{O}_1 in order to express the ontology in a lightweight language \mathcal{O}_2 , using a knowledge compilation (Selman and Kautz, 1996) approach. Once \mathcal{O}_2 has been computed, a reasoner for the lightweight ontology language of \mathcal{O}_2 can be used.

With both of these approximation approaches there is a sacrifice in terms of soundness or completeness of the subsequent entailments. This sacrifice can be illustrated by applying a syntactic approximation to the ontology presented in Figure 2.9 to approximate it to \mathcal{EL}^{++} , which does not support the universal quantifier “ \forall ” (ObjectAllValuesFrom in OWL syntax). In

- (1) $\text{Dog} \sqsubseteq \text{Animal}$
- (2) $\text{Pug} \sqsubseteq \text{Dog}$
- (3) $\text{Collie} \sqsubseteq \text{Dog}$
- (4) $\text{EnglishShepherd} \sqsubseteq \text{Collie}$
- (5) $\forall \text{eats.DogFood} \sqsubseteq \text{Dog}$
- (6) $\text{Terrier} \sqsubseteq \forall \text{eats.DogFood}$
- (7) $\text{Dalmatian} \sqsubseteq \forall \text{eats.PremiumDogFood}$
- (8) $\text{PremiumDogFood} \sqsubseteq \text{DogFood}$

Figure 2.9: Dog ontology example in DL syntax

this example we apply a naïve approximation by simply replacing all with class expression with an atomic class.

Firstly, without approximation an OWL DL reasoner (Motik et al., 2009b) would entail the following subsumption axioms from the ontology shown in Figure 2.9:

- $\text{EnglishShepherd} \sqsubseteq \text{Dog}$
- $\text{Terrier} \sqsubseteq \text{Dog}$
- $\text{Dalmatian} \sqsubseteq \text{Dog}$

Using the same ontology, an \mathcal{EL}^{++} reasoner (Baader et al., 2005) would miss the second and third entailments, since it does not support the axioms on lines 5–7, and would entail only:

- $\text{EnglishShepherd} \sqsubseteq \text{Dog}$

In our syntactic approximation example, we will use a function $\text{approx}(x)$ to approximate unsupported class expressions, such that given a class expression x that is not supported by \mathcal{EL}^{++} , the function returns a new class name to represent that class expression. The unsupported axioms on lines 5–7 are approximated as follows:

- (5) $X1 \sqsubseteq \text{Dog}$
- (6) $\text{Terrier} \sqsubseteq X1$
- (7) $\text{Dalmatian} \sqsubseteq X2$

The function $\text{approx}(x)$ replaces the class expression $\forall \text{eats.DogFood}$ with $X1$ and replaces $\forall \text{eats.PremiumDogFood}$ with $X2$. Using this approximation the \mathcal{EL}^{++} reasoner can also entail:

- $\text{EnglishShepherd} \sqsubseteq \text{Dog}$
- $\text{Terrier} \sqsubseteq \text{Dog}$

Using this syntactic approximation approach the link between Terrier and Dog via the $\forall \text{eats.DogFood}$ expression is maintained by $X1$. However the \mathcal{EL}^{++} reasoner cannot entail $\text{Dalmatian} \sqsubseteq \text{Dog}$ using our naïve approximation, since the semantics of the axiom containing $\forall \text{eats.PremiumDogFood}$ required to infer $\text{Dalmatian} \sqsubseteq \text{Dog}$ are not maintained by the approximation.

By using more sophisticated approximation techniques it is possible increase number of entailments that are maintained in the approximation. For example, the syntactic approximation

approach proposed by Ren et al. (2010) would preserve the *Terrier* \sqsubseteq *Dog* entailment in the resulting \mathcal{EL}^{++} approximation.

Additionally both the syntactic approximation approach proposed by Ren et al. (2010) and semantic approximation proposed by Pan and Thomas (2007) guarantee soundness, while other approaches guarantee completeness (Pan et al., 2009c). In practice some approximation techniques can guarantee both soundness and completeness for a large proportion of entailments, e.g., Pan and Thomas (2007) guarantee both soundness and completeness for answers to so-called “database-style queries”, i.e., queries containing only distinguished variables.^{5,6}

These examples are intended to illustrate that the degree of “completeness” and/or “soundness” of an approximation is dependant on the characteristics of the particular approximation method chosen. For many practical applications of ontology reasoning it can be beneficial to gain scalability and more efficient reasoning by making a sacrifice in terms soundness or completeness, or by restricting the queries to database-style queries (Pan et al., 2009b; Thomas et al., 2009, 2010b; Pan et al., 2012).

2.2.3 Fuzzy Description Logics

Fuzzy Description Logics (fuzzy DLs) extend traditional crisp DLs with the ability to express vagueness, by allowing fuzzy/imprecise classes (Straccia, 2005).

Following from fuzzy sets and fuzzy logic (Klir and Yuan, 1995), fuzzy DLs allow vague classes such as *Tall*, *Cold* and *Quick*, by associating assertion axioms with a fuzzy degree. A class assertion $C(i)$ is associated with a fuzzy degree $n \in [0, 1]$, which states that individual i is an instance of class C with a degree n .

For example, we can assert that an individual *Donald* is a *Person* who is *Tall* with a degree of 0.8 using the following axioms:

$$\begin{aligned} \text{Person}(\text{Donald}) &\geq 1.0; \\ \text{Tall}(\text{Donald}) &\geq 0.8. \end{aligned}$$

The fuzzy degree is computed using a fuzzy membership function (Klir and Yuan, 1995); where a degree of 0 means that the individual is not a member of the class, a degree of 1 means the individual is fully a member of the class, and a degree between 0 and 1 means that the individual is partially a member of the class. For instance, a fuzzy membership function for the class *Tall* could be defined to associate a person’s height in metres with a fuzzy degree.

In fuzzy DLs the results of fuzzy queries can be filtered or ranked with respect to the degree to which they satisfy the query. Straccia (2006) proposed fuzzy DL-Lite, a lightweight fuzzy DL based on DL-Lite (Calvanese et al., 2004). Straccia presents fuzzy DL-Lite as a basis for efficient conjunctive fuzzy query answering. In fuzzy conjunctive queries, each conjunct of the

⁵The semantic approximation approach proposed by Pan and Thomas (2007) is presented in more detail in Section 2.3.

⁶In a conjunctive query, distinguished variables are variables that appear in both the head and body of the query; non-distinguished variables appear only in the body of the query, i.e., non-distinguished variables are existentially quantified.

query is associated with a degree $n \in (0, 1]$.

For example, we can query for all *Person* individuals that are *Tall* with a degree of at least 0.7 using the following fuzzy query.

$$q(x) \leftarrow Person(x) \geq 1.0, Tall(x) \geq 0.7.$$

Pan et al. (2008) propose a framework of fuzzy query languages for fuzzy DL-Lite ontologies. In this framework, two new fuzzy query languages are proposed: fuzzy threshold queries and general fuzzy queries. Fuzzy threshold queries allow a fuzzy threshold for each query atom to be specified, in order to filter query results (as in the previous example). General Fuzzy Queries use the fuzzy degrees associated with query atoms to compute a fuzzy query answer set, where each result is associated with the degree to which it satisfies the fuzzy query.

Fuzzy threshold queries can be expressed using the fuzzy SPARQL syntax proposed by Pan et al. as shown in Figure 2.10.

```
#TQ#
SELECT ?person WHERE {
  ?person a ex:Person ;
  a ex:Tall . #TH# 0.7
}
```

Figure 2.10: Fuzzy SPARQL Example

In this example `ex:Person` instances will only be returned by the query if they are instances of `ex:Tall` with a degree of at least 0.7.

2.3 TrOWL and ONTOSEARCH2

TrOWL is a tractable reasoning infrastructure for OWL 2 (Thomas et al., 2010a) developed at the University of Aberdeen to support a number of useful features. Firstly it supports OWL 2 QL semantic approximation of OWL 2 ontologies (Pan and Thomas, 2007), which allow for efficient conjunctive query answering over ontologies with a large number of individuals. It also allows users to upload ontologies to a repository and provides a SPARQL query interface. In addition the infrastructure contains ONTOSEARCH2, an ontology search engine that provides a search interface to the repository which automatically associates ontology entities with keywords (Pan et al., 2006; Thomas et al., 2007). The ontology search engine in ONTOSEARCH2 uses a fuzzy SPARQL query engine to perform the searches against the repository based on both the keyword index and ontology entailments (Pan et al., 2008).

In the following sections we present semantic approximation and ontology search components in further detail.

2.3.1 Semantic Approximation

The DL-Lite semantic approximation proposed by Pan and Thomas (2007) consists of two steps, firstly an *axiom set* AS is computed by generating all possible DL-Lite axioms using the vocabulary (C, OP, DP, I) of a source OWL DL ontology \mathcal{O}_1 . In this case the axiom set is finite, and polynomial in size with respect to \mathcal{O}_1 .

Next the *entailment set* ES is computed from AS by checking whether each axiom α in AS is entailed by the source ontology using an OWL DL reasoner, i.e., whether $\mathcal{O}_1 \models \alpha$. If α is entailed by \mathcal{O}_1 then it is added to ES . Once the entailment set has been computed, ES is a DL-Lite ontology \mathcal{O}_2 , which is the semantic approximation of \mathcal{O}_1 .

Query answering against \mathcal{O}_2 is sound for all conjunctive queries if \mathcal{O}_1 is sound, and complete for conjunctive queries that do not contain non-distinguished variables.

This completeness property is particularly useful for applications that require only “database-style” queries, i.e., where all variables are distinguished. In this situation all queries against the DL-Lite approximation are guaranteed to be sound and complete.

Definition 2.3 (Database-style query). *A database-style query is a conjunctive query that does not contain non-distinguished variables.*

Another advantage of the semantic approximation approach is that since a reasoner for \mathcal{O}_1 is used to compute the entailment set, semantic approximation can be extended to support any source ontology language where a suitable ontology reasoner is available.

DL-Lite is also the basis for the OWL 2 QL profile, which shares similar computational properties such as tractable query answering. In the TrOWL infrastructure, Thomas et al. (2010a) extend the DL-Lite semantic approximation approach to OWL 2 QL approximations of OWL 2 ontologies. OWL 2 QL is also a super-language of DL-Lite which is part of the OWL 2 specification (Motik et al., 2009a), and offers same soundness and completeness properties as the DL-Lite semantic approximation.

Definition 2.4 (Logical properties of OWL 2 QL semantic approximation). *Query answering against an OWL 2 QL semantic approximation is guaranteed to be sound for all conjunctive queries, and is guaranteed both sound and complete for all database-style queries.*

2.3.2 Searching Ontologies

ONTOSEARCH2 (Pan et al., 2006; Thomas et al., 2007) is an ontology search engine that is based on the TrOWL ontology infrastructure. The basic components of ONTOSEARCH2 are a keyword search engine and a fuzzy SPARQL query engine (Pan et al., 2008). An important feature of ONTOSEARCH2 is that it automatically extracts keywords from ontologies (found in values of annotation properties and identifiers) and then associates them with related classes, properties and individuals.

The default annotation properties used for keyword extraction in ONTOSEARCH2 are: `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`, `owl:versionInfo`, `dc:title`, `dc:description`, `foaf:name`, `skos:prefLabel`, `skos:altLabel`, `skos:hiddenLabel`. In addition to

these properties, the namespace and local part of the URI (the identifier) of each entity in the ontology is also tokenised to generate keywords. These keywords are weighted in a similar way to those used by major search engines (Fishkin, 2011).

The keyword extraction approach used by ONTOSEARCH2 relies heavily on good quality annotations being present in the target ontology. The OWL specification requires the use of identifiers for entities, but does not mandate the use of labels (Bechhofer et al., 2004). However where labels are not present identifiers can also be meaningful for human readers, e.g., the identifier `PostgraduateStudent` could be used to identify the class of *postgraduate students*. On the other hand it is also possible to use terms which may have no meaning to a human reader, e.g., `C10001` as the identifier for a class. Manaf et al. (2010) conducted a survey of the use of identifiers and labels for ontology entities in a corpus of over 300 ontologies. In this survey the authors found that there are in fact very few ontologies that do not use either meaningful identifiers or labels, suggesting that the keyword extraction approach used in ONTOSEARCH2 could be applied in the majority of cases.

In this thesis we define a keyword in the following way:

Definition 2.5 (Keyword). *A keyword k is a single natural language term that does not contain any delimiters; keywords are not case-sensitive.*

Lower and upper case characters in a keyword representing the same letter are considered to be equivalent, e.g., “Keyword” and “keyword” represent the same keyword.

A source of keywords (i.e., annotation or tokenised URI) corresponds to a keyword set, which is defined as follows:

Definition 2.6 (Keyword Set). *A keyword set $K = \{k_1, \dots, k_n\}$ is a set of keywords.*

For example the property assertion:

```
ex:stu foaf:name 'Stuart Rae Taylor' .
```

could correspond to the keyword set:

$$\{\text{“stuart”}, \text{“rae”}, \text{“taylor”}\}.$$

For convenience we also define keyword groups:

Definition 2.7 (Keyword Group). *A keyword group $\mathbf{K} = \{K_1, \dots, K_n\}$ is a set of keyword sets.*

A keyword group can be used to group keyword sets, e.g.,

$$\{\{\text{“stuart”}, \text{“rae”}, \text{“taylor”}\}, \{\text{“donald”}, \text{“rae”}, \text{“taylor”}\}, \{\text{“christine”}, \text{“mary”}, \text{“taylor”}\}\}$$

ONTOSEARCH2 calculates a score n for each keyword k in relation to particular ontology entity i using a scoring function:

$$n = s(i, k).$$

The scores are weighted based on where each keyword appears in the ontology in relation to that ontology entity, e.g., keywords appearing in the `rdfs:label` or identifier of an entity are weighted more highly than keywords appearing in the namespace of the entity.

Once the score has been calculated it is normalised using a sigmoid function (Function 2.1) to a degree between 0 and 1.

$$w(n) = \frac{2}{1.2^{-n} + 1} - 1 \quad (2.1)$$

The score n for each keyword k is associated to an ontology entity i with a degree $d(i, k)$:

$$d(i, k) = w(s(i, k)).$$

Pan et al. do not provide a precise definition of the scoring function $s(i, k)$ used in ONTOSEARCH2, so we cannot present the details of how the weightings of keywords are calculated for ontology entities. However we can still make use of the keyword metadata generated by ONTOSEARCH2 via its fuzzy SPARQL query engine. Metadata queries in ONTOSEARCH2 can also be combined with ontology ABox and TBox queries. For example: *searching for ontologies in which class X is a sub-class of class Y, and class X is associated with the keywords “Jazz” and “Rock”, while class Y is associated with the keyword “Album”*. The search could be represented as by the f-SPARQL query shown in Figure 2.11, where the degree for the keyword “jazz” associated with the entity X corresponds to the degree $d(X, \text{“jazz”})$.

```
#TQ#
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX os: <http://www.ontosearch.org/NS/>

SELECT ?x WHERE {
  ?x os:hasKeyword kw:jazz . #TH# 0.5
  ?x os:hasKeyword kw:rock . #TH# 0.7
  ?x rdfs:subClassOf ?y .
  ?y os:hasKeyword kw:album . #TH# 0.8
}
```

Figure 2.11: Keyword search with f-SPARQL

This feature of ONTOSEARCH2 allows other applications to reuse its ontology-keyword extraction via its f-SPARQL interface which can be accessed using the SPARQL protocol (Clark et al., 2008; Pan et al., 2008).

2.4 Web 2.0 and Tagging

Web 2.0 applications have emerged as providers of new types of web resources, focusing on user collaboration, user-generated content, and social media. These applications include blogs, wikis, mashups and folksonomy-based tagging systems, which have all become an integral part

of the web as it stands today. Folksonomies provide a platform for users to collaboratively annotate resources (such as images, videos or web pages) with tags. Tags are typically one or more keywords used to annotate a resource, and are used to provide additional metadata for resources, such as images, where extracting these keywords automatically would be difficult. For example, Figure 2.12 shows an image with a set of tags describing its content.

More and more Web resources become available in popular folksonomy-based systems, such as Flickr, YouTube and del.icio.us⁷. For example, in April 2008 alone, there were about 2.5 million photos geotagged in Flickr.⁸ In addition, mashup applications combine Web resources from more than one web service, often one or more folksonomy-based system, into an integrated experience (Jackson and Wang, 2007), intended to bring further value to the existing data sources.

2.4.1 Folksonomy-based Tagging Systems

One of the main benefits of a folksonomy-based tagging system, and possible reason for its success, is that the tags are not fixed. So users of the folksonomy can freely invent new tags where necessary, in contrast to a fixed taxonomy of terms. A *tag* may be composed of one or more *keywords* depending on the tagging system, where a keyword corresponds to a single term that does not contain any delimiters (Definition 2.5), and a *keyword set* is a set of these keywords (Definition 2.6).

In this thesis we define a tag in terms of keywords as follows:

Definition 2.8 (Tag). *A tag t is an ordered collection of one or more keywords “ $k_1 \dots k_n$ ” used by a user to annotate a web resource in natural language.*

Figure 2.12 shows an example of a web resource and its set of tags TS :

$$TS = \{\text{“Top Gear”}, \text{“Jeremy”}, \text{“Clarkson”}, \text{“Ariel”}, \text{“Acceleration”}, \text{“Ariel Atom”}\}.$$

To convert a set of tags to a keyword set we define the function $tagkw(t)$ which returns the keyword set for a tag.

Definition 2.9 (Tag to Keyword Set Function). *The tag to keyword set function*

$$tagkw : t \rightarrow 2^K$$

takes a single tag t as an input and returns its corresponding keyword set K , such that each element in the keyword set is a valid keyword.

Similarly we define a function $tagkw'(T)$ which returns the keyword set for a set of tags based on Definition 2.9.

⁷Del.icio.us is a social bookmarking service: <http://del.icio.us/>

⁸Geotagging is the process of annotating photos with geographical metadata.



Tags: Top Gear, Jeremy, Clarkson, Ariel, Acceleration, Ariel Atom

Figure 2.12: Example of an image and its tags.

Definition 2.10 (Tag Set to Keyword Set Function). *The tag set to keyword set function returns a single keyword set K for the set of tags T :*

$$tagkw'(T) = \bigcup_{\forall t \in T} tagkw(t).$$

The function $tagkw'(T)$ can be used to find the corresponding keyword set for the tags TS in the example shown in Figure 2.12 as follows:

$$tagkw'(TS) = \{\text{"top"}, \text{"gear"}, \text{"jeremy"}, \text{"clarkson"}, \text{"ariel"}, \text{"acceleration"}, \text{"atom"}\}.$$

Folksonomies

Informally speaking, a folksonomy involves users, resources and tags, and describes the relationships between them in terms of users assigning tags to resources (Hotho et al., 2006). In this thesis we follow the Mika (2005) and Hotho et al. (2006) formal model of folksonomies. The following definition of folksonomy, used throughout this thesis, is equivalent to the definition of folksonomy presented by Hotho et al. (2006).

Definition 2.11 (Folksonomy). *A folksonomy is a tuple $F = \langle \mathcal{U}, \mathcal{T}, \mathcal{R}, A, \Theta \rangle$ where*

- $\mathcal{U}, \mathcal{T}, \mathcal{R}$ are finite sets of users, tags and web resources, respectively;
- $A \subseteq \mathcal{U} \times \mathcal{T} \times \mathcal{R}$ is a set of tag assignments;
- Θ is a set of tag relations; i.e., for each member \prec of Θ , $\prec \subseteq \mathcal{U} \times \mathcal{T} \times \mathcal{T}$ is a tag relation.

For example, the following assignment $\langle u1, \text{"cat"}, r1 \rangle$ says user $u1$ uses “cat” to tag the

web resource r_1 . We could also say that $\langle \text{"cat"} \prec \text{"animal"} \rangle$ is a user-specific sub-tag relationship between “cat” and “animal” for user u_1 ; meaning that “cat” represent a narrow term than “animal” for u_1 . The sub/super-tag relationship can be used in folksonomy-based systems to support user-specific taxonomies.

In this thesis we refer to these resources, such as r_1 , as *tagged resources*.

Definition 2.12 (Tagged Resource). *A tagged resource is a web resource r , for a folksonomy F where*

- r is in set \mathcal{R} , and
- r also appears in the set of tag assignments A .

The approach taken in this thesis does not consider the relationships between individual users and tag assignments, and rather concentrates on tag assignments at a system level, i.e., tag assignments for all users in the folksonomy as a whole.

2.4.2 Searching Tagging Systems

There are advantages and disadvantages of the tagging approach (Halpin et al., 2007). The folksonomy-based systems allow great malleability and adaptability, and the simplicity of adding and using tags allows them to be more accessible to users than well defined (more complex) classification systems. However, these systems also suffer from problems when it comes to providing meaningful search facilities. In this respect Passant (2007) and Angeletou et al. (2009) identified three core limitations of the folksonomy-based tagging approach:

Tag variation (Passant, 2007) / Tag synonymy (Angeletou et al., 2009). Users can use different tags to refer to the same concept. Consider the following two assignments:

$$\langle u_1, \text{"cat"}, r_1 \rangle, \langle u_2, \text{"kitty"}, r_2 \rangle$$

where users u_1 and u_2 use “cat” and “kitty” to tag two different resources r_1 and r_2 , respectively. Even if u_1 and u_2 have the same meaning in mind for the two different tags, the system \mathcal{S} is not able to relate these tags in a general sense. This limitation can also apply to abbreviations and typo errors.

Tag ambiguity (Passant, 2007) / Tag polysemy (Angeletou et al., 2009). Users can use lexically identical tags to denote different meanings. Consider the following two assignments:

$$\langle u_1, \text{"ma"}, r_3 \rangle, \langle u_2, \text{"ma"}, r_4 \rangle$$

where users u_1 and u_2 use “ma” to tag two different resources r_3 and r_4 . Even if u_1 and u_2 use “ma” to express different ideas, the system \mathcal{S} would still return both r_3 and r_4 when another user (who might have a totally different idea of “ma”) searches for “ma”.

Flat organisation of tags (Passant, 2007) / Basic level variation (Angeletou et al., 2009). The lack of structure in folksonomies does not allow for the explicit definition of relationships

between tags. Consider the following two assignments:

$$\langle u1, \text{"cat"}, r5 \rangle, \langle u2, \text{"animal"}, r6 \rangle$$

where users $u1$ and $u2$ use “cat” and “animal” to tag two different resources $r5$ and $r6$, respectively). When another user searches for “animal”, the system \mathcal{S} is able to return $r6$, but not $r5$.

Several approaches have been proposed that combine folksonomies and ontologies to improve the search facilities offered by more traditional tagging systems (Hotho et al., 2006; Echarte et al., 2007; Lin et al., 2009). We discuss the existing work relating to folksonomies and ontologies in further detail in Chapter 3.

2.5 Query Expansion

Query Expansion (Vechtomova, 2009) in the context of search engines, is the process of reformulating an input query based on some heuristics, in order to achieve either increased precision or recall, often at a cost to the other. Query expansion is applied in situations where it is assumed that users are using short queries or do not attempt refine their query to get better results, such as in web search engines (Silverstein et al., 1999).

One application of query expansion is to reformulate a query to include synonyms, therefore increasing recall by also retrieving documents that do not contain the original search term but do contain synonyms of the original search term. For example a search for “dog” could be reformulated as: “dog OR doggy OR canine”. For this type of query expansion straightforward thesaurus relations could be used.

There are situations where thesaurus relationships are not enough to produce the desired query expansion, e.g., where relationships other than synonyms or domain specific knowledge is required. Ontologies provide a means to structure and relate data with well defined semantics that can be used as a source of knowledge for query expansion (Navigli and Velardi, 2003; Ruotsalo, 2012).

WordNet (Miller, 1995) provides a source of ontological information with a wide lexical coverage. However WordNet consists mainly of subclass (“is-a”) relationships, and for this reason has been criticised for its lack of information about the relationships between concepts that appear in different branches of the taxonomy, but are in fact closely related (Stevenson, 2002). Known as the “tennis problem” (Fellbaum, 1998), this problem occurs in WordNet where, e.g., “tennis ball” and “ball boy” are closely related concepts relevant to tennis, but have no relation other than through the root node of the taxonomy (Stevenson, 2002).

Ontologies expressed using OWL (Section 2.1) offer greater expressivity than that found in WordNet, allowing complex relationships between concepts to be expressed. In addition OWL ontologies offer the potential for a wider coverage of domain-specific knowledge in the form of Linked Data (Section 2.1.2). The complex relationships in these ontologies can also be leveraged using reasoning tools (Section 2.2) to extract further, implicit knowledge, from the

ontology.

For instance consider the dog ontology example shown in Figure 2.13 (originally presented in Chapter 1). In this ontology, the domain knowledge contained in the first four axioms (types of dog) could be extracted for query expansion using a graph traversal approach. However the last four axioms make use of some expressive features of OWL, which entail that `Terrier` and `Dalmatian` are also a types of `Dog`. This inference would be missed by a straightforward graph traversal approach, and therefore in general an ontology reasoner is required to support OWL ontologies for query expansion. The use of an appropriate ontology reasoner with query expansion would allow for an approach generic enough to be applicable to any ontology regardless of the expressivity of its schema.

```
SubClassOf(Dog Animal)
SubClassOf(Pug Dog)
SubClassOf(Collie Dog)
SubClassOf(EnglishShepherd Collie)

SubClassOf(Terrier ObjectAllValuesFrom(eats DogFood))
SubClassOf(ObjectAllValuesFrom(eats DogFood) Dog)
SubClassOf(Dalmatian ObjectAllValuesFrom(eats
    PremiumDogFood))
SubClassOf(PremiumDogFood DogFood)
```

Figure 2.13: Dog ontology using OWL vocabulary.

Query expansion employed in search engines rely on a responsive system to deal with the real time user interaction. However the combination of ontologies and query expansion would be intractable (N2EXPTIME-complete in the case of OWL 2 DL reasoning) and would not considered practical for responsive web applications because of computational cost. The combination query expansion and ontology reasoning could be made practical by making use of lightweight reasoning (Section 2.2.1) to reduce the cost of the reasoning tasks.

2.5.1 Measures of Precision, Recall and Relevance

Query expansion comes at cost in terms of precision or recall depending how the expanded query is applied. We can define precision and recall in terms of relevance, where relevance is a measure of whether a resource is relevant or not relevant to a particular user query (Salton and McGill, 1983). Precision is the fraction of relevant resources returned (true positives) out of all results returned (true positives + false positives). Recall is the fraction of the relevant results returned (true positives) out of the number of all possible relevant results (true positives + false negatives). In this thesis we restrict our definition of relevance to a binary property, where a resource is classed as either *relevant* or *not relevant* by to a domain expert.

Definition 2.13 (Relevance). *A resource is relevant for a particular input query if it has been classified as relevant by a domain expert, otherwise the resource is classified as not relevant.*

We can now formally define precision and recall in terms of relevance.

Definition 2.14 (Precision). *Precision is the fraction of relevant results returned (true positives n_{tp}), out of the total number of results returned (true positives n_{tp} + false positives n_{fp}). Precision is calculated using the following function:*

$$prec(n_{tp}, n_{fp}) = \frac{n_{tp}}{n_{tp} + n_{fp}} \quad (2.2)$$

Definition 2.15 (Recall). *Recall is the fraction of relevant results returned (true positives n_{tp}), out of all possible relevant results (true positives n_{tp} + false negatives n_{fn}). Recall is calculated using the following function:*

$$recall(n_{tp}, n_{fn}) = \frac{n_{tp}}{n_{tp} + n_{fn}} \quad (2.3)$$

For example consider a tagging system which allows users to upload and tag images, and consider a user query to find images relating to dogs. There are 100 tagged resources in the system, 20 of the resources are relevant (Definition 2.13) to queries relating to dogs, and 12 of those relevant resources are tagged with “dog”, while the other 8 are tagged with other keywords related to dogs. There may also be other resources in the system which are not relevant but are also tagged with “dog”.

Now consider an example search query for “dog” without query expansion which returns a total of 16 results: 12 relevant results (true positives), and 4 results that are not relevant (false positives). There are 4 relevant resources in the system which were not returned (false negatives). The precision (Definition 2.14) of this query is 75% ($prec(12, 16)$), and the recall is 60% ($recall(12, 20)$). In practice measuring the recall in tagging systems on the web with possibly millions of resources would be difficult, since a domain expert would be required to count all of the relevant resources in the system.

Next consider a search query for “dog” using query expansion to also search for other breeds of dog (e.g., “dog” OR “collie” OR “dalmatian”). This search returns a total of 20 results: 14 relevant results, and 6 results that are not relevant. Recall in this example is increased to 70%, and precision is reduced to 70%. However using a different query expansion technique could potentially increase precision at a cost to recall. For example, rewriting the input query to reduce ambiguity by concatenating breeds of dog (e.g., “collie dog” OR “dalmatian dog”) is likely to increase precision by reducing the number of irrelevant results, but reduce recall by missing the results that do not match the rewritten query. In this sense query expansion has a trade-off between precision and recall.

2.6 Web Content Management Systems

A Web Content Management System (web CMS) is a system that allows website maintainers and contributors to manage the content of a website via a user interface, without relying on any significant knowledge of the underlying technologies involved in running the website.

More specifically, text, image and links can be added to a website without requiring knowledge of typical web technologies such as HTML, Cascading Style Sheets (CSS) (Bos et al., 2008), JavaScript (International, 1999). Furthermore the web CMS abstracts the typical CRUD (create, retrieve, update, delete) operations performed on the web server, typically by server-side scripting such as PHP and Relational Database Management Systems (RDBMS), such as the MySQL RDBMS. Although other types of Content Management Systems exist (such as Enterprise Content Management Systems (Rockley et al., 2002)), in this thesis we concentrate solely on Web Content Management Systems (such as drupal.org (2001); mediawiki.org (2002); wordpress.org (2003); joomla.org (2005)) and therefore we use *CMS* to refer to *web CMS* from now on.

In addition to managing page content, a CMS can typically model various *content types*. These content types define specialisations of a web page, e.g., a content type for a blog post will require some additional attributes such as author and a topic or category; which would not be required attributes of a standard, non-specialised web page. Most modern CMSs allow the user to define custom content types, which consist of the page structure, page attributes, layout and style.

Popular open source Content Management System software (such as those mentioned previously) allow users to author a range of content. Wordpress and MediaWiki are intended respectively as dedicated blogging and wiki CMSs; Drupal and Joomla are targeted at general purpose web content authoring. However, these systems allow the CMS to be extended by installing additional modules usually known as *themes* and *extensions*. Themes allow the look and feel of the CMS to be altered using a set of *templates* to format the appearance and layout of pages in the CMS; e.g., a theme could be used to style a Wordpress blog as an e-commerce website. Extensions add extra functionality to the CMS; e.g., the Views module (drupal.org, 2005) extends the Drupal CMS with an SQL query builder, allowing site administrators to build, execute and display the results of queries against entities in the Drupal database.

Semantic Drupal (Corlosquet et al., 2009) is of particular interest, since it provides a platform to combine Drupal with the semantic web.⁹ The Drupal Semantic Web Group focus on the integration between Drupal and semantic web technologies such as RDF, RDFa and SPARQL, and have successfully produced a number of Drupal extensions addressing these areas.¹⁰ Since the publication by Corlosquet et al. (2009), RDF in Drupal has become part of the core system (drupal.org, 2007). We discuss Semantic Drupal in further detail in Chapter 3.

In the next chapter we look in more detail at these areas, covering the specific approaches that are closely related to the work in this thesis.

⁹A series of guides for configuring Drupal 7 as “Semantic Drupal” can be found at <http://semantic-drupal.com/>.

¹⁰The Drupal Semantic Web Group can be found at <http://groups.drupal.org/semantic-web>.

Chapter 3

Existing Work

In this chapter we present the existing approaches that most closely relate to our objectives outlined in Chapter 1, and provide details of specific approaches in the areas covered in Chapter 2.

In each of the following sections we evaluate one area of related work, and provide an overview to summarise how the limitations of each approach motivates our own work in the remainder of the thesis.

3.1 Semantic Mashup

The semantic web provides the tools to integrate data sources without ambiguity of identity or semantics based on the principles of Linked Data (Section 2.1). This machine understandable data allows semantic web applications to reuse and aggregate data from multiple sources and make further inferences about it. However, there are many sources of data on the web that are still provided in non-machine-understandable form, often produced by Web 2.0 applications and made available to developers via web services. Machine-understandable data from the semantic web is often seen as separate from the user-generated content from Web 2.0, however it has been suggested that these two areas of the web could benefit each other (Ankolekar et al., 2007; Greaves, 2007; Benjamins et al., 2008).

Linked Data now covers a wide variety of topics and domains (Section 2.1.2), however the semantic web can still benefit greatly from the reuse of existing data available from traditional web services. To this end a number of proposals have been made to integrate, or mashup these two aspects of the web.

Ankolekar et al. (2007) present the potential for combining Web 2.0 and semantic web technologies in a weblog scenario, illustrating that semantic web and Web 2.0 are not in fact competing visions of the Web and with the right focus can be combined to overcome each other's limitations. This is a vision of a community-driven effort of adding semantics to the web as a whole, which is largely reflected by the current state of Linked Data, with a growing interest in embedding RDF as a standard part of web publishing (Corlosquet et al., 2009; Kobilarov et al., 2009; Adrian et al., 2010).

Greaves (2007) also discusses the combination of Web 2.0 and semantic web technologies, concluding that the most crucial area of semantic web technologies that can be of benefit to Web 2.0 lie in its query and reasoning capability. Benjamins et al. (2008) believe that the integration

of semantic web and Web 2.0 technologies can form the basis for the future generation of semantic-service based computing infrastructure.

Battle and Benson (2008) propose an integration of existing web services with the semantic web. The integration works by semantically annotating Representational State Transfer (REST) (Fielding, 2000) web services, so that they can be mapped to an OWL ontology. The mapping is performed by associating each field of the web service's XML response and each of the web service's HTTP query parameters to an OWL entity. This mapping is used to back a SPARQL endpoint that can then be used to query the REST web service as if it was a typical SPARQL endpoint. This approach provides a smooth integration for semantic technologies to utilise data from existing web services. However it does place the burden of mapping on the publishers of this data, who may not have the motivation to do so. Similar mapping approaches have been used, where data from existing websites and web services is mapped to Linked Data. These include dbTune (Raimond and Sandler, 2008), which uses the Music Ontology (Raimond et al., 2007) as a vocabulary for publishing music metadata, and DBpedia (Auer et al., 2007) a community-driven effort to extract *infobox* data from Wikipedia and make it available as Linked Data.

Gruber (2007) presents the idea that ontologies and folksonomies can work together; in particular by means of a *tagging ontology*, posing an integration of data from folksonomy-based systems with the semantic web, and reasoning over tagged data across applications.

The combination of semantic web and Web 2.0 technologies has received much attention in recent years. We believe that folksonomy-based systems could offer a path to reuse user-generated content from Web 2.0 in the semantic web (Section 1.1.3).

In Chapter 5 we present a reusable infrastructure for building applications which can configured to make use of data from both Linked Data and Web 2.0, providing developers with a novel method to leverage both semantic web tools and user-generated content from tagging systems in their applications (Objective 2). However in order to combine these two areas of the web we require an approach to reuse content from folksonomy-based systems (Objective 1).

3.2 Folksonomies and Tagging

Folksonomies provide a lightweight annotation mechanism for annotating web content, by means of user defined textual labels, known as tags. However by their nature, folksonomies lack explicit structure and semantics, thereby limiting search facilities. Limitations such as ambiguity of tags, and variation in tags can result in poor search performance (Section 2.4.2). Much work has been done in terms of extracting structure from folksonomies and improving folksonomy search facilities.

There are advantages and disadvantages of the tagging approach in general (Halpin et al., 2007). Folksonomy systems offer only limited search capabilities. The tag keywords and title are often used in a text based search, which does not consider the structure of the folksonomy

itself. Furthermore, since these documents (tagged resources) are relatively small, usually consisting of short pieces of text (the title and tag keywords), traditional tf-idf ranking is not feasible (Hotho et al., 2006).¹ Folksonomy-based systems allow great malleability and adaptability, and the simplicity of adding and using tags allows them to be more accessible to users than well defined (more complex) classification systems. However, these systems also suffer from problems (Passant, 2007), such as ambiguity in the meaning of tags and flat organisation of tags.²

An enhanced folksonomy search algorithm is presented by Hotho et al. (2006), along with a formal model of folksonomies which considers tag relations.³ Inspired by PageRank (Brin and Page, 1998), this approach works on a graph structure extracted from the folksonomy, based on the co-occurrence of tags and users. FolkRank can extract certain structures from the folksonomy, such as topic groups and communities of users. Hotho et al. show that extracting structured data from the folksonomy can help address some of the inherent limitations of the tagging approach. Although this approach does not directly address the problem of combining folksonomy-based systems with semantic web technologies, it shows that structured data does implicitly exist in these systems suggesting that it could potentially be exploited in ontology-based applications.

An ontology for representing tags, resources and users in a folksonomy-based tagging system is presented by Echarte et al. (2007). Additionally an algorithm to instantiate the ontology by importing from tagging systems is presented. The goal of this approach is to make use of the semantics of OWL DL to resolve some of the limitations of folksonomies; most notably tag variations such as synonyms. One important aspect of this approach is that in the *tagging ontology*, a tag is a class (rather than a string literal), which has a set of labels; each label corresponding to a tag variation. This grouping of tags dramatically increases recall, since in this approach tagged resources are associated with all tag variations. In a sense this is a form of query expansion, where a search term can be expanded to each variation of the tag, based on the tagging ontology. However no algorithm to compute tag variations is presented.

Echarte et al. have shown that ontologies and query expansion can be applied to resolve some of the search limitations associated with traditional folksonomy-based systems. Additionally the authors demonstrate that ontologies offer a path to combining technologies from the semantic web and Web 2.0. However although the results are encouraging, their approach does not address the major *tag ambiguity* limitation folksonomy-based systems (Section 2.4.2), leading to the likelihood of sacrificing precision in this approach.

Lin et al. (2009) propose an approach to extract hierarchical ontological structures from a folksonomy. This approach aims to compute the implicit structure present within folksonomies

¹tf-idf (term frequency-inverse document frequency), is a statistical method used to determine how important a word is in a document.

²The limitations of folksonomy-based systems are presented in further detail in Section 2.4.2.

³In this thesis we follow this formalisation and have presented it in further detail in Section 2.4.

to enhance browsing and to allow the folksonomy to be integrated with semantic web applications. This approach makes use of WordNet (Miller, 1995) as an upper ontology but also extends the WordNet approach to support so called *jargon* and *compound* tags. Relationships between these tags are computed using co-occurrence and other analysis to extract the ontology.

Lin et al. also demonstrate that the combination of ontologies and folksonomies is feasible. However the approach suffers from two major drawbacks. Firstly, the ontology extraction process is limited by the upper ontology used in this approach, since WordNet does not provide the coverage required for more current and domain-specific topics, e.g., sports, music or film. Secondly, as discussed in Section 2.5, for query expansion to be effective, hierarchical “is-a” relationships are often not sufficient. We discuss the use of ontological structures and query expansion in the literature in further detail in Section 3.3.

Halpin et al. (2007) expressed a major concern of the folksonomy approach, i.e., whether or not in the folksonomy-based system consensus about which tags best describe certain web resources can be stabilised, so that such tags are used most often with the folksonomy-based systems. There could be a serious problem if no coherent categorisation scheme can emerge at all from collaborative tagging. This is a crucial point because ranking techniques employed by major web search engines today are, to a large extent, influenced by peoples’ opinion of web resources (Hotho et al., 2006). Although Halpin et al. concluded that tagging distributions *tend to* stabilise into power law distributions, the process of stabilisation (with respect to certain contexts) depends on the number of active users and to some extent on the temporal duration of the tagging process, according to their empirical study. Concrete predictions are not yet available on the number of active users and duration of time needed to complete the stabilisation process. The situation of searching across multiple folksonomy based systems could be even worse, since different styles and environments provided by different systems could make it harder to reach stabilised consensus. Some might even argue that it is impossible to reach such web scale stabilised consensus, as consensus from different systems might be conflicting with each other.

The existing work suggests that the combination of structured data and folksonomies can help resolve some of the limitations of folksonomy-based systems. This literature also demonstrates that ontologies can be used as both a means of representing the implicit structures within the folksonomy-based system (Hotho et al., 2006; Echarte et al., 2007) and as a source of ontological relationships to aid the process of extracting meaningful data from these systems (Lin et al., 2009).

In Chapter 4 we propose to make use of query expansion, driven by domain ontologies to both populate a simple tagging ontology and reuse these resources within ontology-based semantic web applications (Objective 1). This should be done in a way which does not sacrifice the key benefit provided by the folksonomy based approach, i.e., in the straightforward annotation process that provides ease of participation for users. We also aim to apply query expansion in a way that is not limited any single domain (or upper ontology), and that can help address the limitations of search in folksonomy based systems (Objective 1.1). In the next section we look

in more detail at how query expansion could be applied to achieve these aims.

3.3 Information Retrieval and Structured Data

Automatic text indexing and query expansion have a long history in the field of information retrieval (Jones, 1991), and more recently information retrieval techniques using structured data have been studied (Voorhees, 1993; Kekäläinen and Järvelin, 1998; Navigli and Velardi, 2003).

In this section we present some prominent examples of existing work related to the use of structured data for information retrieval to determine how best to make use of these techniques to provide a basis for our approach to reuse resources from folksnonomy-based systems in ontology-based semantic web applications. We believe that use of OWL ontologies as a source of structured data in query expansion could potentially serve as a basis for reusing these web resources in ontology-based applications.

Voorhees (1993) describes a document retrieval experiment which attempts to exploit the relationships contained within WordNet (Miller, 1995) for word sense disambiguation and document indexing using concepts from WordNet. “Is-a” relationships (sub/super class relationships) are used to disambiguate terms extracted from text documents and query terms, where synonymy (multiple words having the same meaning) and polysemy (a single word having multiple meanings) are sources of ambiguity. In this approach query and document senses are matched based on the automatic word sense indexing procedure. Voorhees found that while concept-based retrieval was effective in some cases, it was not as effective as existing stem-based approaches. When ambiguity in the query terms was not a problem, the concept-based retrieval performed better. Voorhees points out that query terms are often difficult to disambiguate because queries often consist of few terms. Interestingly this situation is also mirrored in web search where typical user queries often consist of few search terms (Silverstein et al., 1999; Kekäläinen and Järvelin, 1998; Navigli and Velardi, 2003). However Voorhees points out that in order to be effective, a concept-based disambiguation approach must also be effective where short queries are used. Voorhees (1998) confirms similar results using WordNet for query expansion in a later work. In addition Voorhees (1993) suggests that the ambiguity problem could be worse using wider domain than WordNet provides; this observation also suggests that narrowing the domain could reduce the ambiguity problem.

Towell and Voorhees (1998) present a word sense disambiguation technique using semantic information from WordNet. In their experiments the authors found that their technique was effective where the set of possible senses for polysemous words were known. In the cases where the possible word senses were unknown, the classifier was less effective and more prone to selecting an incorrect sense or a word.

Qiu and Frei (1993) present a concept-based query expansion approach using domain knowledge contained in an automatically constructed similarity thesaurus. The authors point out that that a query expansion approach is generally more effective when it is used to retrieve a greater proportion of *useful documents*, rather than a higher total number of documents. Qiu and Frei stress the importance of how the additional search terms used for query expansion

are selected. In their experiments the additional search terms found by selecting terms representing similar concepts to the terms in the original query. In this setting their concept-based retrieval approach showed an increase in retrieval performance when compared with previous approaches using structured data for information retrieval.

Kekäläinen and Järvelin (1998) investigated the effect of query structure on a query expansion approach using ontological relationships between concepts. The authors compare the retrieval performance of five expansion methods: no expansion, synonym expansion, narrower concept expansion, associative expansion, and an expansion method using a combination of these three relationships. The experiments performed by Kekäläinen and Järvelin showed that the performance of the query expansion technique was dependent on the structure of the expanded query, i.e., how the expanded query is used to retrieve resources from the target system. In addition the authors found that the largest expansion, which used all four relationships performed best.

The query structure in the experiments is related to the expansion method used. In this experiment concepts are represented in the query by a string containing the natural language terms describing that concept. The query engine used in the experiments supports relatively complex queries, but the same idea could be applied to web search, where users queries tend to contain fewer words. For example, a simple query using narrower concept expansion for types of dog could be expanded from “dog” to “dog OR collie OR (english AND shepherd) OR dalmatian”; or to provide context to disambiguate the term “dog” the query could be expanded using different operators where the original keyword is added conjunctively with other related keywords “(collie AND dog) OR (english AND shepherd AND dog) OR (dalmatian AND dog)”. (Kekäläinen and Järvelin, 1998), and later Järvelin and Kekäläinen (2000), show that highly structured queries (more complex than the previous example) offer even greater retrieval performance.

Navigli and Velardi (2003) present a series of experiments with five different ontology-based query expansion methods, using two different query evaluation strategies. The first strategy is to only perform query expansion on monosemous words (unambiguous search terms), the second strategy is to apply the query expansion to after performing word sense disambiguation (WSD). Navigli and Velardi suggest that in fact the first strategy offers more advantages due to the poor precision and recall performance of WSD techniques, indicating that the more interesting results of their experiments come from focusing on how the query is expanded, rather than how to disambiguate the original search terms. On this basis their experiments emphasise the comparison of four of the five expansion methods that apply when only monosemous words are selected; the authors accept that this query strategy increases precision at a cost to recall, which is often the goal of query expansion. The expanded queries are evaluated using the Google web search engine in an open domain setting, under the assumption that query expansion is most useful for short queries of two or three search terms, which is often the case for typical web search queries (Silverstein et al., 1999).

Navigli and Velardi (2003) make use of WordNet (Miller, 1995) and SemCor (Miller et al., 1994) as sources of ontological information used to expand the query. The four expansion methods are based on *is-a*, *has-a*, *attribute value* and *similar-to* relationships (and their inverses), extracted directly from WordNet, and *gloss* and *topic* relationships extracted from SemCor and WordNet using a natural language processor. The results of the experiments showed that expansion methods using the semantic relationships derived from the ontologies offered better performance than expansion methods using synonyms or hyponyms (“is-a” relationships) alone. The experiments show that expansion methods based on ontology relationships are useful especially for short searches and where word sense disambiguation is not necessary.

In these experiments both the ontology (WordNet) and target system (Google) were used in an open domain setting. However if the ontology used for query expansion is domain specific, then the WSD problem may be less likely to occur when the domain is narrowed to a specific one, e.g., sports, music or film. Unfortunately WordNet does not directly contain such domain-specific ontological information. Furthermore the approach is limited by the natural language processor used to extract the domain-specific relationships from WordNet and SemCor. This is a serious limitation of the approach, firstly in that the some of the most effective ontological relationships used by the expansion methods rely on the effectiveness of the natural language processor, and secondly that the approach is limited to the domain coverage of WordNet, which is unlikely to contain knowledge of current or domain-specific topics such as those mentioned previously. This observation indicates that we could apply a similar query expansion approach using domain-specific ontologies from the Linked Data on the web. These ontologies are expressed using OWL, a much more expressive ontology language, which in turn means that more ontological relationships are available for query expansion compared with a WordNet-based approach. A query expansion approach based on linked data could make use of the core ontology relationships in OWL (Section 2.1), which also relate the relationships used in the approach proposed by Navigli and Velardi (2003). In addition, a Linked Data based approach would have the potential to cover a wider range of domain-specific knowledge.

Despite some controversy about the effectiveness of using structured data for information retrieval (Voorhees, 1993, 1998; Towell and Voorhees, 1998), we believe that the work in this area has also shown that given the correct application, the use of structured data can be effective (Qiu and Frei, 1993; Kekäläinen and Järvelin, 1998; Järvelin and Kekäläinen, 2000; Navigli and Velardi, 2003). In the next section we review some of the prominent query expansion approaches relating to ontologies and discuss our conclusions relating to our approach and the objectives outlined in Chapter 1.

3.4 Query Expansion and Linked Data

Ontologies in the form of Linked Data provide varied corpora of structured information covering a wide range of domains that could be used for query expansion. OWL ontologies provide the potential for query expansion techniques to make use of the expressive semantic relationships found in OWL ontologies (Section 2.1).

Lee et al. (2008) propose a domain specific query expansion approach which makes use of a domain ontology to expand search terms for learning object retrieval in a learning object metadata (LOM) based system.⁴ Because users of this system lack the domain specific knowledge of the topic they are learning, the LOM retrieval scenario is similar to the web search scenario, in the sense that in both cases users tend to provide few search terms in their query. In this approach, learning objects are indexed with classes from a domain ontology which covers the domain of the learning object domain. The classes in the ontology are annotated with keywords so that they can be matched against a user's query. The query is expanded by building a 'user intention tree' of classes from the domain ontology based on the users query. The query expansion in this situation is performed by identifying target classes based on the input query, building the user intention tree and then retrieving learning objects indexed with the classes in the user intention tree.

The query expansion approach is based on a domain-specific ontology, reducing the need for word sense disambiguation, and is domain-independent in terms of the ontology used to build the user intention tree. This allows the approach to be applied to a wider range of domains than previous WordNet-based approaches. In this case Lee et al. evaluate the approach in a LOM containing Java control structures.

However in this approach the TBox structure used to compute the expansion (i.e., to build the UIT) is fixed, and does not provide the flexibility of the expansion methods approaches proposed by Kekäläinen and Järvelin (1998) and Navigli and Velardi (2003), suggesting that this approach may not be appropriate for all domains and domain ontologies. For instance an ontology with a large ABox but relatively simple TBox is not likely to be suitable for this query expansion approach. Furthermore the resources to be retrieved are manually indexed directly with ontology classes, which limits the approach to learning objects in this case. Additionally in this approach classes in the domain ontology are indexed with keywords and it is not clear whether a manual indexing approach is required before the query expansion algorithm can be applied.

The experiment performed by the authors confirms that domain ontologies can be applied with query expansion for domain-specific search. However the approach presented is tied to learning object retrieval, and is limited to TBox data in the ontology. In addition the approach is based on the target resources being indexed with classes from the domain ontology, and those classes being indexed with keywords. These observations imply that for a more general application domain ontologies to query expansion, a more general approach to extracting keywords from ontologies, and using the output of the query expansion to retrieve resources is required.

Ruotsalo (2012) presents a domain-specific query expansion, based on a vector space model (VSM) (Salton et al., 1975) of RDF triples, and RDFS subsumption reasoning. Queries can be made using a combination of keywords (indexed from RDF literals) and RDF resources to retrieve documents that have been annotated with ontology classes. The query expansion

⁴A learning object is a resource used to gain knowledge of a specific topic.

approach can be used with the extended RDF vector space model alone, or with the addition of ontology reasoning. When reasoning is enabled the query expansion can make use of the entailed RDFS class hierarchy to enhance the VSM-based approach. This approach is evaluated in the cultural heritage domain, where documents have been annotated using the Dublin Core metadata vocabulary (DCMI, 2003). The experiments presented show an increase in accuracy when ontology reasoning is used, and overall the results show a significant improvement when an optimal combination of query expansion and reasoning is used.

The retrieval approach is similar to the approach proposed by Lee et al. (2008), since the target resources to be retrieved have already been annotated with vocabulary from the domain ontology being used for query expansion. Although both approaches allow the use of domain specific ontologies, the actual evaluation of the search in the approach proposed by Navigli and Velardi (2003) is more general in the sense that the target resources are not directly linked to the source of metadata used for the query expansion. The main limitation of the approaches presented by Ruotsalo and Lee et al. is that they are limited to cases where target resources are already associated with entities in the domain ontology.

Ruotsalo (2012) showed a significant increase in performance when ontology reasoning was used by the query expansion method. In their approach, RDFS reasoning was used to include implicit subsumption (“is-a”) relationships. However Navigli and Velardi (2003) showed that ontology-based query expansion approaches are more effective when additional core relationships from the ontology are used. The combination of these two outcomes suggests that ontology reasoning that includes other ontology relationships could improve the performance of the query expansion, by allowing the use of other implicit relationships in the ontology. Furthermore a query expansion approach using OWL reasoning could take advantage of semantic relationships from any OWL ontology available as Linked Data, regardless of the expressivity of its schema, such as the “dog ontology” example presented in Section 2.5. It may be the case the RDFS reasoning was chosen to improve runtime performance over OWL reasoning, however as also identified in Section 2.5, lightweight reasoning could also be applied to address this limitation.

Navigli and Velardi (2003) showed that expansion methods that can take advantage of core ontology relationships offer better performance than synonym/hyponym based expansion methods alone. Navigli and Velardi also demonstrated that query expansion is effective even when word sense disambiguation is not considered. Lee et al. (2008) and Ruotsalo (2012) showed that domain-specific ontologies can be effective, and Ruotsalo (2012) showed that ontology reasoning can improve the performance of the query expansion further. However only the approach proposed by Navigli and Velardi (2003) retrieves resources where no existing semantic annotations are present in the target system.

In Chapter 4 we aim to provide a query expansion based approach to reuse tagged resources in ontology-based semantic web applications. This approach should allow for a selection of expansion methods to be used in a domain-specific setting so that all of the core ontology relationships (Definition 2.1) within OWL ontologies can be exploited (Objective 1.1). It is also

important to consider how the expanded query in each case is used (Kekäläinen and Järvelin, 1998) to ensure that the expanded query is effective. We propose that these expansion methods should use ontology reasoning both to improve the performance of the query expansion, and to make the approach general enough include relationships from expressive ontologies (Objective 1.2). We aim to address the high computational cost associated with OWL DL reasoning by making use of lightweight ontology reasoning to form the basis for practical semantic web applications. This approach should make use of domain ontologies so that the approach is not limited to any specific source of metadata. Finally to make the approach general in terms of ontologies and the resources being retrieved, the query expansion approach should be decoupled from both the source ontology and the target system being searched. We propose to achieve this with an automatic ontology-keyword association method based on ONTOSEARCH2 (Section 2.3.2), meaning that following Navigli and Velardi (2003), both the input and output of the expansion methods are keywords. Our approach should also carefully consider the structure of the query that makes use of the keywords from the query expansion (Kekäläinen and Järvelin, 1998).

3.5 Semantic Web Content Management

Nowadays web content is frequently authored by a user using some form of Content Management System (CMS), including some more specific types of content management, such as wikis and blogs. Integrating Linked Data into the CMS paradigm could not only allow users unfamiliar with the complexities of semantic web technologies to consume Linked Data, but also to produce it based on some of the structured data held in many content management systems (Objective 3). Therefore an approach for semantic web data content management could greatly benefit the web in terms of reuse and aggregation of data held within CMSs.

Semantic MediaWiki (SMW) (Krötzsch et al., 2006) is a semantic extension of MediaWiki,⁵ which allows users to semantically annotate wiki mark-up in the content of wiki articles. In this approach, categories and articles are annotated with OWL classes, relations (links to other articles) and attributes (data values) are annotated with OWL properties. Furthermore, SMW can be queried with its own wiki-based query syntax, allowing queries to also be embedded in wiki articles. Since SMW is annotated with OWL vocabulary, the corresponding OWL assertions can be exported, allowing the wiki's contents to be queried by SPARQL and reused by other semantic web tools. The result of these SMW annotations when combined with Wikipedia could allow Wikipedia to be directly exported to OWL, potentially allowing for a richer integration between Wikipedia and the web of data than is possible with the DBpedia approach (Bizer et al., 2009b).

While this approach allows external OWL vocabularies to be used within SMW, it does not support presenting existing linked data within the wiki, and therefore is limited to managing the OWL axioms generated only from its own wiki articles.

⁵MediaWiki is a popular wiki-engine used throughout the web, most notably it is used by <http://www.wikipedia.org/>.

The BBC's Dynamic Semantic Publishing (DSP) architecture (Rayfield, 2012) was designed to allow BBC journalists to annotate news articles with metadata from a domain ontology.⁶ In this approach, journalists write articles in a similar fashion as they would with a traditional CMS, but can then attach semantic metadata to those articles (*assets*) by annotating them with ontology classes and/or individuals (indirectly via a tagging process). Additionally DSP makes use of the reasoning services provided by the OWLIM semantic repository (Bishop et al., 2011), which implements the OWL 2 RL and OWL 2 QL profiles for lightweight reasoning, to infer broader related annotations based on the domain ontology. Entities in the domain ontology can also be associated (by the journalist) with a set of assets by means of an intermediate simple tagging ontology, so that related assets can then be aggregated into the final article web page.

This approach provides an integration between previously static journalist authored articles and semantic metadata, exploiting the power of reasoning to automatically integrate and aggregate further content into the news article, while hiding the complexities of the underlying domain ontology from the users of the system. DSP both consumes and produces Linked Data, however the approach is geared towards producing articles with annotations to a fixed ontology, or one that is managed outside of the CMS. This approach produces linked data only for a specific ontology, i.e., the tagging ontology used to associate the articles with ontology entities held in a triple store. The major limitation of this approach is that it does not allow the user to create or edit entities in the domain ontologies and therefore is limited to producing data for the intermediate tagging ontology.

Corlosquet et al. (2009) present an extension to Version 6 of the Drupal Content Management System to allow Drupal site administrators to export data from Drupal nodes (pages in the CMS) as RDF. In this approach, content types and fields are given mappings to specified classes and properties. These mappings are then used to embed RDFa triples in the HTML output, allowing the data from a Drupal website to be consumed by external semantic web tools. Furthermore the generated RDF output can also be stored in a local SPARQL endpoint so that it can be queried using SPARQL. An approach for importing external RDF data is also presented, via their RDF Proxy module. This approach requires the user to specify a SPARQL CONSTRUCT query, which is then evaluated against a remote SPARQL endpoint. The RDF Proxy module then uses the previously specified RDF mappings to map the resulting RDF triples to a content type. A new node of the corresponding content type is then instantiated and cached in the Drupal database.

This RDF Proxy approach relies on a certain level of SPARQL expertise from the site administrator, since the CONSTRUCT query is used to map the predicates from the remote store to those used in the local Drupal content types. However, the RDFa output simply relies on the site administrator being aware of the appropriate RDF types and properties for a content type and provides a straight-forward method to allow external semantic web tools to consume

⁶DSP was first used to build the BBC World Cup 2010 website http://news.bbc.co.uk/sport1/hi/football/world_cup_2010/default.stm.

data held in Drupal's database. Since the publication of this work some of the RDF support presented has become part of the core Drupal 7 system.⁷ In general however the core RDF features in Drupal 7 do not allow read and write of Linked Data held in external sources (e.g., a triple store) with the Drupal CMS, since the approach is intended only to either export Drupal data as Linked Data or import existing Linked Data into Drupal's database.

Some of the limitations of consuming Linked Data in Content Management Systems are overcome by Clark (2010) with the SPARQL Views Drupal module. This approach essentially adds a SPARQL query builder to the powerful Views module.⁸ This module allows site administrators to specify a view of a set of fields via a GUI. This specification is then used to generate the appropriate SPARQL query to instantiate the view; therefore not relying on any SPARQL knowledge by the user. In this approach the RDF mappings are specified for specific *SPARQL Views resource types* (intended to correspond an RDF resource) which are a set of fields, each with an RDF predicate mapping; similar to the content type RDF mappings in Corlosquet et al. (2009).

The strength of SPARQL Views is in its utilisation of the Views module's powerful GUI building capabilities, allowing users to build complex GUIs without requiring any web programming knowledge. Additionally, since Views is also a query builder for SQL databases, SPARQL Views makes use of its framework to provide a query builder for SPARQL, meaning the user does not require any knowledge of SPARQL. However, this approach is intended to be read only, i.e., views only display the results from SPARQL queries, but do not allow the corresponding SPARQL endpoint to be updated. Another limitation of this approach, is that when the number of required SPARQL Views resource types increases, e.g., when building views for a large set of OWL classes (where each class would typically be specified as an individual resource type) the task of initially building and maintaining the set of resource types and associated views becomes a serious burden for the user. For example, an ontology with 10 classes, each with 10 properties would correspond to 10 resource types, 100 fields with 100 RDF mappings and at least one single view for each resource type. This leads to the conclusion that, for medium or large sites consuming Linked Data, an OWL to CMS mapping on top of the field to RDF predicate mapping would be greatly beneficial. Furthermore, extending this mapping to generate both views and an update mechanism for the source ontology would allow users to manage Linked Data from within the CMS.

The approaches presented in this section go a long way to applying the advantages of modern Content Management Systems to Linked Data. However so far no approach for general read and write maintenance of Linked Data exists that can be compared to traditional content management systems.

In Chapter 6 we propose an extension to existing content management systems which allows the full management of ontology instances data, i.e., adding the ability to: create, retrieve,

⁷Drupal RDF Core module: <http://drupal.org/project/rdf>.

⁸Drupal Views module: <http://drupal.org/project/views>.

update, delete Linked Data stored in an external triple store from within a web content management system (Objective 3). We propose to automate the process of configuring the CMS based on the target ontology schema.

3.6 Our Approach

We now revisit the research questions presented in Section 1.2 and present our approach to achieve the objectives outlined in Chapter 1 based on the existing work discussed in this chapter.

Objective 1 (Folksonomy Integration) *Can semantic web applications be configured to reuse user-generated content from folksonomy-based tagging systems, without sacrificing the benefits of folksonomies by limiting the ease of user contribution?* In Chapter 4 we make use of query expansion, driven by domain ontologies to provide a method to reuse tagged resources in semantic web applications. This will be done in a way which does not sacrifice the key benefit provided by the folksonomy based approach, in the straightforward annotation process that provides the ease of participation for users and allowing users to continue to tag resources without knowledge of the underlying ontologies. This approach is inspired by the existing work discussed in Section 3.1, Section 3.2, and Section 3.3.

Objective 1.1 (Folksonomy Precision) *Can the approach proposed in Objective 1 be configured to have high precision?* In Chapter 4 we show that our query expansion approach can retrieve tagged resources for reuse in semantic web applications with high precision. To support a wide range of domains we allow query expansion based on user-specified ontologies that can make use of data from both the TBox and ABox of the ontology. The approach supports expansion methods based on ontological relationships, and automatically extracts keywords from ontologies so that a manual indexing and annotation process is not required. This approach is based on the existing work discussed in Section 3.2 and Section 3.3.

Objective 1.2 (Folksonomy Reasoning) *Can the proposed approach in Objective 1 make use of ontology reasoning to support a wide range of ontologies, regardless of the expressivity of their schema?* In Chapter 4 we develop our approach for combining ontology-based applications with tagged resources should make use of ontology reasoning so that it can make use of both the explicit and implicit knowledge in ontologies. This approach should be able to support practical applications in terms of scalability and efficiency by making use of lightweight reasoning. This approach is inspired by the existing work presented in Section 3.3.

Objective 2 (Reusable Infrastructure) *Can a reusable infrastructure be configured to building applications that make use of the approach proposed in Objective 1?* In Chapter 5 we present a reusable infrastructure for building applications which can make use of data from both Linked Data and Web 2.0, by reusing tagged resources from folksonomy-based systems in semantic web applications. This infrastructure is configurable so that it can be

applied to a variety of domains and is designed to support lightweight reasoning to allow for large scale inference. This approach is inspired by the limitations of existing work identified in Section 3.1.

Objective 3 (Linked Data CMS) *Can traditional web content management systems be configured to maintain repositories of Linked Data?* In Chapter 6 we present an extension to existing content management systems which allows the full management of ontology instances data, i.e., create, retrieve, update, delete. We propose to automate the process of configuring the CMS based on the target ontology schema and make use of ontology reasoning so that the approach can make use of both the implicit and explicit structure of domain ontologies. This approach is inspired by the limitations of existing work identified in Section 3.5.

While fulfilling these objectives this thesis makes the following core contributions:

- Our Folksonomy Search Expansion approach makes use of ontology reasoning to automate the reuse of tagged resources in Linked Data and ontology-based semantic web applications, without bothering the untrained users of the tagging systems with the complexity of the underlying ontologies. This approach can be configured to give high precision results based on the structure information in OWL ontologies, by using a selection of expansion methods based on core ontological relationships. This approach contributes a novel use of ontologies for query expansion to retrieve tagged resources from folksonomy-based tagging systems. The approach also contributes an automatic ontology-keyword association and use of lightweight reasoning that allow it to exploit the knowledge contained in domain ontologies, regardless of the complexity of their schema. Lightweight reasoning is used to provide a basis for developing practical applications in terms of efficiency and scalability (Chapter 4).
- Our Reusable Infrastructure for Semantic Mashup provides an approach for building applications which can be configured to make use of our Folksonomy Search Expansion approach to reuse tagged resources in Linked Data and ontology-based semantic web applications. This approach contributes a reusable infrastructure that allows semantic web applications to be configured to use lightweight and folksonomy search expansion to build practical semantic web applications based on ontology reasoning and tagged resources (Chapter 5).
- Our Linked Data CMS approach allows existing CMS software to be configured to create a web site based on a group of ontology classes, which we call class based browsing. We formally define CMS entities and class based browsing and then make use of a user-defined configuration to map ontological entities to CMS entities. This mapping is then used to generate the CMS structure required to create a set of web pages based on the ontology. The mapping process uses an ontology reasoner and SPARQL query engine

to generate the required CMS entities based on ontology entailments. This approach contributes a new method of combining traditional content management systems with ontologies, allowing ordinary web users to contribute to linked data using familiar CMS tools. This approach also contributes a formalisation of CMS entities and a Drupal-based implementation of our approach (Chapter 6).

By fulfilling these objectives, this thesis shows how lightweight ontology reasoning can be used to reuse resources from folksonomy-based tagging systems in semantic web applications, while addressing some of the limitations of the tagging approach. This thesis also shows how semantic web applications can be configured to make use of our folksonomy search expansion approach in a reusable infrastructure of tools built on lightweight ontology reasoning. Finally this thesis shows how ontology reasoning can be applied to allow Linked Data to be managed in a typical CMS setting, familiar to users of traditional web content management systems. In summary, by achieving these objectives we aim to narrow the gap between these two key areas of the web.

Chapter 4

Folksonomy Search Expansion

In this chapter we present our lightweight reasoning-based approach for reusing tagged resources in semantic mashup applications. More specifically we propose Folksonomy Search Expansion, which makes use of domain ontologies and lightweight reasoning to retrieve resources from folksonomy-based tagging systems with a high level of precision, by making use of a number of expansion methods based on the core ontology relationships (Definition 2.1).

In our Folksonomy Search Expansion approach we make use of lightweight ontology reasoning to compute related sets of keywords for classes, properties and individuals in domain ontologies (Objective 1.2). These keywords are then used to search tagging systems for the corresponding tagged resources associated with these ontology entities (via keywords). This approach allows users to continue to contribute tagged resources to the web using well established methods (in folksonomy-based tagging systems) while also being able to exploit the benefits of ontologies via folksonomy search expansion, without bothering untrained users with the rigidity of the underlying domain ontologies (Objective 1, Section 3.6). Ontology reasoning is used so that both the explicit and implicit information from domain ontologies can be exploited by the expansion methods. Our approach can make use of lightweight ontology reasoning (Definition 2.2) to allow to provide a basis for practical semantic web applications.

Since tags are often ambiguous (Section 2.4.2), attempting to match tags with keywords derived in a naïve way from entities in an ontology can result in low precision (Definition 2.14) when retrieving relevant (Definition 2.13) tagged resources. To address this problem, we introduce several *expansion methods*, which make use of the structure of the ontology to build the set of keywords for an ontology entity (Objective 1.1, Section 3.6). The expansion method can be chosen by the developer of a semantic mashup application automate the query expansion process for their particular application domain, without bothering the users of the application with the complexity of the underlying ontologies.

We present a case study and experiments in Section 4.6, which provide an evaluation of our approach in terms of precision using a case study in the music domain. We extend our previous work (Pan et al., 2009b) in this chapter by presenting further details of our approach along with a selection of new expansion methods.

In the remainder of this chapter, we first revisit the limitations of traditional folksonomy searching in Section 4.1. Next we present our Folksonomy Search Expansion approach for

ontology-based query expansion in folksonomy-based tagging systems in Section 4.2. We then formally define each proposed expansion method in Section 4.3. In Section 4.4 we present the reasoning requirements of our Folksonomy Search Expansion approach. The implementation of our approach based on lightweight reasoning in the Taggr system is presented in Section 4.5. Finally in Section 4.6 we present a case study and experiments using Folksonomy Search Expansion in the music domain.

4.1 Limitations of Folksonomy Search

Let us first revisit the formal model of folksonomies (Section 2.4.1) and the limitations of tagging from the aspect of search (Section 2.4.2).

Given a folksonomy-based tagging system \mathcal{S} that uses the folksonomy $\mathbf{F} = \langle \mathcal{U}, \mathcal{T}, \mathcal{R}, A, \Theta \rangle$ (Definition 2.11),¹ there are three main limitations of tagging regarding search in \mathcal{S} as identified by Passant (2007) and Angeletou et al. (2009):

- **Tag variation (tag synonymy).** Consider the following two assignments: $\langle u1, \text{"cat"}, r1 \rangle$, $\langle u2, \text{"kitty"}, r2 \rangle$ (users $u1$ and $u2$ use “cat” and “kitty” to tag two different resources $r1$ and $r2$, respectively). Even if $u1$ and $u2$ have the same meaning in mind for the two different tags, the system \mathcal{S} is not able to relate these tags in a general sense.
- **Tag ambiguity (tag polysemy).** Consider the following two assignments: $\langle u1, \text{"ma"}, r3 \rangle$, $\langle u2, \text{"ma"}, r4 \rangle$ (users $u1$ and $u2$ use “ma” to tag two different resources $r3$ and $r4$). Even if $u1$ and $u2$ use “ma” to express different ideas, the system \mathcal{S} would still return both $r3$ and $r4$ when another user (who might have a totally different idea of “ma”) searches for “ma”.
- **Flat organisation of tags.** Consider the following two assignments: $\langle u1, \text{"cat"}, r5 \rangle$, $\langle u2, \text{"animal"}, r6 \rangle$ (users $u1$ and $u2$ use “cat” and “animal” to tag two different resources $r5$ and $r6$, respectively). When another user searches for “animal”, the system \mathcal{S} is able to return $r6$, but not $r5$.

For example, in the results from a search on YouTube for “Focus” (the Dutch progressive rock band), only 3 of the top 20 items in the list are relevant to that band. The remaining 17 search results relate to alternative meanings of the word “focus”. This simple example illustrates that ambiguity in the meaning of tags is a significant problem for domain specific folksonomy searches. Furthermore, for each of the 3 relevant results, the tags list for each result contained few other relevant tags than “Focus” and the name of the song in the video. In every case, information on at least the type of music, the album, the band members and other relevant information was omitted from the tag list.

¹ \mathbf{F} is a folksonomy where $\mathcal{U}, \mathcal{T}, \mathcal{R}$ are finite sets of users, tags and web resources, respectively, $A \subseteq \mathcal{U} \times \mathcal{T} \times \mathcal{R}$ is called tag assignments, Θ is a set of tag relations.

4.1.1 How Can Ontologies Help?

Here are two observations related to the limitations:

1. The issues of tag variation and flat organisation of tags indicate that the tag relations in Θ are not useful enough for searching at the system level (i.e., *all* users in the system). The tag relations in Θ are with respect an individual user only, and do not capture the consensus about which sets of tags are used by the community as a whole (i.e., the system user) for a particular concept.
2. The issue of ambiguity indicates we need to be able to associate tags with some contexts, so as to reduce the ambiguity.

Observation 2 clearly suggests ontologies could be helpful, as they contain domain-specific knowledge. Observation 1 suggests that ontological structures should (at least) be used in the system level.

In the next section, we will illustrate our approach for combining ontologies and folksonomies, and then show how it can be applied in several key scenarios.

4.2 Folksonomy Search Expansion

The section presents the details of our Folksonomy Search Expansion approach to reuse tagged resources in ontologies-based applications and address the problems of traditional folksonomy search highlighted in Section 4.1.

Informally, the main idea of ontology-based folksonomy search expansion is to make use of the (implicit or explicit) structure of ontologies to define a set of expansion methods that allow us to relate keyword sets (Definition 2.6) that are associated with ontology entities, to further keyword sets for query expansion. These keywords sets are then used to filter tagged resources (Definition 2.12) based on their keyword sets (Definition 2.10). There are two main functions used in our approach: the search function and the expansion function.

The search function $S(K, expansionMethod)$ takes an initial keyword set K (usually relating to an API query from a mashup application) and an expansion method as inputs, and returns a subset of the tagged resources held by a specified tagging system (i.e., the search results):

$$S : K \rightarrow 2^{\mathcal{R}}.$$

The subset of tagged resources is computed by filtering the tagged resources in the tagging system against the keyword group (Definition 2.7) returned by the expansion function $E(K, expansionMethod)$. Each set in the keyword group is interpreted disjunctively by the search function, and each keyword is interpreted conjunctively for a keyword set.²

²The search function is formally defined in Section 4.3.2.

For example the following keyword group:

$$\begin{aligned} &\{\{\text{"stuart"}, \text{"rae"}, \text{"taylor"}\}, \\ &\{\text{"donald"}, \text{"rae"}, \text{"taylor"}\}, \\ &\{\text{"christine"}, \text{"mary"}, \text{"taylor"}\}\} \end{aligned}$$

would be interpreted as:

$$\begin{aligned} &(\text{"stuart"} \text{ AND } \text{"rae"} \text{ AND } \text{"taylor"}) \\ &\text{OR } (\text{"donald"} \text{ AND } \text{"rae"} \text{ AND } \text{"taylor"}) \\ &\text{OR } (\text{"christine"} \text{ AND } \text{"mary"} \text{ AND } \text{"taylor"}). \end{aligned}$$

The expansion function is passed a keyword set K and the expansion method by the search function. The expansion function uses the expansion method to compute a keywords group, with each keyword set corresponding to a particular ontology entity identified by the expansion method:

$$E : K \rightarrow 2^K.$$

The expansion method defines the pattern of the ontology that should be used as the source of additional keywords.^{3,4}

In other words, the search function computes the set of tagged resources that correspond to the ontology entities identified by the expansion function, by matching the keywords contained in the tags, with keywords related to the ontology entities identified by the expansion function for the keywords in K .

4.2.1 How Keywords Relate to Ontologies

To support our Folksonomy Search Expansion approach we require a method of associating keywords with ontologies entities (classes, properties, and individuals). In our approach we use ontology-keyword association (Definition 4.1), to compute the set of keywords related to ontology entities to provide the additional keywords used to perform query expansion on the folksonomy.

Our approach relies on an ontology-keyword association method which we define as follows, building on the definition of keywords (Definition 2.5) and keywords sets (Definition 2.6):

Definition 4.1 (Ontology-keyword Association). *For each ontology entity i (class, property, individual) in a given ontology \mathcal{O} , there exists an associated keyword set $K(i)$ and each keyword $k \in K(i)$ associates to i with a degree $d(i, k) \in (0, 1]$. Similarly, each keyword k is associated with a set of classes $C(k)$, properties $P(k)$ and individuals $O(k)$, where k also associates to each entity i (class, property, individual) with a degree $d(i, k) \in (0, 1]$.*

³In practice each expansion method could be implemented as a parameterised SPARQL query. A look-up table could then be used to retrieve the relevant SPARQL query for each expansion method.

⁴The expansion function for each *expansionMethod* is formally defined in Section 4.3.3.

Note that in Definition 4.1 the keyword association function $K(i)$ can be used independently of the degree function $d(i, k)$, since keywords associated with a degree of 0 are not included in set $K(i)$.⁵

This ontology-keyword association definition is based on the ontology-keyword association methods developed for the ONTOSEARCH2 ontology search engine (Section 2.3.2). Although we have reused the ontology-keyword association approach from ONTOSEARCH2 here, in principle we could make use of any other ontology-keyword association method that provides a similar API, i.e., in general we do not rely on the particular implementation of $K(i)$, $C(k)$, $P(k)$, and $O(k)$ provided by ONTOSEARCH2.

Intuitively our approach is based on relating the keyword sets for tagged resources, with the keyword sets for ontology entities in $K(i)$.

4.2.2 Search Expansion Methods

In this section we give a brief overview of six expansion methods which can be used with our folksonomy search expansion approach. These expansion methods relate closely to the core relationships used in ontologies and controlled vocabularies (Definition 2.1). We give examples of how each expansion method can be used in practical scenarios.

In principle the approach is not limited to just these six expansion methods, however we have limited those described in this section to what we believe are the most useful expansion methods.

An expansion method defines the structure of the ontology which is to be used to expand the initial keywords set passed to the expansion function $E(K, expansionMethod)$ which is used to compute the keyword group used to search the folksonomy. The following list is a brief overview of each expansion method. The formalisation of each expansion method is presented later in Section 4.3.

Sub Class Expansion (SCE) Retrieves the keyword group relating to a class's subclasses. The input keyword set K is used to identify a class D , and K is expanded to a keyword group consisting of the keyword sets for each subclass E , such that $E \sqsubseteq D$ holds in the target ontology.

Class Sibling Expansion (CSE) Retrieves the keyword group relating to a class's siblings. The input keyword set K is used to identify a class D , and K is expanded to a keyword group consisting of the keyword sets for each class E , such that $D \sqsubseteq G$ and $E \sqsubseteq G$ holds in the target ontology.

Class Instance Expansion (CIE) Retrieves the keyword group relating to a class's instances. The input keyword set K is used to identify a class D , and K is expanded to a keyword group consisting of the keyword sets for each individual i , such that $D(i)$ holds in the target ontology.

⁵Degrees are in the range $(0, 1]$ so that keywords with degree 0 are not included in $K(i)$.

Individual Class Expansion (ICE) Retrieves the keyword group relating to an individual's classes. The input keyword set K is used to identify an individual i , and K is expanded to a keyword group consisting of the keyword sets for each class D , such that $D(i)$ holds in the target ontology.

Individual Property Expansion (IPE) Retrieves the keyword group relating to an individual's properties. The input keyword set K is used to identify an individual i , and K is expanded a keyword group consisting of the keyword sets for each property P , such that $\{i\} \sqsubseteq \exists P.\top$ holds in the target ontology.⁶

Individual Property Value Expansion (IPVE) Retrieves the keyword group relating to an individual's property values for a specified property. The input keyword set K is used to identify an individual i , and K is expanded a keyword group consisting of the keyword sets for each individual j , using the specified property P such that the relationship $P(i, j)$ holds in the target ontology.

The six proposed search expansion methods correspond to the core relationships used in ontologies and controlled vocabularies: generalisation, instance, partitive, associative, and equivalence (Definition 2.1). Sub Class Expansion and Class Sibling Expansion make use of generalisation relationships (SubClassOf/"is-a"); Class Instance Expansion, Individual Class Expansion and Individual Property expansion make use of instance relationships (class and property assertions); and Individual Property Value expansion makes use of association/partitive relationships.⁷

Furthermore the relationship used in each of the six expansion methods can exploit the inferred relationships in the ontology using an ontology reasoner. As a result the expansion methods could be implemented by SPARQL queries against the target ontology using a minimal amount of RDF triples (i.e., not requiring explicit graph traversal in the query). Note that the classes in *equivalence* relationships (i.e., $C \equiv D$) are also captured by generalisation queries since the reasoner will infer that the two subclass axioms $C \sqsubseteq D, D \sqsubseteq C$ are equivalent to the class equivalence axiom $C \equiv D$.

For each expansion method, we allow the option of appending the original keyword set K to each keyword set. We refer to this option as *retain mode*, i.e., to append the original keyword set to each keyword set returned by the expansion function, we say the expansion is performed with retain mode enabled. Retain mode adds the input keyword set conjunctively to each keyword set returned by the expansion function.

4.2.3 Applications of Folksonomy Search Expansion

Our Folksonomy Search Expansion approach is intended to allow semantic mashup developers to reuse content (tagged resources) from tagging systems in their mashup applications. The

⁶The axiom $\{i\} \sqsubseteq \exists P.\top$ means that the individual i is related via the property P to some individual in the ontology.

⁷Association and partitive relationships can be expressed using object property assertion axioms. However the distinction between these relationships is used in the expansion methods.

mashup application may often have to make automated queries to the tagging system based on a user's interaction, but without the user's intervention. In this situation the search terms used to search a tagging system are often generated by the mashup application. Our approach allows the developer to specify expansion methods to be used with these automated search terms in order to increase the precision of the results retrieved from a tagging system. The expansion method makes use of a domain ontology (which may also be used by the mashup application directly in some other way) to provide the source of metadata (additional keywords) for the folksonomy search expansion.

The effectiveness of the approach depends on the design and quality of the domain ontology in order to provide a good source of additional keywords for the search expansion. Well designed ontologies (Gangemi, 2005) can provide good coverage of a particular domain, however if a term, class, property, or individual does not appear in the ontology then it cannot be returned by the expansion. It is therefore important for developers to select the target ontology carefully for their intended domain. The linked open data cloud (Bizer et al., 2011) and schema.rdfs.org (schema.rdfs.org, 2011) could be used as a starting point for developers to discover ontologies covering a wide variety of domains.

Halpin et al. (2007) observed that the user community of a tagging system tend to stabilise into a consensus over the classification of tagged resources. The task for a developer using folksonomy search expansion is to identify this classification and select an appropriate expansion method to allow them to exploit the characteristics of the tagging system. However this observation also suggests that different tagging systems, with different user communities, may not necessarily reach the same consensus about how tagged resources should be classified, and as a consequence we accept that the developer may not be able to reuse the same expansion method in all scenarios.

Folksonomy Search Expansion could also be used directly by users to perform domain specific searches of tagging systems where an appropriate expansion method has been selected for the search interface. Silverstein et al. (1999) conducted a large scale study of user behaviour in the context of a web search engine and found that users often use short queries, and often do not attempt refine their query within a particular session. These results indicate that our folksonomy search expansion approach may also be beneficial to users searching tagging systems in a domain specific setting.

4.2.4 Example Scenarios

This section outlines some prominent examples of the limitations of folksonomy search in tagging systems, and how our folksonomy search expansion approach could be applied in each case. These example scenarios assume that a domain ontology is available in order to identify the correct ontology entities for the particular expansion method. In practice Linked Open Data (Section 2.1.2) and Schema.org (schema.rdfs.org, 2011) can be used by developers to locate sources of OWL and RDF data covering a number of domains. In addition the developer must do some experimentation to determine the pattern from their chosen ontology that should be

used as a source of additional keywords for the search expansion.

In each of the following examples we use the following conventions for ease of readability: *Search Term(s)* corresponds to K ; *Example Output* is a keyword group KG returned by the expansion method. Each keyword set returned by the expansion function $E(K, expansionMethod)$ is denoted as a string of keywords. For example the string “example keyword set”, is a keyword set containing the keywords {“example”, “keyword”, “set”}. A keyword group is therefore denoted as { “first keyword set”, “second keyword set”, ... }.

These example scenarios are related to typical applications of query expansion (Section 2.5), and in each case aim to increase either precision or recall, potentially at a cost to the other.

Music Videos

Example 1: A developer would like to search YouTube for videos relating to a specific music artist. “Focus” is an ambiguous search term on YouTube and does not commonly relate to the music artist Focus. However, the developer observes that the YouTube user community often tag videos with both the artist name and song title. This pattern could be applied in a mashup application for all artist searches to help increase precision.

Search Term(s):	“Focus”
Description:	The expansion method uses retain mode to concatenate the original search term <i>Focus</i> with the title of each song created by Focus, resulting in a set of searches that add context to the original search terms.
Example Output:	{ “Focus Hocus Pocus”, “Focus Silvia”, ... }.
Expansion Method:	Individual Property Value Expansion (IPVE); with the property specified as <i>createdSong</i> and retain mode enabled.

Table 4.1: Music Videos: Artists (Example 1)

Example 2: A developer would like to search YouTube for videos relating to a specific *music genre*. However, many related videos on YouTube are may not be tagged with genre information. A search for “Rock” could be expanded to include artists in the genre. Additionally, since an artist name search may result in poor precision, these artist names could then be expanded to help reduce ambiguity.

Search Term(s):	“Rock”
Description:	The expansion replaces the original search term <i>Rock</i> with each artist in the Rock genre. These artists may then be further expanded to reduce ambiguity.
Example Output:	{ “Led Zeppelin”, “AC/DC”, ... }.
Expansion Method:	IPVE; with retain mode disabled.

Table 4.2: Music Videos: Genres (Example 2)

Music Images

Example 3–5: A developer would like to search Flickr for images relating to a specific music artist. However, the search term *Metallica* returns many unrelated results.⁸ Adding further search terms may increase the precision of the results returned. However, the developer may not be able to reuse the expansion method from the previous example, since users may not tag *Flickr images* with particular song names. In this example, the names of band members and music-related keywords (e.g., Live, Concert, Music etc.) are added to increase the search precision. If the Music Ontology (Raimond et al., 2007) is used for search expansion, band member names could be found in the ABox of the ontology and general music-related keyword (domain vocabulary) would be found in the TBox.

Search Term(s):	“Metallica”
Description:	The expansion concatenates the original search term <i>Metallica</i> with the name of each band member of Metallica.
Example Output:	{ “Metallica James Hetfield”, “Metallica Lars Ulrich”, ... }.
Expansion Method:	Individual Property Value Expansion (IPVE); with the property specified as <i>hasBandMember</i> ; and retain mode enabled.

Table 4.3: Music Images: Band Members (Example 3)

⁸Flickr’s tag search returned 3 out of 25 relevant results (12-Dec-2008).

Search Term(s):	“Metallica”
Description:	The expansion concatenates the original search term <i>Metallica</i> with keywords related to the artist class in the ontology. Note that this particular expansion depends on the quality of annotations in the target ontology. In this example we assume that <i>Metallica</i> is an instance of classes such as <i>Live-Band</i> , <i>MusicArtist</i> etc.
Example Output:	{ “Metallica Live”, “Metallica Concert”, “Metallica Music”, ... }.
Expansion Method:	Individual Class Expansion (ICE); with retain mode enabled.

Table 4.4: Music Images: Music-related Keywords I (Example 4)

Search Term(s):	“Metallica”
Description:	Depending on the structure of the target ontology, the expansion can also be performed by concatenating the original search term <i>Metallica</i> with the keywords related to the individual’s properties. For example, the property <i>performed-LiveAt</i> contains the keywords ‘performed’ and ‘live’.
Example Output:	{ “Metallica Live”, “Metallica Performed”, ... }.
Expansion Method:	Individual Property Expansion (IPE); with retain mode enabled.

Table 4.5: Music Images: Music-related Keywords II (Example 5)

Film Clips

Example 6 A developer would like to search YouTube for videos clips using the film title “The Wicker Man”. However, *The Wicker Man* could refer to the Iron Maiden song The Wicker Man, the 1973 film, or 2006 film. The developer could add context to the original search terms to help disambiguate between the various different meanings of *The Wicker Man* based on a domain ontology containing film metadata. In this example, including the names of associated people such as the director, writer and starring actors is used by the expansion method.

Search Term(s):	“The Wicker Man”
Description:	The expansion concatenate the original search terms <i>The Wicker Man</i> with the names of staring actors, writer and director.
Example Output:	{“The Wicker Man Edward Woodward”, “The Wicker Man Christopher Lee”, “The Wicker Man Anthony Shaffer”, ... }.
Expansion Method:	Individual Property Value Expansion (IPVE); with the property specified as <i>hasAssociatedPerson</i> , such that in the target ontology <i>hasAssociatedPerson</i> is a super-property of the <i>hasActor</i> , <i>hasDirector</i> and <i>hasWriter</i> properties; and with retain mode enabled.

Table 4.6: Film Clips: Associated People (Example 6)

Pet Photos

Example 7: A developer would like to find images relating to pet dogs. However, they discover that not all images of dogs are tagged with “dog” or “dogs”. The search can be expanded to also search for various different types of dog attempt to improve recall. This example makes use of an ontology that contains a class hierarchy of dog breeds to expand the original search.

Search Term(s):	“dogs”
Description:	The expansion returns a set of keyword groups corresponding to each sub-type of dog in the ontology.
Example Output:	{ “Collie”, “Yorkshire Terrier”, “Dalmatian”, ... }.
Expansion Method:	Sub Class Expansion (SCE); with retain mode disabled.

Table 4.7: Pet Photos: Breeds of Dog (Example 7)

Sports Cars Videos

Example 8: A developer would like to find videos of Italian sports cars. However, videos of cars may not be tagged with such a broad set of tags (“italian sport car”), but may be tagged with the manufacturer and model of car, e.g., “Ferrari 360 Modena”.

Search Term(s):	“italian sports car”
Description:	We can attempt improve the recall in this example by identifying classes associated with the keywords “italian sports car” and expanding this to include keywords related to instances of these classes.
Example Output:	{ “Ferrari 360”, “Lamborghini Diablo”, “Maserati GranTurismo”, ... }.
Expansion Method:	Class Instance Expansion (CIE); with retain mode disabled.

Table 4.8: Sports Car Videos: Sports Car Models (Example 8)

Related Resources

Example 9: A developer would like to find similar resources to those tagged with “Ferrari”. This search expansion shows one way that folksonomy search expansion can be used to suggest similar or related resources, such as those shown on YouTube and other tagging systems.

Search Term(s):	“ferrari”
Description:	The search is expanded to include direct siblings of Ferrari. For example, Ferrari could be a direct sub class of Sports Car and Italian Car. In this case, the user would be returned resources relating to other types of sports car and other types of Italian car.
Example Output:	{“Maserati”, “Aston Martin”, “Mercedes Benz”, ... }.
Expansion Method:	Class Sibling Expansion (CSE); with retain mode disabled.

Table 4.9: Related Resources: Sibling Sports Cars (Example 9)

This section illustrates that folksonomy search expansion may be applied in a number of different scenarios. It is worth noting that each example search expansion targets either precision or recall (at a cost to the other), and a single folksonomy search limitation. However it may be possible to combine expansion methods to reduce the tradeoff between precision and recall. Additionally, using the same technique it would be possible to address multiple folksonomy search limitations. We call this type of search expansion, *hybrid expansion* but leave this to future work.

4.3 Formalisation of Expansion Methods

In this section we present a formal definition of the expansion methods introduced in Section 4.2 using ontology-keyword association (Definition 4.1).

We also show how the following expansion methods could be used to improve the internal structure of folksonomies by constructing additional tag relations following the formal model of folksonomies (Definition 2.11).

4.3.1 Tag Relations

As Mika (2005) indicated, tag relations are often thesaurus relations, such as those specified in the SKOS specification (Miles et al., 2005). Following Hotho et al. (2006), we consider tag relations such as ‘narrower’ \prec_n , ‘broader’ \prec_b and ‘related’ \prec_r .

For example, the following assignment $\langle u1, \text{“cat”}, r1 \rangle$ says user $u1$ uses “cat” to tag the web resource $r1$; the following tag relation $\langle \text{“cat”} \prec_n^{u1} \text{“animal”} \rangle$ says that the tag “cat” is narrower than the tag “animal” for user $u1$.

In addition, there are some properties of the three tag relations:

1. If $\langle tag1 \prec_n^u tag2 \rangle$, then $\langle tag2 \prec_b^u tag1 \rangle$, and vice versa.
2. If $\langle tag1 \prec_r^u tag2 \rangle$, then $\langle tag2 \prec_r^u tag1 \rangle$, i.e., ‘related’ is symmetric.

4.3.2 Search Function

The search function $S(K, expansionMethod)$ is used to search a tagging system using folksonomy search expansion. We define the search function in terms of the expansion function $E(K, expansionMethod)$ (which we formally define in Section 4.3.3), the formal definition of folksonomies (Definition 2.11), and the tag set to keyword set function (Definition 2.10).

Definition 4.2 (Search Function). *The search function $S(K, expansionMethod)$ takes a keyword set K representing a user’s search terms, and an expansion method as inputs. The search function returns a set of resources by searching the tagging system \mathcal{S} using the specified expansion method:*

$$S(K, expansionMethod) = \{r \mid \langle r, T \rangle \in \Phi(K), K' \subseteq tagkw'(T), \\ K' \in E(K, expansionMethod)\}$$

where $\Phi(K)$ is the search function for a tagging system \mathcal{S} , which interprets each of the search terms in K conjunctively as “ k_1 AND k_2 AND ... k_n ” such that $k_1 \dots k_n \in K$; $T \subseteq \mathcal{T}$ and $r \in \mathcal{R}$ for the folksonomy $\mathbf{F} = \langle \mathcal{U}, \mathcal{T}, \mathcal{R}, A, \Theta \rangle$ used by the tagging system \mathcal{S} ; $E(K, expansionMethod)$ is the expansion function; and $tagkw'(T)$ is the tag set to keyword set function.

4.3.3 Expansion Methods

Firstly to simplify presentation we present a single definition of the expansion function with retain mode enabled which can be used for all expansion methods.

Definition 4.3 (Retain Mode Expansion Function). *To enable retain mode for any expansion method, the input keyword set K is appended to each keyword set in the keyword group returned by the expansion function.*

$$\mathbf{E}(K, \text{expansionMethod}, \text{retain}) = \{K' \cup K \mid K' \in \mathbf{E}(K, \text{expansionMethod})\}$$

We can now define the expansion function based on ontology-keyword association (Definition 4.1), without retain mode, for each of the expansion methods introduced in Section 4.2. We also use the definitions of keywords (Definition 2.5), keyword sets (Definition 2.6), and keyword groups (Definition 2.7).

Definition 4.4 (Sub Class Expansion (SCE)). *Sub Class Expansion (SCE) returns a keyword group containing the keyword set $K(E)$ for each sub class E of each class D that is a sub class of the specified class C , where D associated with all keywords in the specified keyword set K .*

$$\mathbf{E}(K, \text{SCE}(C)) = \{K(E) \mid \mathcal{O} \models E \sqsubseteq D, D \sqsubseteq C \text{ and } D \in \bigcap_{\forall k \in K} \mathcal{C}(k)\}.$$

where E is an atomic class and \sqsubseteq refers to the subsumption relationship.

There are a few remarks based on Definition 4.4:

- (i) Note that we use logical entailment (\models) in the above definition, so that every class subsumption $E \sqsubseteq D$ and $D \sqsubseteq C$ entailed by \mathcal{O} is included, and not just the ones explicitly stated in \mathcal{O} .
- (ii) It is easy to limit the above definition to one that allows only direct subsumptions \sqsubseteq_1 .⁹
- (iii) The expansion can be scoped to a specific level of the class hierarchy by specifying class C as a named class, and can be scoped to the ontology as a whole by specifying class C as the top class \top .¹⁰
- (iv) We could construct a set of broader tag relations based on SCE and the *tag to keyword set* function $\text{tagkw}(t)$ (Definition 2.9), since the keyword sets returned by the expansion method for class E are related to narrower concepts (sub classes) than class D (identified by the keywords in K) in the ontology:

$$\{\langle t_1 \prec_b^{u_{\mathcal{O}}} t_2 \rangle \mid \text{tagkw}(t_2) \in \mathbf{E}(\text{tagkw}(t_1), \text{SCE}(\top))\}$$

where $u_{\mathcal{O}}$ is a system user representing the ontology \mathcal{O} .

⁹We use “ \sqsubseteq_1 ” to denote direct subsumption relationships, where $C \sqsubseteq_1 D$ means that class C is a direct subclass of class D in the ontology classification hierarchy.

¹⁰“ \top ” corresponds to owl:Thing, which is the class of all individuals (or “top” class) in \mathcal{O} .

Definition 4.5 (Class Sibling Expansion (CSE)). *Class Sibling Expansion (CSE) returns a keyword set for each sibling class of the classes in $\mathbb{C}(k)$.*

$$\mathbf{E}(K, CSE(C)) = \{K(E) \mid \mathcal{O} \models E \sqsubseteq_1 G, D \sqsubseteq_1 G \sqcap C \text{ and } D \in \bigcap_{\forall k \in K} \mathbb{C}(k)\}.$$

where \sqsubseteq_1 refers to direct subsumption relations.

We could construct a set of related tag relations based on CSE:

$$\{\langle t_1 \prec_r^{u\mathcal{O}} t_2 \rangle \mid tagkw(t_2) \in \mathbf{E}(tagkw(t_1), CSE(\top))\}.$$

Definition 4.6 (Class Instance Expansion (CIE)). *Class Instance Expansion (CIE) returns a keyword set for each instance of the classes in $\mathbb{C}(k)$.*

$$\mathbf{E}(K, CIE(C)) = \{K(o) \mid \mathcal{O} \models D(o), D \sqsubseteq C \text{ and } D \in \bigcap_{\forall k \in K} \mathbb{C}(k)\}.$$

We could construct a set of broader tag relations based on CIE:

$$\{\langle t_1 \prec_b^{u\mathcal{O}} t_2 \rangle \mid tagkw(t_2) \in \mathbf{E}(tagkw(t_1), CIE(\top))\}.$$

Definition 4.7 (Individual Property Value Expansion (IPVE)). *Individual Property Value Expansion (IPVE) returns a keyword set for each individual that relates (via a specified property P) to the individuals in $\mathbb{O}(k)$.*

$$\mathbf{E}(K, IPVE(P, C)) = \{K(q) \mid \mathcal{O} \models P(i, q), C(i) \text{ and } i \in \bigcap_{\forall k \in K} \mathbb{O}(k)\}.$$

We could construct a set of related tag relations based on IPVE:

$$\{\langle t_1 \prec_r^{u\mathcal{O}} t_2 \rangle \mid tagkw(t_2) \in \mathbf{E}(tagkw(t_1), IPVE(U, \top))\}.$$

Definition 4.8 (Individual Class Expansion (ICE)). *Individual Class Expansion (ICE) returns a keyword set for each class of each individual in $\mathbb{O}(k)$.*

$$\mathbf{E}(K, ICE(C)) = \{K(D) \mid \mathcal{O} \models D(i), D \sqsubseteq C \text{ and } i \in \bigcap_{\forall k \in K} \mathbb{O}(k)\}.$$

We could construct a set of related tag relations based on ICE:

$$\{\langle t_1 \prec_r^{u\mathcal{O}} t_2 \rangle \mid tagkw(t_2) \in \mathbf{E}(tagkw(t_1), ICE(\top))\}.$$

Definition 4.9 (Individual Property Expansion (IPE)). *Individual Property Expansion (IPE)*

returns a keyword set for each property that relates to each individual in $O(k)$.

$$\mathbf{E}(K, IPE(C)) = \{K(P) \mid \mathcal{O} \models \{i\} \sqsubseteq \exists P.\top \sqcap C \text{ and } i \in \bigcap_{\forall k \in K} O(k)\}.$$

We could construct a set of related tag relations based on IPE:

$$\{\langle t_1 \prec_r^{u_{\mathcal{O}}} t_2 \rangle \mid tagkw(t_2) \in \mathbf{E}(tagkw(t_1), IPE(\top))\}.$$

4.3.4 Practical Considerations

As we consider entailments in the given ontology \mathcal{O} , the computational cost could be expensive, since the entailment problem in OWL DL is NEXPTIME-complete. It is important to point out that all the required ontology entailments in the above expansion methods are included in the OWL 2 QL semantic approximation of a given OWL 2 DL ontology (Section 2.3.1). This suggests that theoretically, by making use of semantic approximations, the proposed methods can be scalable. This is confirmed by our experiments presented in Section 4.6.2.

Furthermore, if the set of tags returned by search expansions is too large, one could apply some strategies to keep it small, such as by setting a threshold for the association degrees. A typical scenario where our approach could be used is in domain-specific mashup application. In such a system the developer should tailor the exact expansion used to their own ontology, application domain and target tagging system.

4.4 Reasoning Requirements

In this section we describe the ontology reasoning requirements for folksonomy search expansion. The expansion methods described in the previous section rely on ontology entailments. However as discussed in the previous section, OWL DL reasoning comes at a high computational cost. It is still possible to perform search expansion using RDF simple entailment, however this would sacrifice the completeness of the reasoning, and therefore the number of keyword sets returned by the expansion function.

In this section we revisit the Dog Ontology example from Section 2.5 (Figure 4.1) to illustrate the effect of each of the main classes of ontology reasoning which we expect to be used with folksonomy search expansion.

- (1) Dog \sqsubseteq Animal
- (2) Pug \sqsubseteq Dog
- (3) Collie \sqsubseteq Dog
- (4) EnglishShepherd \sqsubseteq Collie
- (5) $\forall eats.DogFood \sqsubseteq$ Dog
- (6) Terrier \sqsubseteq $\forall eats.DogFood$
- (7) Dalmatian \sqsubseteq $\forall eats.PremiumDogFood$
- (8) PremiumDogFood \sqsubseteq DogFood

Figure 4.1: Dog ontology example in DL syntax

We assume the ontology-keyword association functions (Definition 4.1) have been defined for the ontology shown in Figure 4.1 as follows:¹¹

$$\begin{aligned}
K(Animal) &= \{\text{"animal"}\} \\
K(Dog) &= \{\text{"dog"}\} \\
K(Pug) &= \{\text{"pug"}\} \\
K(Collie) &= \{\text{"collie"}\} \\
K(EnglishShepherd) &= \{\text{"english"}, \text{"shepherd"}\} \\
K(Dalmatian) &= \{\text{"dalmatian"}\} \\
K(DogFood) &= \{\text{"dog"}, \text{"food"}\} \\
K(PremiumDogFood) &= \{\text{"premium"}, \text{"dog"}, \text{"food"}\} \\
K(eats) &= \{\text{"eats"}\} \\
\\
C(\text{"animal"}) &= \{Animal\} \\
C(\text{"dog"}) &= \{Dog, DogFood, PremiumDogFood\} \\
C(\text{"pug"}) &= \{Pug\} \\
C(\text{"collie"}) &= \{Collie\} \\
C(\text{"english"}) &= \{EnglishShepherd\} \\
C(\text{"shepherd"}) &= \{EnglishShepherd\} \\
C(\text{"dalmatian"}) &= \{Dalmatian\} \\
C(\text{"food"}) &= \{DogFood, PremiumDogFood\} \\
C(\text{"premium"}) &= \{PremiumDogFood\} \\
P(\text{"eats"}) &= \{eats\}
\end{aligned}$$

4.4.1 OWL 2 DL Search Expansion

Ideally folksonomy search expansion should be performed with the full power of an OWL 2 DL reasoner, since the expansion methods make use of ontology entailments. The advantage of this reasoning approach is that the expansion method will be complete. More precisely, the expansion method will make use of all possible OWL 2 DL entailments when computing the keyword group. As a result the expansion fun can return a larger keyword group by making use of the ontology entailments.

An OWL DL reasoner can infer the following subsumption relationships from the ontology shown in Figure 4.1.

$$Dog \sqsubseteq Animal$$

¹¹The keyword sets have been created by tokenising the entity names in the example ontology. It is known that in practice entity names often provide meaningful labels (Manaf et al., 2010).

Pug \sqsubseteq Dog
 Collie \sqsubseteq Animal
 Collie \sqsubseteq Dog
 EnglishShepherd \sqsubseteq Animal
 EnglishShepherd \sqsubseteq Dog
 EnglishShepherd \sqsubseteq Collie
 Terrier \sqsubseteq Animal
 Terrier \sqsubseteq Dog
 Dalmatian \sqsubseteq Animal
 Dalmatian \sqsubseteq Dog

Using Sub Class Expansion (SCE) called as $E(\{\text{"dog"}\}, SCE(Animal))$ we can expand the input keyword set $\{\text{"dog"}\}$ using OWL DL entailment and return a keyword group (Definition 2.7) containing the following keyword sets:

$$E(\{\text{"dog"}\}, SCE(Animal)) = \{\{\text{"pug"}\}, \{\text{"collie"}\}, \{\text{"dalmatian"}\}, \{\text{"english"}, \text{"shepherd"}\}, \{\text{"terrier"}\}\}.$$

In this example the expansion method is scoped to subclasses of *Animal* in the ontology. The expansion method identifies the class *Dog* by checking all subclasses of *Animal* that have all keywords in $\{\text{"dog"}\}$ using $C(k)$. Then the keyword set is found for each sub class E of *Dog* using $K(E)$, and is added to the keyword group to be returned by the expansion function.

Although ontology reasoning can be used to infer *implicit* subsumption relationships where complex class expressions are used (e.g., the subclass relationship between *Dalmatian* and *Dog* in the ontology shown in Figure 4.1), it cannot account for *missing* subclass relationships in the class hierarchy. For example if the ontology engineer were to omit Axiom 8 ($PremiumDogFood \sqsubseteq DogFood$) then the reasoner would not entail that *Dalmatian* is a subclass of *Dog*, and in turn the expansion function would not be able to return $\{\text{"dalmatian"}\}$ in the keyword group. In this situation it is clear that modelling errors in the ontology can effect the results returned by the search expansion.¹²

As mentioned in the previous section, OWL 2 DL reasoning is computationally expensive and restricts the scalability of the folksonomy search expansion approach, in terms of the size of the ontology used by the expansion method. In the following sections we discuss two reasoning approaches that allow for scalability.

4.4.2 Non-Reasoning Search Expansion

Folksonomy Search Expansion can be performed using the asserted structure of an ontology, i.e., without making use of an ontology reasoner. For scalability and/or performance reasons it is often practical to make use of only the explicitly asserted axioms. This greatly simplifies the computational requirements for the search expansion methods, reducing the ontology

¹²Ontology repairing techniques have been applied to help debug and repair common modelling errors in OWL ontologies, e.g., Parsia et al. (2005) and Lambrix et al. (2012).

entailment checking to RDF entailment checking.

If the ontology is sufficiently simple that the pattern used in the expansion methods matches the asserted ontology structure then the expansion methods are complete without the need for any ontology reasoning. For example, given an ontology $A \sqsubseteq C$, $B \sqsubseteq C$, Sub Class Expansion (SCE), returns the complete set of keywords for an expansion on C . However if there is a second level in the subsumption hierarchy, e.g., $D \sqsubseteq B$, then SCE is no longer complete, since $\mathcal{O} \not\models D \sqsubseteq C$.

For example, without an OWL DL reasoning only the asserted subsumption axioms from the ontology shown in Figure 4.1 are available to the expansion function:

```
Dog  $\sqsubseteq$  Animal
Pug  $\sqsubseteq$  Dog
Collie  $\sqsubseteq$  Dog
EnglishShepherd  $\sqsubseteq$  Collie
```

Using Sub Class Expansion (SCE) (Definition 4.4), again called as $E(\{\text{"dog"}\}, SCE(Animal))$, without reasoning results in the following keyword group:

$$E(\{\text{"dog"}\}, SCE(Animal)) = \{\{\text{"pug"}\}, \{\text{"collie"}\}\}.$$

In this example the reasoner cannot infer that `EnglishShepherd`, `Terrier`, and `Dalmatian` are sub classes of `Dog`. As a consequence the expansion function returns only two out of the five keyword sets returned in the previous example.

Where runtime performance is critical, there are two possible alternatives that allow some sacrifice between performance, and completeness of the results returned by the expansion function. Firstly, materialisation of all or part of the ontology so that the inferences required by the required expansion method can be asserted to the ontology, meaning runtime reasoning is no longer required. This materialisation could be implemented by running the expansion method without an input keyword set K (i.e., so that the expansion targets all entities in the ontology) and storing the result. However, this approach can have some significant drawbacks. The materialised ontology may increase in size (number of axioms) to such an extent as to slow down the query response time beyond what would be considered acceptable for particular application, or the storage space required for the materialised axioms may exceed the available storage capacity.

For example many state-of-the-art ontology reasoners fail to classify very large ontologies, such as the GALEN biomedical ontology (Rector and Rogers, 2006), either due to timeout or memory exhaustion (Motik et al., 2009d; Ren et al., 2010).¹³

When materialising the ontology is not practical, it is possible to use a lightweight reasoner

¹³The reasoner evaluation presented by Motik et al. (2009d) evaluates several state-of-the-art ontology reasoners against a range of well known large and complex real world ontologies. Their evaluation was performed on a system with 2 GB of physical memory, and each classification attempt was allowed to run for a maximum 30 minutes before the timeout was reached.

to perform the search expansion reasoning, which often reduces the drawbacks usually associated with the materialisation approach.

4.4.3 Lightweight Reasoning Search Expansion

The OWL 2 profiles (Section 2.2.1) provide three tractable sub languages of OWL 2 DL, which can be applied to Folksonomy Search Expansion. Each profile poses a set of restrictions on OWL 2 DL in order to ensure that reasoning tasks are tractable. Additionally, each profile is suited to a particular class of reasoning tasks, and therefore particular types of ontology and OWL expressivity.

We have outlined in the following text how each profile could be applied in our Folksonomy Search Expansion approach:

OWL 2 EL is suited to expansion methods where the target ontology has a large number of classes and properties. This language provides and allows for a large variety of class constructors and property axioms. However it is not specifically tailored to reasoning about individuals. OWL 2 EL can be particularly useful for efficiently computing a search expansion using Sub Class Expansion (SCE) (Definition 4.4) and Class Sibling Expansion (CSE) (Definition 4.5).

OWL 2 QL is designed for large scale ABox query answering. Both class and property expressions are heavily restricted in this profile. However OWL 2 QL reasoning can scale to an extremely large number of individuals, where the ontology only requires a relatively simple TBox. This profile is useful for most of the search expansion methods, provided the ontology TBox does not require much expressivity.

OWL 2 RL provides all the property expressions available in OWL 2 DL, but makes strict restrictions on the expressivity of class constructors. These features make this profile particularly suited to ontologies with a rich ABox, requiring reasoning about object property assertions. An OWL 2 RL reasoner would be suited to Individual Property Value Expansion (IPVE) (Definition 4.7) and Individual Property Expansion (IPE) (Definition 4.9).

Approximation of OWL 2 DL Search Expansion

In order to support the wide range of domain ontologies available on the web, we require some sort of trade off between the reasoning and non-reasoning approaches to provide support for the various sizes and complexities of these ontologies. The OWL 2 profiles provide a well defined set of lightweight ontology languages for supporting folksonomy search expansion. However, one single profile is not necessarily suitable to cover the full range of expansion methods and domain ontologies. Materialisation of domain ontologies using an OWL 2 DL reasoner would provide greater support, but as discussed previously, the size of the resulting knowledge base can be prohibitive.

An approximation approach can be applied in Folksonomy Search Expansion to provide users with the benefits of the expressive power of OWL 2 DL, with the efficiency of a tractable

ontology language (Section 2.2.2). One method of approximation is semantic approximation (Section 2.3.1), which compiles an ontology into a tractable ontology language using an OWL 2 DL reasoner. The semantic approximation approach provided by the TrOWL infrastructure (Section 2.3) approximates OWL 2 DL ontologies to an OWL 2 QL approximation, allowing the use of lightweight OWL 2 QL reasoners at query time. Query answering in this situation is guaranteed to be sound and complete for *database-style queries* (Definition 2.3). This would allow for complex ontologies which contain a large number of individuals to be used efficiently in Folksonomy Search Expansion where the pattern used in the expansion method does not contain any non-distinguished variables (a database-style query). This condition holds for all of the expansion methods defined in Section 4.3.3; i.e., the set of ontology entities identified by the expansion method is guaranteed to be sound and complete with respect to the original OWL 2 DL ontology.

4.5 Implementation

This section describes how our folksonomy search expansion approach has been implemented in our system called Taggr. Our implementation makes use of TrOWL (Section 2.3) for ontology keyword association, reasoning platform and ontology repository.¹⁴ Further details of our usage of TrOWL are described in Section 5.1.

4.5.1 Folksonomy Search Expansion with Taggr

Taggr provides an ontology-enabled common interface for folksonomy based systems.¹⁵ It stores a basic tag ontology in TrOWL, capturing the relationships between users, tags and resources in the folksonomy based systems it supports. It provides the functionality of gathering resources and their related tags from the systems that it supports, and updating the tag ontology from time to time. A description of the tag ontology is presented in Appendix D.

Furthermore, Taggr allows users to specify which search expansion method and which reference ontology to use for search expansion. The expansion method and the entities in the reference ontologies are used by the search function (Definition 4.2) to filter tagged resources held in the tag ontology. As the ontologies used by the expansion methods could potentially require the full expressive power of OWL 2 DL, Taggr can make use of the semantic approximation (Section 2.3.1) of the reference ontology for all the entailment checking. An OWL 2 QL reasoner is used for runtime inference. Due to the logical properties of semantic approximation (Definition 2.4), TrOWL can provide sound and complete results for all the needed entailment checking (Pan and Thomas, 2007).¹⁶ As mentioned in Section 4.2, all the proposed search expansion methods are available for application developers to choose, which allows expansion methods to be tailored to the nature of tagging in different domains and types of resources that are to be retrieved.

¹⁴TrOWL is a tractable ontology reasoning infrastructure for OWL 2 DL, developed at the University of Aberdeen.

¹⁵Taggr currently supports YouTube and Flickr.

¹⁶Semantic Approximation guarantees sound and complete results for queries with no non-distinguished variables.

The screenshot shows the Taggr Beta website. At the top, the Taggr logo is displayed with the text 'public beta 0.0.1'. Below the logo is a navigation bar with links for Home, Login, Technology, and Contact us. A large green banner with the text 'semantic tagging for the masses' is prominently featured. The main content area is divided into two columns. The left column, titled 'Welcome', contains a paragraph of text explaining the service, followed by a registration form with fields for Email Address, User Name*, Password, and Confirm Password, and a Register button. The right column, titled 'latest news', shows a date '05/01/2007' and the text 'Launch of public beta for Taggr'. A green horizontal bar is at the bottom of the page.

Figure 4.2: Taggr Beta: <http://www.taggr.org/>

Architecture

Taggr is a web based system designed to provide developers access to folksonomy search expansion. A search request to Taggr consists of the search keywords, the expansion method, and the target ontology URI. A result set is returned to the user as an RDF/XML graph describing the results. The results graph consists of the tagged resource URLs and other metadata such as title, description and tag list.

Figure 4.3 shows the high level architecture of Taggr. The following gives a brief explanation of each component:

Web Service Interface Taggr provides a straight forward RESTful web service interface (Fielding, 2000). The parameters are specified as an HTTP query, and the results are returned as an RDF/XML graph.

Tagged Resource Search This component deals with the search requests to Taggr and performs the search for tagged resources. It sends a request to the Search Expansion component to retrieve the set of expanded keyword groups based on the specified ontology. The tag ontology is then queried to retrieve tagged resources using the keywords specified by the user. The results from the tag ontology are then filtered based on the set of keyword groups returned by the Search Expansion component. We present more details of the algorithm later in this section.

Search Expansion This component takes the original search keywords and the expansion method and performs the search expansion $E(K, expansionMethod)$. The expansion method

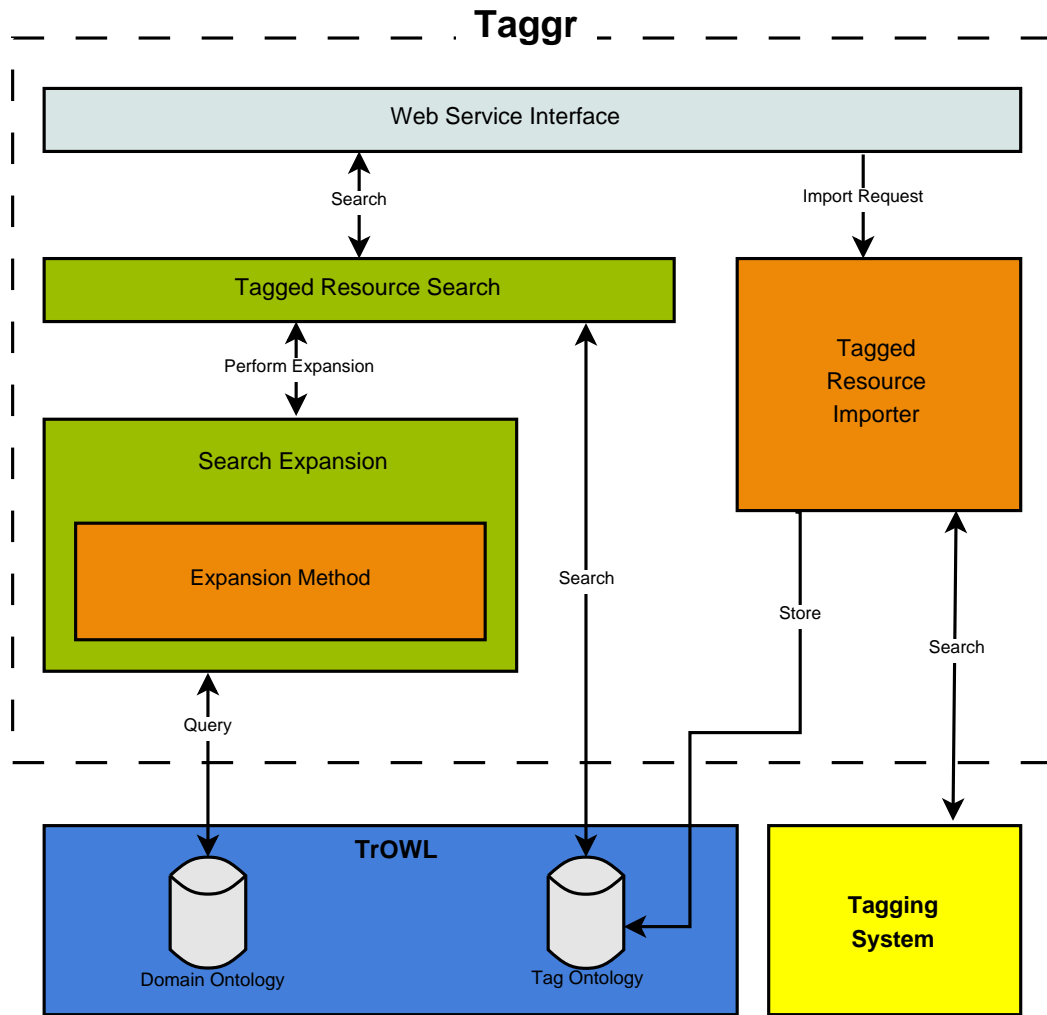


Figure 4.3: Taggr Architecture

generates a SPARQL query which is performed against the specified domain ontology. The results are processed and then returned to the Tagged Resource Search component.

Tagged Resource Importer This component deals with populating the Tag Ontology held in TrOWL. It retrieves a list of search keywords and searches a specified tagging system, without using search expansion. An RDF description of the results is then generated and stored in TrOWL using the tag ontology vocabulary. In practice the tagged resource importer can be used to generate a cache of metadata from the tagging system that can be used by the tagged resource search at query time.¹⁷

TrOWL is used as the ontology repository, query engine and reasoner. It also computes the keywords of each ontology entity in its repository. The query interface accepts SPARQL conjunctive queries.

¹⁷The tag ontology is used to store the metadata from tagging systems required for the tagged resource search. This metadata consists of the the users, tags, and URL of the tagged resources. The tag ontology can be populated in advance or populated at runtime as new requests are made. Caching metadata from tagging systems is usually permitted under the terms of use of the particular tagging system's developer API.

Tagging System represents any tagging system supported by Taggr. The web service provided by the tagging system is used to fetch tagged resources to populate the tag ontology.

The above architecture is implemented as a Java J2EE application and is run on the Apache Tomcat servlet container.^{18,19} The communication between Taggr and its external components (TrOWL and the tagging systems) uses the HTTP protocol.

Search Expansion Algorithm

The two main functions required to execute the tagged resource search are the search function $S(K, expansionMethod)$ (Section 4.3.2), which is implemented by Algorithm 4.1, and the expansion function $E(K, expansionMethod)$ (Section 4.3.3), which is implemented as Algorithm 4.2.

Algorithm 4.1 The search function takes a set of keywords and expansion method as inputs and returns a set of tagged resources with their associated metadata. Firstly the $buildSPARQL_O(K)$ function is used on line 3 to lookup the parameterised SPARQL query to retrieve matching tagged resources from the tag ontology (Appendix D) without expansion. The function fills in the values in the query for each keyword in the keyword set K and returns the SPARQL query. The generated SPARQL query is used on line 4 to query the tag ontology using the function $executeSPARQL_O(query)$. In our implementation the $executeSPARQL$ function uses the OWL 2 QL query engine in TrOWL, which runs in AC^0 in terms of data complexity (Section 2.2.1). Each result initially returned by the tag ontology on line 4 is a triple $\langle r_{RS}, K_{RS}, t_{RS} \rangle \in RS$; where r_{RS} is the tagged resource, K_{RS} is the set of keywords representing the resource's tags, and t_{RS} is the *title* describing the resource. The expansion function (Algorithm 4.2) is called on line 5 to retrieve the keyword groups relating to K using the $expansionMethod$. The algorithm then filters the set of keywords in RS for each tagged resource in the tag ontology identified by keywords in K , against the keyword groups returned by the expansion function in KG . If the keywords for a particular resource are a subset of at least one of the expansion keyword groups, then that resource is added to the results R . The algorithm is deterministic and terminates once all results in the sets RS and KG have been processed. In practice large result sets R could be limited to the *top k* results using a ranking feature. Taggr has the functionality to return a fixed number of arbitrarily selected results. More complex ranking functions could be applied but we regard this as future work.

Algorithm 4.2 The expansion function takes a set of keywords and an expansion method as inputs and returns a keyword group KG . The algorithm generates a SPARQL query based on the specified expansion method $expansionMethod$ and keyword set K and returns a keyword group KG , where each keyword set in KG corresponds to an ontology entity identified by the expansion method after some post-processing has been performed on the result set. The function $buildSPARQL_E(K, expansionMethod)$ on line 3 returns a SPARQL query using a

¹⁸Java Enterprise Edition (Java EE) is the Java enterprise platform for developing web services: <http://www.oracle.com/technetwork/java/javaee/overview/index.html>.

¹⁹Apache Tomcat is a web server for Java EE applications: <http://tomcat.apache.org/>.

Algorithm 4.1 Taggr Search Function

```

1: function  $S(K, expansionMethod)$ 
2:    $R \leftarrow \emptyset$ 
3:    $query \leftarrow buildSPARQL_{\mathcal{O}}(K)$ 
4:    $RS \leftarrow executeSPARQL_{\mathcal{O}}(query)$  ▷ Query the tag ontology
5:    $KG \leftarrow E(K, expansionMethod)$  ▷ Keyword groups from expansion method
6:   for all  $\langle r_{RS}, K_{RS}, t_{RS} \rangle \in RS$  do
7:     for all  $\langle o_{KG}, K_{KG} \rangle \in KG$  do
8:       if  $K_{KG} \subseteq K_{RS}$  then ▷ Filter results from tag ontology
9:          $R \leftarrow R \cup \{\langle r_{RS}, K_{RS}, t_{RS} \rangle\}$ 
10:      end if
11:    end for
12:  end for
13:  return  $R$ 
14: end function

```

lookup table to retrieve the SPARQL query template for the specified expansion method (which are defined in Section 4.3) and filling in the keywords from K . The generated query is used on line 4 to query the domain ontology using the $executeSPARQL_E(query)$ function. In our implementation the $executeSPARQL$ function uses the OWL 2 QL query engine in TrOWL, which runs in AC^0 in terms of data complexity (Section 2.2.1). Each result in RS is a pair $\langle o_{RS}, K_{RS} \rangle$; where o_{RS} is an ontology entity identified by the expansion and K_{RS} is the set of keywords associated with o_{RS} . The results returned by the expansion query are post-processed to ensure that each keyword set K_{RS} differs from the original keywords in K ; if the *retain* flag has been set to *true*, then K is appended to K_{RS} ; finally K_{RS} is trimmed to keep it within a maximum size. The algorithm is deterministic and terminates once each keyword set returned by the expansion method in RS has been processed.

Algorithm 4.2 Taggr Expansion Function

```

1: function  $E(K, expansionMethod)$ 
2:    $KG \leftarrow \emptyset$ 
3:    $query \leftarrow buildSPARQL_E(K, expansionMethod)$ 
4:    $RS \leftarrow executeSPARQL_E(query)$ 
5:   for all  $\langle o_{RS}, K_{RS} \rangle \in RS$  do
6:     if  $K_{RS} \setminus K \neq \emptyset$  then ▷ Ensure each keyword set differs from the original keywords
7:       if  $retain = TRUE$  then ▷ Retain keywords
8:          $K_{RS} \leftarrow K_{RS} \cup K$ 
9:       end if
10:      if  $|K_{RS}| > MAX\_SIZE$  then ▷ Trim large keyword groups
11:         $removeShortestWords(K_{RS})$ 
12:      end if
13:       $KG \leftarrow KG \cup \{\langle o, K_{RS} \rangle\}$ 
14:    end if
15:  end for
16:  return  $KG$ 
17: end function

```

4.6 Case Study and Experiments

In this section we provide a proof of concept evaluation of our approach using our case study semantic mashup application MusicMash2 to show how folksonomy search expansion can be used in practice. We then evaluate the precision (Definition 2.14) of the results returned by the search expansion in this case.

4.6.1 Case Study: MusicMash2

MusicMash2 (Pan et al., 2009a) is a semantic mashup application (Figure 4.4), which is intended to integrate music-related content from various folksonomy based tagging systems and music metadata web services.

As discussed in Section 3.2, search both within and across folksonomy based systems is an open problem. A naïve approach to folksonomy search, such as those provided by most tagging systems,²⁰ results in poor precision in domain specific searches, where more than 20% of the results are not relevant to the query (see Section 4.6.3). MusicMash2 addresses this problem by making use of the folksonomy search expansion methods provided by Taggr. Currently MusicMash2 allows users to retrieve tagged resources relating to music artists, albums and tracks, which are described in a populated domain ontology used by MusicMash2. Individual Property Value Expansion (IPVE) (Definition 4.7) is chosen as the search expansion method for MusicMash2. For example, when searching for videos relating to a specific artist, MusicMash2 will call Taggr to make the following search expansion, $S(artistName, IPVE(foaf:made, owl:Thing))$, i.e., this search expansion will search the tagging system by expanding an artist name (*artistName*) to things related to the music artist in the ontology via the foaf:made property. The goal of this expansion is to reduce the ambiguity of the original search term *artistName* by providing a set of *artistName* + *songTitle* keyword sets.

To use the IPVE search expansion method, developers of MusicMash2 need to have a domain ontology of music metadata, populated with data needed by MusicMash2. Firstly, the Music Ontology (Raimond et al., 2007) is chosen here as it provides the main classes and properties for describing music on the web. Secondly, to populate the Music Ontology, MusicMash2 makes use of web services which provide needed information (each in their own proprietary format), such as, MusicBrainz, Last.fm and DBpedia.²¹ MusicMash2 maps the data from each music metadata web service to the standardised Music Ontology format and submits the populated ontology to TrOWL. The populated ontology can then be exploited by the IPVE search expansion method by Taggr.

The MusicMash2 case study originally (Taylor et al., 2008) followed a “bootstrapping the semantic web” approach (Halpin and Davis, 2007) to produce its corpus of music metadata (Pan et al., 2009a). However since MusicMash2 was first implemented, many projects providing music metadata as RDF/OWL have matured (Raimond and Sandler, 2008; Bizer et al.,

²⁰The YouTube Developer API provides access to YouTube’s search functionality: <http://www.youtube.com/dev/>

²¹MusicBrainz (<http://musicbrainz.org>), Last.fm (<http://www.last.fm>) and DBpedia (<http://dbpedia.org>) web services that provide various types of music metadata.

2009b; Dixon and Jacobson, 2011). We believe that the linked data cloud can currently provide metadata for development of this type of semantic mashup in many domains (Bizer et al., 2011), and envision that it will continue to expand its coverage as the semantic web continues to develop.

4.6.2 Proof of Concept Evaluation: MusicMash2

In this section, we present our proof of concept evaluations (on usefulness and scalability) of Individual Property Value Expansion (IPVE) (Definition 4.7) to address the problem of ambiguity illustrated by the Music Video example in Section 4.2.4. We have used our case study application, MusicMash2, to provide a basis to refine and evaluate the implementation of this particular expansion method. For this reason, we believe that an evaluation of IPVE provides a good practical illustration of how a folksonomy search expansion method can perform in a real world application.

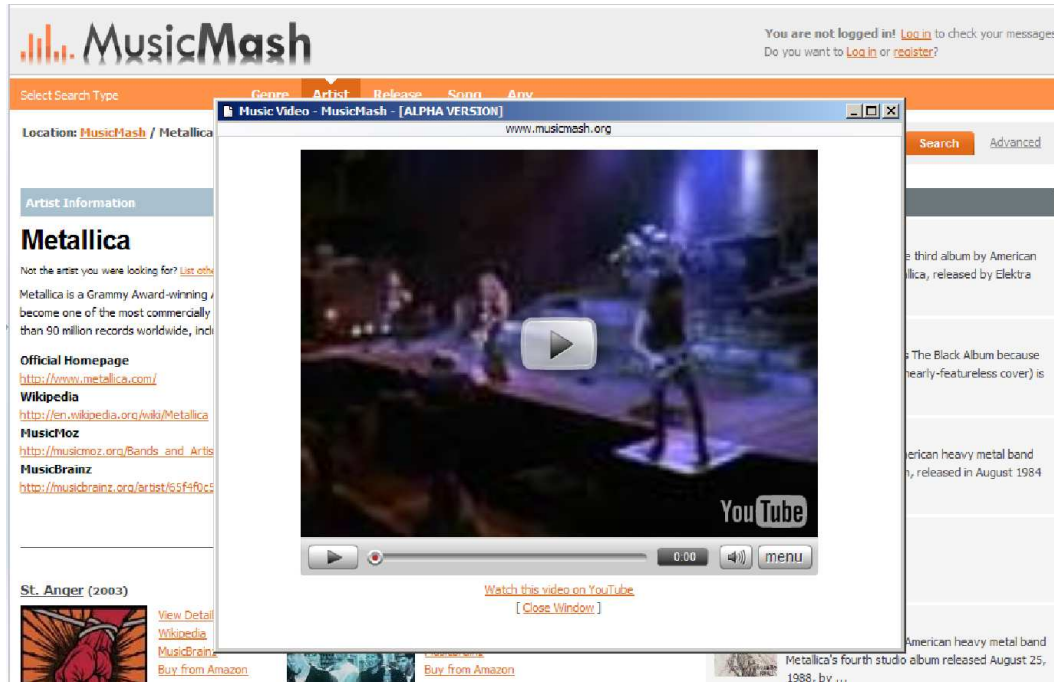


Figure 4.4: MusicMash2 Alpha: <http://www.musicmash.org/>

4.6.3 Usefulness of Folksonomy Search Expansion

Based on our case study (Section 4.6.1), we evaluated the usefulness of our approach by performing identical *top k* searches for music artists with YouTube and Taggr using the IPVE search expansion. We provide this evaluation to show how our system deals with ambiguity in the music domain, which we feel is a typical application of our folksonomy search expansion approach. The set of 15 search terms (Table 4.10) used in the evaluation have been chosen to evaluate how well Taggr with IPVE search expansion handles ambiguity in the music domain when compared with the standard YouTube search facilities. The 15 search terms were based on a selection of music artists used in the MusicMash2 system and reflect a range from high to

low precision when using the tagging system directly.

Search #	Search Term	Search #	Search Term
1	The Beatles	9	Thursday
2	Porcupine Tree	10	Pelican
3	Radiohead	11	Focus
4	Metallica	12	Isis
5	Eels	13	Yes
6	Doves	14	Kiss
7	Finch	15	Jet
8	Strung Out		

Table 4.10: Artist names used in the evaluation.

First of all, we performed top k searches using each system to retrieve the top 20 results. A domain expert then objectively classified each result as either relevant or not relevant with respect to the meaning of the original search. In the case of this evaluation, a result was classified as relevant if it featured the specified artist. The precision (Definition 2.14) of each result set (Figure 4.5) was calculated using Function 4.1, where n_{tp} is the number of relevant results and n_{fp} is the number of irrelevant results in the result set.²²

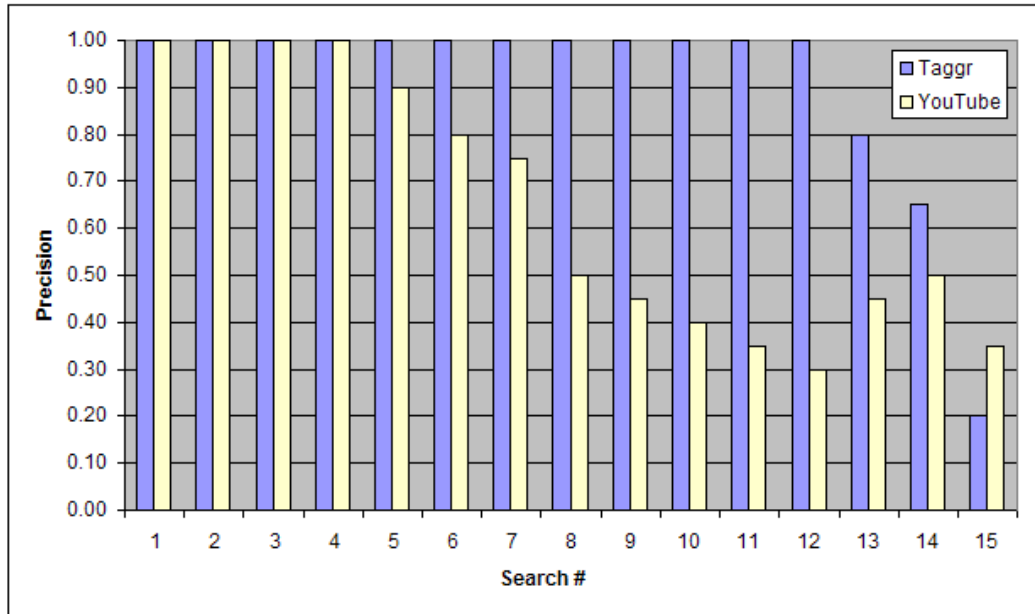
$$P(n_{tp}, n_{fp}) = \frac{n_{tp}}{n_{tp} + n_{fp}} \quad (4.1)$$

To prepare for this evaluation, Taggr’s tag ontology was populated with the top 500 results from YouTube for each of the search terms in Table 4.10. These 500 results typically included both music and non-music related content. Taggr IPVE made use of a populated Music Ontology for search expansion, with the object property specified as foaf:made. More precisely, the search expansion $S(artistName, IPVE(foaf:made, owl:Thing))$ was used (Algorithm 4.1), where *artistName* is the full name of the artist used as the search term. The results from the search function are returned in an arbitrary order, and the top 20 are selected.²³

Figure 4.5 shows that in all but one search, Taggr out performed YouTube in terms of precision. YouTube returned good results for unambiguous artist names but poor results for ambiguous names. In searches 1–4 both YouTube and Taggr return results with perfect precision with these unambiguous artist names. With the more ambiguous artist names in searches 5–12 YouTube performs worse, while Taggr retains perfect precision. In searches 13 and 14, Taggr loses some precision for the highly ambiguous artist names “Yes” and “Kiss”. In the final search, 15, Taggr performs slightly worse than YouTube by 15%. Our initial reaction is that for highly ambiguous artist names and song titles, the artist name plus song title combination

²²Note that we do not measure recall (Definition 2.15) in this evaluation, since classifying all relevant resources in YouTube for each query would be infeasible given the size of the YouTube dataset. However based on the query expansion methodology employed in this experiment we would expect to increase precision at the cost of reducing recall (Section 2.5.1).

²³The experiment does not depend on the order of the results since Taggr does not have a ranking or ordering feature.

Figure 4.5: Result set precision for *top k* search.

used by IPVE is less effective than YouTube’s built in search facilities. A simple approach to addressing this issue would be to attempt to rank the results by examining the complete set of tags for a resource and checking how many are actually matched by the keywords generated in the search expansion.

This evaluation is intended to show how our approach performs in terms of reducing ambiguity in a typical real-world application our approach (Section 4.2.3). We have chosen the IPVE search expansion to illustrate the approach, since this particular expansion method seems to be most widely applicable (Section 4.2.4).

4.6.4 Scalability of Expansion Methods

First of all, it should be noted that the scalability of our implementation depends on the performance of TrOWL. Several evaluations of TrOWL (Pan and Thomas, 2007; Pan et al., 2008) have been made using the Lehigh University Benchmark (Guo et al., 2005), and these have shown that TrOWL is scalable for large ontologies, containing of millions of individuals.

We have performed our own experiments on the performance of IPVE in Taggr by using two different sized datasets. More specifically, Taggr was used to perform the following search expansion $E(artistName, IPVE(foaf:made, owl:Thing))$, i.e., search with some artist name (*artistName*) for everything related to it via the foaf:made property. This experiment was carried out using a typical desktop computer (Table 4.11).

The experiment relies on a populated Music Ontology being present in TrOWL’s ontology repository. MusicMash2 was used to prepare the data for the experiment. A list of artists was created by selecting the most popular artists from the Audioscrobbler web services (using the example user profile from the API documentation.²⁴) The list was then expanded by including

²⁴Audioscrobbler user profiles are available in XML format: <http://ws.audioscrobbler.com/1.0/user/RJ/topartists.xml>

Operating System	Windows XP Pro
CPU	Intel Pentium 4 3.0GHz Dual Core
RAM	2Gb
Hard Disk	80Gb SATA
JVM	Sun JVM version 1.6
Database	PostgreSQL 8.2

Table 4.11: Specifications of test computer

the top 50 similar artists for each of the artists. MusicMash2 then performed a batch process to retrieve information relating to each artist (including other artists) from MusicBrainz, DBpedia and Audioscrobbler web services. This data was used to populate the two versions of the Music Ontology.

Table 4.12 shows the performance of the system computing the result of the expansion function (Algorithm 4.2) using the IPVE expansion method (Section 4.3) for a single query. This experiment was performed with a small dataset of 1,300 artists (30,000 total individuals) and a larger dataset of 5,500 artists (136,000 total individuals). The larger dataset results in an ontology containing over 4,500,000 ground axioms. It is clear to see that further evaluations with larger datasets will need to be carried out to determine how this approach scales with *extremely large* datasets. Given the time (about 2 seconds for the larger dataset; see the T_{IPVE} column in Table 4.12) used for the IPVE search expansion, we believe that with a dedicated database server and further optimisations of our prototype implementation, this application will scale effectively over ontologies containing billions of axioms.

$N_{Artists}$	N_{Albums}	N_{Songs}	N_{Axioms}	T_{IPVE} (ms)
1303	2,531	30,182	$\sim 1,000,000$	155
5521	11,336	136,040	$\sim 4,500,000$	2036

Table 4.12: Performance of Individual Property Value Expansion (IPVE)

4.7 Summary

How to apply semantic web technologies to improve folksonomy based systems and social networks has been a pressing issue for the semantic web community (Mika, 2005). In this chapter we have shown how ontology reasoning can be applied to reuse resources from folksonomy-based tagging systems in semantic web applications, by making use of our lightweight reasoning-based folksonomy search expansion approach (Objective 1). In Section 4.4 we showed how lightweight reasoning could be used to make use both the explicit and implicit information from domain ontologies, while also allowing for efficient and scalable semantic web applications (Objective 1.2).

We also have addressed the problem of search in tagging systems with a general framework of expansion methods, showing that different expansion methods can be useful in potentially different situations. In particular, we used our case study semantic mashup application MusicMash2, to demonstrate the use of folksonomy search expansion and provided a proof of concept evaluation to illustrate its usefulness and scalability. The results of our experiments indicate that the proposed technique can handle very large ontologies and allows folksonomies to be searched more effectively, increasing the precision of folksonomy search results by using our search expansion approach to reduce the ambiguity of search terms (Objective 1.1).

While the experiments in chapter addresses the problem of ambiguity in tagging systems, our search expansion methods can be extended to address the lack of structure and tag variation limitations. For example, users searching for a high level concept such as “animal” may find that resources in tagging systems are not frequently tagged with such a high level concept. In this situation another expansion method such as Sub Class Expansion (SCE) should be used (further examples were presented in Section 4.2.4). Additionally to address the problem of search across tagging systems, different expansions could be used for each target tagging system. For example, in Section 4.2.4 we use different expansions to search YouTube and Flickr for resources related to music artists.

The approach presented in this chapter provides folksonomy search transparently for users of the semantic mashup applications. We have addressed the problem of how to exploit the benefits of ontology to improve folksonomy search, without bothering untrained users of folksonomy-based systems with its rigidity. We have also provided a method to allow users, such as those of MusicMash2, to exploit the benefits of ontology-enhanced folksonomy search without knowledge of either the ontology infrastructure or the related ontologies. Interestingly, for all the search expansion methods, the required ontology entailments are within the expressive power of OWL 2 QL, which allows us to make use of OWL 2 QL semantic approximations (Section 4.4) of given domain ontologies. Although the domain ontologies can very sophisticated, the folksonomy search expansion methods only require the semantic approximations of such ontologies. It has been argued that the web needs lightweight ontologies (Mika, 2005). Our work also suggests that the web can make use of lightweight representation/approximations of (potentially complicated but still useful) ontologies, with clear advantages.

Chapter 5

Configurable Semantic Mashup Applications

In this chapter we show how semantic web applications can be configured to make use of our Folksonomy Search Expansion approach to reuse content from folksonomy-based tagging systems in semantic mashup applications. We achieve this goal by providing a reusable infrastructure which provides a set of configurable folksonomy search expansion methods and an ontology repository and reasoning infrastructure via a set of web APIs.

Our reusable infrastructure is built on a suite of lightweight reasoning based components. We make use of ontology reasoning in our infrastructure so that applications can make use of ontology entailments, and do not have to be concerned with how the assertions are made in the ontology (i.e., applications can make use of both the explicit and implicitly asserted information in ontologies). We use lightweight reasoning (Definition 2.2) in particular so that practical web applications can be both efficient and scalable in terms of the ontology reasoning services used.

In Chapter 4 we presented Folksonomy Search Expansion, our lightweight reasoning-based approach to reusing tagged resources held in folksonomy-based tagging systems in domain-specific web applications using ontologies. We now answer the question of how applications can be built and configured to use our folksonomy search expansion approach.

In the remainder of this chapter, we first present our infrastructure for building semantic mashup applications which can reuse tagged resources based on our folksonomy search expansion approach (Objective 2, Section 5.1). We then show how these applications can be configured to make use of the appropriate expansion method. Finally provide a proof of concept evaluation of our reusable infrastructure in Section 5.5 with a case study in the music domain.

5.1 Reusable Infrastructure for Semantic Mashup

To provide a platform for reusing tagged resources in semantic web mashup applications we propose a reusable infrastructure. This infrastructure provides developers with the tools to build domain-specific semantic mashup applications that can integrate ontology-based semantic mashup applications with content from folksonomy-based systems. Our infrastructure is designed to lower the barrier for developers to create applications that makes use of our folksonomy search expansion approach.

This chapter is presented in the context of MusicMash2, a semantic mashup application in the music domain. MashMash2 is built using our reusable infrastructure (Pan et al., 2009a, 2012), allowing it to be configured to make use of folksonomy search expansion to make use of tagged resources with high precision in the context of a semantic mashup application.

5.1.1 Three Layers of Semantic Tools

Our reusable infrastructure consists of three layers of semantic tools, which can all be accessed by external applications via APIs. The three layers of our infrastructure are shown in Figure 5.1, and consist of the following components:

Layer 1: Semantic Mashup Layer This is the semantic mashup application layer. Applications in this layer can make use of content from Linked Data, Web 2.0 APIs, and folksonomy-based tagging systems. These applications can be configured to use the search expansion methods provided by layer 2. The details of this layer are described in Section 5.4.

Layer 2: Folksonomy Search Expansion Layer This layer represents Taggr, an implementation of our Folksonomy Search Expansion approach (Chapter 4). Taggr deals with folksonomy search requests from layer 1 and makes use of the ontology reasoning infrastructure provided by layer 3. The details of this layer are described in Section 5.3.

Layer 3: Ontology Querying and Storage Layer This layer provides the reasoning and query services required by layers 1 and 2. It also provides the ontology keyword association used by Taggr to perform folksonomy search expansion. The details of this layer are described in Section 5.2.

In the following sections, we present each layer of our reusable infrastructure in more detail.

5.2 Layer 3: Ontology Querying and Storage

The ontology querying and storage layer is provided by TrOWL (Section 2.3). TrOWL is a tractable ontology reasoning infrastructure for OWL 2 DL. We make use of TrOWL in our reusable infrastructure, for ontology querying and storage. First we give an overview of its relevant features in this section, then describe how these features are used in Taggr.

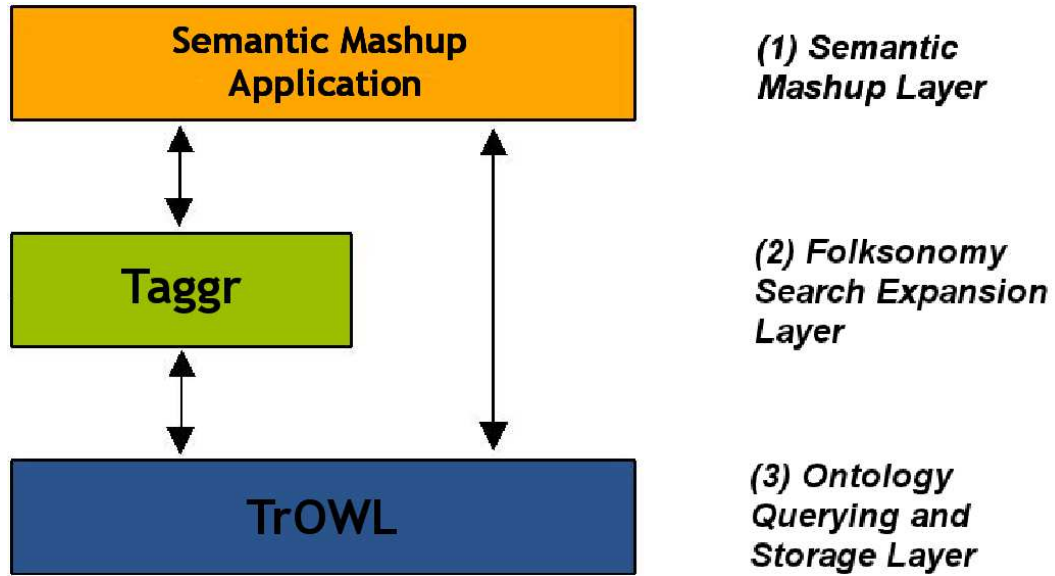


Figure 5.1: Resuable Infrastructure for Semantic Mashup

The main components of TrOWL include an OWL 2 QL reasoner, Quill (Thomas et al., 2010a); an ontology repository, where users can submit and share ontologies; and a SPARQL query engine, which rewrites user queries so they can be executed on the repository.

ONTOSEARCH2 is an ontology search engine that is based on the TrOWL ontology infrastructure. The basic components of ONTOSEARCH2 are a keyword search engine and a fuzzy SPARQL query engine (Pan et al., 2008). An important feature of ONTOSEARCH2 is that it automatically extracts keywords from ontologies (found in values of annotation properties and identifiers) and then associates them with related classes, properties and individuals. Default annotation properties include the `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`, and `owl:versionInfo` properties; we also define the `dc:title`, `dc:description`, and `foaf:name` properties (from Dublin Core (DCMI, 2003) and FOAF (Brickley and Miller, 2010)) as annotation properties. The namespace and identifier of each entity in the ontology are also tokenised to generate keyword metadata. These keywords are weighted based in a similar way to those used by major search engines; further details of the ranking are presented in Section 2.3.2.

The TrOWL SPARQL endpoint allows users to combine conjunctive ABox queries, TBox reasoning queries, and metadata queries (including the keyword association generated by ONTOSEARCH2) in the graph pattern of the SPARQL query. This metadata is used by Taggr to relate keywords to ontology entities when computing the search expansion. Furthermore Taggr makes use of the fuzzy SPARQL query engine to filter keywords using a Fuzzy Threshold query (Section 2.2.3).

Both the semantic mashup layer and the folksonomy search expansion layer use the TrOWL repository and query engine. This allows the folksonomy search expansion layer access to the ontologies used by applications in the semantic mashup layer for search expansion.

5.2.1 Ontology Query Answering

The SPARQL endpoint in TrOWL provides applications with scalable and efficient query answering. To achieve these performance characteristics all OWL 2 DL ontologies submitted to the TrOWL repository are first compiled to an OWL 2 QL representation using an approximation technique called semantic approximation (Section 2.3.1). SPARQL queries then have to be rewritten based on the original input ontology (Calvanese et al., 2005), and also with respect to the structure of the underlying repository in TrOWL. After the rewriting process the queries executed against the repository have the same computational characteristics as SQL queries. Pan and Thomas (2007) carried out evaluations of this query engine and showed that it can scale to millions of individuals on a desktop workstation.

5.2.2 Fuzzy Query Answering

The query engine in ONTOSEARCH2 was previously extended by Pan et al. (2008) to handle fuzzy SPARQL queries (Section 2.2.3). This allows our infrastructure to also handle vagueness in ontologies. For example, in our music mashup MusicMash2 this could be used to handle vague information generated by user interaction with the application as shown in Figure 5.2; where #TQ# declares a threshold query and #TH# is used to specify thresholds for atoms in the query. Therefore this query searches for an instance of `MusicArtist` which is a member of the class `Popular` with degree 0.7, and a member of the class `Active` with degree 0.8.

```
#TQ#
PREFIX music:<http://www.musicmash.org/>

SELECT ?x WHERE {
  ?x a music:MusicArtist ;
    a music:Popular ; #TH# 0.7
    a music:Active .  #TH# 0.8
}
```

Figure 5.2: Querying music data with f-SPARQL

Pan et al. (2008) showed that performance of the fuzzy OWL 2 QL query engine is in most cases close to the performance of the crisp OWL 2 QL query engine.

5.2.3 Fuzzy Ontology Searching

The keyword metadata generated by ONTOSEARCH2 (Section 2.3.2) can also be queried using the fuzzy SPARQL query engine. Additionally metadata queries can be combined with ontology ABox and TBox queries; we refer this type of query as a keyword-plus-entailment searches. For example: *searching for ontologies in which class X is a sub-class of class Y, and class X is associated with the keywords “Jazz” and “Rock”, while class Y is associated with the keyword “Album”*. The search could be represented as by the f-SPARQL query shown in Figure 5.3.

In this query, `kw:jazz`, `kw:rock`, and `kw:album` are representative individuals for keywords “jazz”, “rock”, and “album”, respectively.


```
#TQ#
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX os: <http://www.ontosearch.org/NS/>

SELECT ?x WHERE {
  ?x os:hasKeyword kw:jazz . #TH# 0.5
  ?x os:hasKeyword kw:rock . #TH# 0.7
  ?x rdfs:subClassOf ?y .
  ?y os:hasKeyword kw:album . #TH# 0.8
}
```

Figure 5.3: Keyword search with f-SPARQL

The thresholds 0.5, 0.7, and 0.8 are specified by users using threshold (#TH#) annotations in the query. Although the keyword search queries are supported by our infrastructure, Pan et al. do not provide guidance on how a user can go about selecting these thresholds in ONTOSEARCH2. However based on what we know about the ontology-keyword extraction method used in ONTOSEARCH2 from Section 2.3.2, we can assume that increasing a threshold will filter out less important keywords in the query, i.e., keywords that have been given lower score by ONTOSEARCH2. A user could experiment with the query by first omitting the #TH# annotations from the query, which will perform a keyword search without filtering keywords against degrees. The user could then reintroduce the thresholds to gauge how different thresholds effect the query results.

For example by performing this type of experimentation on the query shown in Figure 5.3, a user would find that increasing the thresholds decreases the number of results returned by the query, and decreasing the thresholds increases the number of results.

Figure 5.4 shows the ONTOSEARCH2 search and query web interface, which provides users with access to the search engine and query engine. The *Search Engine* form allows users to perform a keyword search on to TrOWL repository. The *Query Engine* allows users to evaluate f-SPARQL queries over the TrOWL repository.

The features available in the web interface shown in Figure 5.4 are also available to developers via a RESTful web service (Fielding, 2000). Taggr makes use this web service to access the fuzzy query engine in ONTOSEARCH2 to perform keyword-plus-entailment queries as part of the search expansion.

5.3 Layer 2: Folksonomy Search Expansion

In this section we revisit the implementation of our folksonomy search expansion approach presented in Section 4.5 and present the details of the implementation relating to the reusable infrastructure.

5.3.1 Common Folksonomy Interface

The Taggr system provides a simplified interface to TrOWL to perform useful operations that are related to folksonomy based systems. It stores a tag ontology in the TrOWL repository, which

The screenshot shows the ONTOSEARCH2 web interface. At the top is a blue header with the title 'ONTOSEARCH2' and a navigation menu with links: Home, About, Statistics, Register, Submit a URL, SPARQL, and Help. Below the header, a 'Welcome to ONTOSEARCH2' message states that the project is at the University of Aberdeen Computer Science Department and aims to help facilitate reuse of ontologies. It includes a note to use the form below for keyword-based searches or basic SPARQL queries. The interface is divided into two main sections: 'Search Engine' and 'Query Engine'. The 'Search Engine' section has a text input field for a query and 'Search' and 'Reset' buttons. The 'Query Engine' section has a larger text area for a query, containing a sample SPARQL query, and also has 'Search' and 'Reset' buttons.

Search Engine

Query:

Query Engine

Query:

```
#CTQ#
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX os: <http://www.ontosearch.org/NS/>

SELECT ?X ?Y WHERE {
  ?Y os:hasKeyword "person" . #FT# 0.6
  ?X rdfs:subClassOf ?Y .
}
```

Figure 5.4: ONTOSEARCH2 Search and Query Interface

is used to capturing the relationships between users, tags and resources in the folksonomy based systems that it supports.¹

Taggr allows users to provide a set of tagged resources and their related tags to be added to the tag ontology in the TrOWL repository. A web service and traditional web interface are provided so that users can interact with the tag ontology without having to understand the internal representation used by the system. The web interface shown in Figure 5.5 allows users to log in to the Taggr system and provides a graphical web interface to access the search expansion web service. The web service interface provides access to Taggr’s features via a RESTful API which can be used by mashup applications. Java-based applications can make use of the client API implementation provided by Taggr.²

5.3.2 Evaluating Search Expansion

Taggr uses the f-SPARQL query engine provided by ONTOSEARCH2 to evaluate the search expansion function $E(K, expansionMethod)$. The expansion method used in our folksonomy search expansion experiments in Section 4.6, which used to expand an initial search for an artist name to include song titles, is implemented using an f-SPARQL query. For example, $E(\{metallica\}, IPVE(foaf:made, owl: Thing))$ (Definition 4.7) could be implemented using the fuzzy threshold query shown in Figure 5.6.

The thresholds 0.6 and 0.7 are generated by the expansion method implementation in Taggr, which means that in general we do not expect users to specify these thresholds. However, the Taggr API provides the functionality to optionally override the default thresholds used

¹A description of the tag ontology is presented in Appendix D.

²An overview of the Taggr client API is presented in Appendix E.

The screenshot shows the Taggr web interface. At the top, there is a navigation bar with links for Home, Login, Technology, and Contact us. Below this is a large green banner with the text "semantic tagging for the masses". The main content area is divided into two columns. The left column is titled "Welcome" and contains a paragraph about the beta version of Taggr, a tool for integrating semantic tagging systems. It mentions that the service is being developed at the University of Aberdeen Computer Science Department and is using Semantic Web technologies such as RDF and OWL. It also mentions that users can consolidate and organize their tags from sites such as YouTube, Flickr, and Delicio.us. Below the text is a registration form with fields for Email Address, User Name*, Password, and Confirm Password, and a Register button. A footnote at the bottom of the form states: "* User name cannot contain spaces, or any non alphanumeric characters. This will be used to identify your tags and tagclouds." The right column is titled "latest news" and contains a date "05/01/2007" and a headline "Launch of public beta for Taggr".

Figure 5.5: Taggr Web Interface

in the expansion method. This feature allows the output of the expansion function to be experimented with by adjusting the thresholds, in the same way as the ONTOSEARCH2 fuzzy keyword search described in Section 5.2.3.

5.4 Layer 1: Semantic Mashup

The semantic mashup layer can be implemented by an application that makes use of the components of the reusable infrastructure. In what follows we describe in detail how the Semantic Mashup Layer relates to the other layers presented in this chapter.

Traditionally, a mashup is a kind of data-integration application, where information regarding input queries is collected from different sources, in different media, and presented to users in a coherent manner. A semantic mashup is an extension of this idea, along the lines of integrating linked data from various sources at an application level to be presented in a single web application.

The Linked Data Principles (Bizer et al., 2009a) outline a set of rules or guidelines that make data integration a natural part of Linked Data. As a result, the effort involved in the data integration tasks that may have been a time consuming part of building traditional Web 2.0 mashup applications are greatly reduced with linked data.

However, the data integration problem arises again when a semantic mashup application, based around linked data, is needed to reuse data from traditional web services. In the previous

```

#TQ#
PREFIX os: <http://www.ontosearch.org/NS/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?k
FROM <http://www.musicmash.org/ontology/artists.owl>
WHERE {
    ?y os:hasKeyword "metallica" . #TH# 0.6
    ?y foaf:made ?x .
    ?x os:hasKeyword ?k . #TH# 0.7
}

```

Figure 5.6: Expansion Method keyword retrieval with f-SPARQL

chapter, we presented Folksonomy Search Expansion, which aims to address this application-level integration problem by presenting a method to reuse data from folksonomy-based tagging systems in semantic web applications.

Taggr, our folksonomy search expansion implementation, provides the benefit of a common interface to these tagging systems for semantic mashup applications. This common interface allows developers of semantic mashup applications to reuse tagged resources without the burden of directly addressing the search limitations (Section 4.1) or creating mappings for each tagging system supported by Taggr.

In the remaining section we present details of MusicMash2, a semantic mash up application based on our reusable infrastructure.

5.4.1 Integrating Folksonomies and Semantic Mashup

MusicMash2 is an ontology-based semantic mashup application, which is intended to integrate music related content from various folksonomy based tagging systems, linked data, and music metadata web services, in a single mashup application. MusicMash2 provides functionality for users to search for tagged images and videos that are related to artists, albums, and songs. MusicMash2 corresponds to the semantic mashup layer in our three layer architecture.

In order for this layer to take full advantage of Taggr’s search expansion methods, MusicMash2 uses the TrOWL ontology repository to store a populated Music Ontology (Raimond et al., 2007), containing the data used in the mashup. This allows both Taggr and MusicMash2 to be able to access the same data for use in the search expansion and for presentation to the user in the mashup application.

A search expansion method should be configured by the developer for each semantic mashup application (Section 4.2.3). In the case of MusicMash2, we use Individual Property Value Expansion (IPVE) (Definition 4.7), with the property specified as foaf:made. This particular expansion is tailored to return high precision results for artist name searches on YouTube. We present more details of how this expansion was chosen in our case study in Section 5.5.

For other semantic mashups, the expansion method can be tailored to suit the application’s domain, ontology and tagging system. A number of example sceneries are presented in Section 4.2.4 to illustrate a some typical search expansions that can be configured for use in

semantic mashups in many domains.

5.4.2 MusicMash2 Architecture

The high level architecture of MusicMash2 can be used as a template for building other semantic mashup applications using our reusable infrastructure. Figure 5.7 shows the flow of data between the main components of this architecture. It is built using a typical Model View Controller (MVC) pattern, commonly used in web applications.³ In this particular application we have used Java J2EE, although this architecture can be applied to most web application development platforms. We outline the main components below.

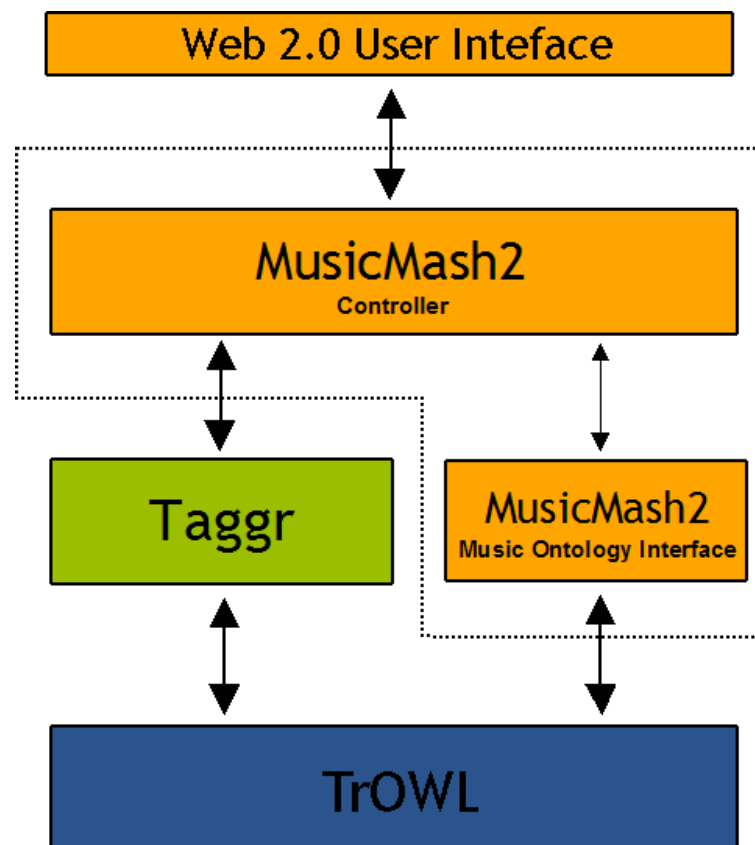


Figure 5.7: MusicMash2: Architecture

Web 2.0 User Interface This component deals with presenting music-related content to the user using JSP, HTML, CSS and JavaScript. It is a typical Web 2.0 style browser-based interface, which does not depend on any specialisation for the semantic web technologies used by the other components.

MusicMash2 Controller This component deals with delegating requests from the user interface component to Taggr and the Music Ontology Interface. A client version of the Taggr API is used to send folksonomy search requests to the Taggr system; the configured search expansion method is specified at this point. This component also deals with requests for

³MVC is an architectural design pattern that focuses on separating presentation and domain logic.

data from the music ontology held in TrOWL. The interaction between TrOWL and MusicMash2 is abstracted by the Music Ontology Interface component.

MusicMash2 Music Ontology Interface This component replaces the Data Access Object (DAO) component found in traditional relational database driven application.⁴ The ontology interface accepts typical DAO requests, e.g., *fetchArtist(artistName)*, and returns a domain object to the controller to be rendered by the user interface. Essentially this components creates SPARQL queries based on *fetch* requests from the controller, sends the query to TrOWL, then processes the returned result set.

Taggr This component is the Taggr system, which is an external service to MusicMash2 and implements our folksonomy search expansion approach. A simple client API is provided for semantic mashup applications to make search expansion requests.⁵

TrOWL This component represents the TrOWL ontology infrastructure, which is also an external service. Again a simple client API is provided to allow applications to send SPARQL queries and receive result sets (much like JDBC in Java (JSR-000221, 2011)).⁶ TrOWL is used directly by MusicMash2 to query the music ontology, and indirectly via Taggr to search for tagged resources.

In Section 5.5 we present a case study that further details the development of MusicMash2 from the perspective of a Web 2.0 developer.

5.4.3 Configuring Folksonomy Search Expansion

In order to make use of the folksonomy search expansion layer (Section 5.3), semantic mashup applications must be configured in the appropriate way, to make use of the expansion methods provided by Taggr. There are three key parameters in the configuration, which are passed to Taggr when making a folksonomy search expansion request. We define the folksonomy search expansion configuration in terms of these three key parameters as follows.

Definition 5.1 (Folksonomy Search Expansion Configuration). *The elements of a folksonomy search expansion configuration are: the expansion method; the target ontology; and the target tagging system. These elements are as follows:*

Expansion Method *The expansion method used to perform the folksonomy search expansion. The expansion method should be one that is implemented by Taggr. This expansion method is used as a parameter to the expansion function (Algorithm 4.2).*

Target Ontology *The target ontology used by Taggr to evaluate the expansion method (Algorithm 4.2). This ontology should already be present in the TrOWL repository.*

⁴A Data Access Object (DAO) is a software component that provides a layer of abstraction for data access to a database, or some other persistence mechanism.

⁵An overview of the client API is provided in Appendix E.

⁶JDBC provides an API for database access in Java.

Target Tagging System *The target tagging system to be used by Taggr to evaluate the search expansion (Algorithm 4.1). This tagging system should be specified as one of the tagging systems supported by Taggr.*

For example MusicMash2 is configured with the IPVE expansion method, the Music Ontology and the YouTube tagging system.

5.5 Case Study: Mashing Linked Music Data

In this section, we describe a concrete scenario illustrating how the reusable infrastructure presented in Section 5.1 could be used to build a semantic mashup application, using typical web development and mashup techniques. Let's consider the story of "Emma", a keen web developer with an interest in Web 2.0 and new web technologies in general.



Figure 5.8: MusicMash 1.0 Screenshot

A Web 2.0 Application

Emma's interest in Web 2.0 had grown from her interest in music. She enjoys browsing tagging websites such as YouTube and Flickr to view video and image content of her favourite music artists, Wikipedia to view discographies and biographies, and Amazon to view and purchase related products. However, she found that it was inconvenient to browse so many different websites to find content related to a single artist.

Meanwhile, she had been reading about web services and mashup as part of her interest in web development. After reading a few articles on the web, Emma decided that she could address

this inconvenience by building her own mashup application. The goals of this application were to combine music-related resources – videos, images, biographies and discographies – into a single website.

Emma also decided that if her application were to be of use, it would have to provide accurate search results in a timely fashion. Emma named her new web application “MusicMash” and began work on the project. This first version of MusicMash is shown in Figure 5.8.

Searching Folksonomies

To retrieve video and image content for her new site, Emma made use of the public web service APIs provided by YouTube and Flickr. She quickly developed the first prototype. This version allowed users to search based on the artist name. MusicMash used the artist name when making calls to the YouTube and Flickr APIs to retrieve videos tagged with each word from the artist’s name.

Emma soon found that this early version of MusicMash suffered from the major drawback that artist names are often ambiguous search terms in YouTube and Flickr. For example, when she searched for Focus (the dutch Progressive Rock band), only 5 out of the top 20 results returned by YouTube were relevant to that artist.

She noticed that users on YouTube often tagged music videos with both the artist name and song title. She soon realised that including a song title with the artist name resulted in much more relevant search results. Emma then extended MusicMash to populate a database of music metadata retrieved from the MusicBrainz web service. Using this metadata, Emma could extend MusicMash to automatically expand a simple artist name search, to an *artist name plus song title* search, for each song by that artist. This search expansion technique resulted in an impressive increase in the precision of the search results. However, the volume of API calls needed for the search expansion resulted in an unacceptable amount of time for the search to return.

Emma was now happy with the accuracy of MusicMash searches but disappointed by the effort to implement the search expansion technique for each type of search she needed.

The Switch to Taggr

Emma learned of a system called Taggr that provides a number of expansion methods for various tagging systems that can be accessed via a web service API. The Taggr API allows Emma to input the original search terms from the user and some extra parameters to specify how the search expansion should be performed.

More specifically, the parameters indicate whether video or image resources should be returned; what the input search term(s) should identify, S (a music artist in this case); where to find the extra keywords for the search expansion, T (a song title in this case); and how S and T are related, P (a music artist is the creator of a song). Taggr uses OWL DL ontologies to represent its metadata internally. The parameters S and T should be specified as OWL classes and P as an OWL property.

Emma decided to redevelop MusicMash using the Taggr API, rather than accessing YouTube and Flickr directly. However, Emma did not know which OWL class was used to identify music artists and song titles on the web. She followed a link from the Taggr website to ONTOSEARCH2. She then made use of ONTOSEARCH2's ontology search engine to find out which ontologies contained resources relating to music. Emma typed "music" into the ONTOSEARCH2 search engine and one of the first results returned was from the Music Ontology. Emma compared the Music Ontology with other music schemas available on the web from Schema.org (schema.rdfs.org, 2011) and the Linked Open Data cloud (Cyganiak, 2011). After some investigation Emma decided that the vocabulary from the Music Ontology could be used to describe music artists, albums and songs in her dataset.

She also realised that the Music Ontology also contained information that could be used to perform search expansions for other types of query. For example, a genre name could be expanded to include artists in that genre.

She then set about trying some searches on Taggr using the classes `mo:MusicArtist` and `mo:Track`, related via the property `foaf:made`; allowing her to replicate the search expansion from MusicMash. She first used Taggr to check which new keywords were generated by its search expansion. Emma tried the keyword "Coldplay" and was surprised to see that Taggr did not provide any new keywords. She then searched ONTOSEARCH2 directly for "Coldplay" and again, no results were returned. Emma realised that she would have to provide the Music Ontology individuals herself in order for the search expansion to work correctly.

From Relational Databases to Ontologies

Since ontology individuals are required by Taggr to replicate the MusicMash search expansion. Emma decided to drop her database of music metadata in favour of Music Ontology instances stored in the TrOWL repository. TrOWL's submission and query engine provided the tools that she needed insert new individuals into the repository and query against them. Emma decided to populate her ontology using web services that can be easily linked. More specifically, using MusicBrainz API for basic artist, album and song information she could extend the metadata with other sources that referred to MusicBrainz identifiers, such as Last.fm and DBpedia. This new version of MusicMash was enhanced with music metadata from the Music Ontology and named MusicMash2. Figure 5.9 shows an artist page from MusicMash2, which displays data from the Music Ontology and videos from Taggr.

The final two problems which were left for Emma to address occurred on the occasions that there is no Music Ontology instances relating to a user search or where there were insufficient resources returned by Taggr. She decided that for any search for which MusicMash2 did not immediately return more than five results to the user, a request would be made to Taggr to populate its tag ontology with more resources from tagging systems in its library. Taggr would then send requests to its supported tagging systems to retrieve the first 500 results based on the original search term(s). Similarly when MusicMash2 returns no individuals in the Music Ontology relating to the search, it initiates a background task to retrieve the required information

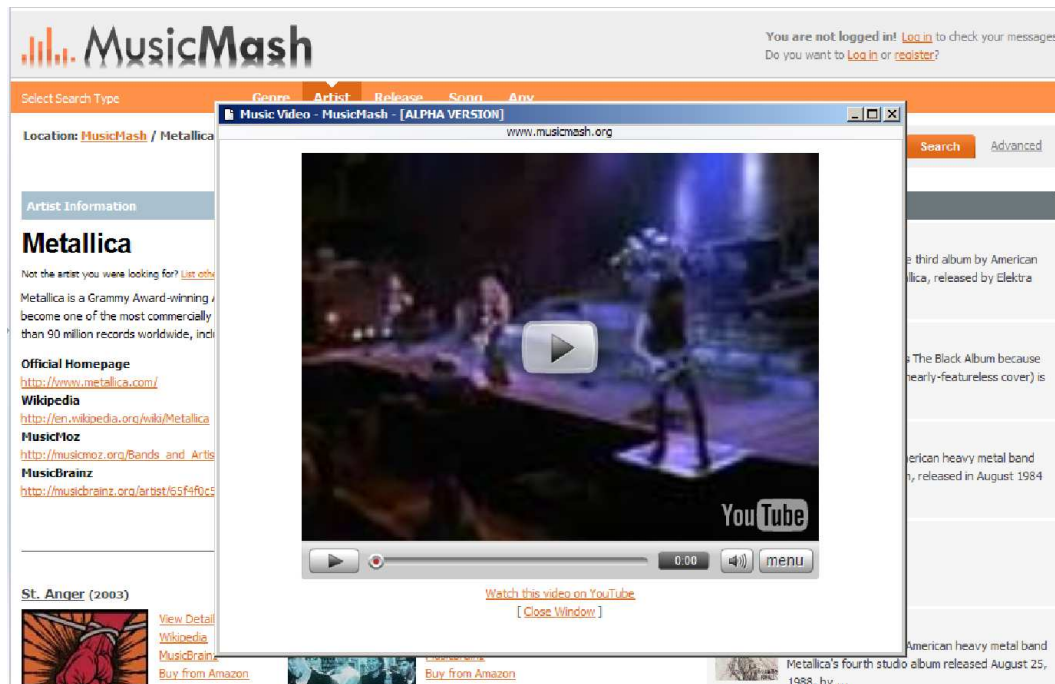


Figure 5.9: MusicMash2 Screenshot

from the web services in its library. The advantages of this method is that information relating to previously unknown artists can be added automatically to the Music Ontology in TrOWL and tag ontology in Taggr, following a “bootstrapping the semantic web” approach (Halpin and Davis, 2007). Emma decided that it was an acceptable trade-off that the first user should wait for the information to be retrieved, in order that future searches would return in a more acceptable amount of time.

5.5.1 Example User Scenario

A typical scenario for MusicMash2 can be illustrated by a user searching for information related to an artist. The user first enters the name of the artist into the search box. On completion of a successful search, MusicMash2 displays information related to the artist. This includes a short abstract from DBpedia, the artists discography and links to the artists homepage and Wikipedia articles. The user can also select the Video Gallery tab to display videos relating to the current artist. The Video Gallery makes use of Taggr to return high precision search results for related videos.⁷

We can show the usefulness of the two main components of our infrastructure by illustrating the benefits of a semantic mashup such as MusicMash2 over a standard Web 2.0 application such as the original MusicMash system. Firstly, the TrOWL infrastructure provides a repository of ontologies allowing an ontology generated by one application to be easily reused by other applications in the same domain. In our scenario, MusicMash2 is continually contributing to the Music Ontology stored in the TrOWL repository. In the original MusicMash RDBMS approach,

⁷An example artist page can be viewed at the following URL: <http://www.musicmash.org/artist/Metallica>.

all information generated by the application is locked in the application's own proprietary database and cannot be easily reused by third party applications. Secondly, Taggr's folksonomy search expansion methods provide a powerful platform for domain-specific applications to retrieve multimedia resources from tagging systems. While Taggr can be used by standard Web 2.0 applications, the information used by Taggr to perform the folksonomy search expansion is retrieved from the ontologies stored in the TrOWL repository. MusicMash2 ensures that Taggr can always find resources by keeping the music ontology up to date.

5.6 Summary

In this chapter we have described how semantic mashup applications can be built to use our three layer semantic infrastructure and configured to use folksonomy search expansion to reuse content from folksonomy-based tagging systems in ontology-based applications (Objective 2). In this scenario we make use of the scalable reasoning services provided by the TrOWL reasoning infrastructure both in semantic mashup application and in our folksonomy search expansion. Our infrastructure makes use of lightweight ontology reasoning, to provide practical ontology reasoning services to semantic mashup applications.

We also exploit fuzzy reasoning to handle the keyword matching used in Taggr's folksonomy search expansion implementation. Both the standard and fuzzy query engines are based on a lightweight OWL 2 QL query engine, which makes use of the semantic approximation provided by TrOWL to reduce heavyweight OWL 2 DL reasoning tasks to lightweight OWL 2 QL reasoning.

In our case study we described the MusicMash2 system. This is an application which combines semantic web technologies, freely available ontologies, open sources of data, and Web 2.0 web services from Flickr and YouTube. We have shown how the combination of these technologies can make folksonomy searching more accurate within a specific domain – particularly in areas where a simple keyword search is too generic to produce relevant results. The second benefit of our approach is that by combining open ontologies with information retrieved from proprietary knowledge bases, we increase the access to this information through open interfaces. Since TrOWL is publicly accessible through a standardised SPARQL interface (Prud'hommeaux and Seaborne, 2008), it is possible for other applications to be built on top of the ontologies generated by MusicMash2. By using this technique to add value to existing folksonomy-based websites, we provide a platform on which to stimulate further development and/or population of domain ontologies.

Chapter 6

Configuration of Linked Data Content Management Systems

Traditional web Content Management Systems (CMS) are widely used and well understood by web users for authoring web content. These tools could greatly benefit the semantic web in their reuse to allow web users to generate Linked Data using familiar tools (Section 1.1.2). In this chapter we propose Linked Data Content Management Systems (Linked Data CMS), an approach to configuring existing CMS software to generate web sites based on ontology classes, properties, and individuals.

A Linked Data CMS performs similar operations to those of a traditional CMS but whereas a traditional CMS uses a data model of content types stored in some relational database back end, a Linked Data CMS deals with performing CRUD operations on linked data held in a triple store, in the context of a content management system.

The Linked Data CMS provides users with the tools to manage linked data in a typical CMS setting, allowing repositories of linked data to be managed using the same techniques as traditional blogs and wikis (Objective 3, Section 6.2). This approach is based on our work published in Taylor et al. (2013).

In our Linked Data CMS approach we define a mapping between certain OWL classes (plus some additional metadata), and entities in a traditional CMS. We call this mapping *class based browsing* because the ontology classes are the central entities in our mapping. In our class based browsing approach the resulting CMS contains a set of page templates for the ontology classes specified in the configuration, e.g., a *person page template* could be configured to display instances of a *Person class*. These page templates are then used to create individual pages for each instance of the relevant ontology class, e.g., a page about Tim Berners-Lee could use the *person page template*.

We also introduce *Reasoning-Driven Configuration*, which configures the class based browsing and Linked Data CMS based on a domain ontology. In reasoning-driven configuration, an ontology reasoner is used to automate the process of configuring the CMS. Ontology reasoning is used so that both explicitly asserted and inferred instances of browsing classes and properties can be used to construct the pages and links between pages in the CMS. This process can be highly scalable and efficient by using lightweight ontology reasoner (Definition 2.2) for

entailment checking, meaning that the approach can potentially scale to very large ontologies.¹

We have implemented a prototype of our class based browsing and reasoning-driven configuration approach using the Drupal CMS. In the remainder of this chapter we refer to this implementation as the Drupal Linked Data CMS. We provide a proof of concept of the approach with a case study for the CURIOS project (Beel et al., 2013; Tait et al., 2013), which deals with managing cultural heritage linked data using the Drupal Linked Data CMS.

In the remainder of this chapter, we first present our formalisation of traditional web content management systems in Section 6.1, and then we present our Linked Data CMS approach in Section 6.2, for integrating linked data with these traditional content management systems. We then provide details of our Drupal-based Linked Data CMS implementation in Section 6.3, and present a proof of concept of our approach in Section 6.4 with a case study in the cultural heritage domain.

6.1 Formalisation of CMS Entities

Here we propose a straightforward way of formally describing the entities of a traditional content management system. The idea is to capture the features of the CMS that can be used to represent ontological entities in our Linked Data CMS.

Definition 6.1 (Content Management System (CMS)). *A CMS is a tuple $\mathcal{S} = \langle T, F, R, P \rangle$ where*

- *T is a set of page templates, which define the structure of particular types of page in the CMS (i.e., sets of fields and relationships);*
- *F is a set of fields, which are used to display literal values;*
- *R is a set of relationships, which define links to another pages in the CMS;*
- *P is a set of page instances, which use a page template to display some specific information.*

Fields and Relationships

Fields are used to display literal values, such as name or date of birth. Relationships are used to provide links to other page instances in the CMS.

Definition 6.2 (Field). *A field is a single $f = \langle f_{lab} \rangle \in F$, where f_{lab} is a human readable label for the field.*

Definition 6.3 (Relationship). *A relationship is a pair $r = \langle r_{lab}, r_T \rangle \in R$, where*

- *r_{lab} is the relationship label;*
- *$r_T \in T$ is a template for the pages for the objects of the relationship.*

¹Depending on the lightweight reasoning approach used, the Linked Data CMS could be implemented to support complex TBoxes, very large ABoxes, or highly efficient runtime query answering (Section 2.2.1).

Page Templates

A page template consists of a set of fields, and a set of relationships. This entity is used as a template for creating page instances in the CMS (i.e., a page instance has the same fields and relationships as its page template).

Definition 6.4 (Page Template). *A page template for a CMS $\langle T, F, R, P \rangle$ is a tuple*

$$t = \langle t_{lab}, T_F, T_R \rangle \in T$$

where

- t_{lab} is the page template label;
- T_F is a set of fields, such that $T_F \subseteq F$;
- T_R is a set of relationships, such that $T_R \subseteq R$.

Page templates also contain the information on how to render page instances, which would typically consist of an HTML and CSS template to controlling the page layout, however we do not include the display concerns here.

Page Instances

A page instance is an instantiated page template, for a particular set of field and relationship values.

Definition 6.5 (Page Instance). *A page instance for a CMS $\langle T, F, R, P \rangle$ is a tuple*

$$p = \langle p_{lab}, p_T, P_F, P_R \rangle \in P, \quad \text{where}$$

- p_{lab} is the page title;
- p_T is a page template $t = \langle t_{lab}, T_F, T_R \rangle \in T$;
- P_F is a set of pairs $\langle f_i, f_{val} \rangle$ where $f_i \in T_F$ and f_{val} is a literal value (the field value);
- P_R is a set of pairs $\langle r_i, r_{val} \rangle$, where $r_i \in T_R$, $r_{val} \in P$ and the page template of r_{val} is the same as the template specified in r_i .

6.1.1 CMS Entity Example

We can now show how the CMS entities defined in the previous section could be used to describe a CMS about people. This example is for a CMS \mathcal{S} containing pages about a person *May*, with two children *Pam* and *Chris*. It can be described using CMS entities as follows:

$$\mathcal{S} = \langle \{person\}, \{name, dob\}, \{parentOf\}, \{may, pam, chris\} \rangle;$$

where *person*, *name*, *dob*, *parentOf*, *may*, *pam*, *chris* are entities defined below.

The page template *person* is defined using the fields and relationships in \mathcal{S} , and the page

May's Page

- **Full Name:** May
- **Date of birth:** 1972-01-05
- **Children:** [Pam](#), [Chris](#)

Figure 6.1: Example CMS page displaying details of a person

instance *may* would be populated with the data about May:

$$\begin{aligned}
 person &= \langle \text{"Person Page"}, \{name, dob\}, \{parentOf\} \rangle; \\
 may &= \langle \text{"May's Page"}, person, may_F, may_R \rangle; \\
 may_F &= \{ \langle name, \text{"May"} \rangle, \langle dob, \text{"1972-01-05"} \rangle \}; \\
 may_R &= \{ \langle parentOf, pam \rangle, \langle parentOf, chris \rangle \};
 \end{aligned}$$

where *pam* and *chris* are page instances for Pam and Chris respectively, which are defined using the *person* page template in a similar manner as for *may*. The fields and relationships are defined as:

$$\begin{aligned}
 name &= \langle \text{"Full Name"} \rangle; \\
 dob &= \langle \text{"Date of birth"} \rangle; \\
 parentOf &= \langle \text{"Children"}, person \rangle.
 \end{aligned}$$

An example web page for this page instance is shown in Figure 6.1.

6.2 Linked Data Content Management

Our approach focuses on browsing ontology entities in a CMS, based on ontology classes. The idea is that a particular set of classes from an ontology's class hierarchy is selected to provide *views* of their instances in the CMS; we call this approach class based browsing. An ontology reasoner is used to infer implicit class and property membership when generating page instances in the CMS based on class based browsing.

The use of reasoning to generate the CMS means that the classes and relationships selected for class based browsing can make use of their inferred instances in the target ontology, and therefore the selection of ontology classes is not limited to the asserted structure of the ontology.

Consider the OWL ontology fragment shown in Figure 6.2, if the class `Animal` is selected for class based browsing, then an ontology reasoner could infer that instances of `Dalmatian` should also be displayed on the `Animal` page.

$$\begin{aligned}
& \text{Dog} \sqsubseteq \text{Animal} \\
& \forall \text{eats.DogFood} \sqsubseteq \text{Dog} \\
& \text{Dalmatian} \sqsubseteq \forall \text{eats.PremiumDogFood} \\
& \text{PremiumDogFood} \sqsubseteq \text{DogFood}
\end{aligned}$$

Figure 6.2: Animal ontology with complex class hierarchy.

6.2.1 Class Based Browsing

The Linked Data CMS configuration (which we refer to as *the configuration*) is a set of set of ontology classes, along with some metadata to allow the Linked Data CMS to configure the CMS entities in \mathcal{S} described in the previous section. We call the combination of these classes and associated metadata *browsing classes*.

The Configuration

The central entity in our Linked Data CMS is the configuration.

Definition 6.6 (Linked Data CMS Configuration). *A Linked Data CMS configuration for an ontology \mathcal{O} with components $\langle \mathbf{C}, \mathbf{OP}, \mathbf{DP}, \mathbf{I}, \mathcal{A}, \mathcal{T} \rangle$ is a pair $\mathcal{C} = \langle B, \text{lab} \rangle$,² where*

- B is a set of browsing classes; and
- $\text{lab} : (\mathbf{C} \cup \mathbf{OP} \cup \mathbf{DP} \cup \mathbf{I}) \rightarrow \text{String}$ is a function that maps ontology entities to human-readable labels.

Browsing Classes

A browsing class contains an ontology class (called the *base class*) and some associated metadata which is used as a view of a set of individuals in the CMS. This includes the datatype and object properties that have been chosen for presentation for instances of the base class. The Linked Data CMS mapping uses the set of browsing classes to create the set of page templates in the CMS that correspond to the chosen base classes.

Definition 6.7 (Browsing Class). *A browsing class in a configuration $\mathcal{C} = \langle B, \text{lab} \rangle$ for the ontology $\mathcal{O} = \langle \mathbf{C}, \mathbf{OP}, \mathbf{DP}, \mathbf{I}, \mathcal{A}, \mathcal{T} \rangle$ is a tuple*

$$b = \langle b_C, B_{DP}, B_{OP}, b_\omega \rangle \in B, \quad \text{where}$$

- $b_C \in \mathbf{C}$ is the base OWL class;
- B_{DP} is a set of datatype properties $\subseteq \mathbf{DP}$, such that $\forall p : p \in B_{DP} : \mathcal{O} \models b_C \sqsubseteq \text{domain}(p)$;
- B_{OP} is a set of object properties $\subseteq \mathbf{OP}$, such that $\forall p : p \in B_{OP} : \mathcal{O} \models b_C \sqsubseteq \text{domain}(p)$;
- b_ω is a function that assigns to each object property $p \in B_{OP}$ a class $c \in \mathbf{C}$, such that $\mathcal{O} \models \text{range}(p) \sqsubseteq c$ and $\exists b' \in B : c = b'_C$.

²Section 2.2 gives a description of the sets \mathbf{C} , \mathbf{DP} , \mathbf{OP} of an ontology \mathcal{O} .

6.2.2 Linked Data CMS mapping

Given an ontology \mathcal{O} and a configuration $\mathcal{C} = \langle B, lab \rangle$ for \mathcal{O} , Linked Data CMS mapping algorithm (Algorithm 6.1) can be used to construct a CMS $\mathcal{S} = \langle T, F, R, P \rangle$. The definition of a Linked Data CMS mapping makes use of the notion of entailment (“ \models ”) from \mathcal{O} , ontology reasoning is required to both construct the CMS and to validate a potential configuration, i.e., test that all the conditions are satisfied.

The Linked Data CMS mapping can be summarised as follows. First a field is created for each datatype property in any B_{DP} . A page template t can indirectly contain other page templates, via T_R , and so page templates and relationships are created in two phases. The first phase creates the outer structure of each page template and stores these structures in $T_{hash}[]$, which associates an ontology class with the page template that will represent it. The second phase fills in the links between templates, via the relationships, updating the template structures in place. Where a relationship is required to hold a template, a pointer to the relevant template is retrieved from $T_{hash}[]$. Finally the page instances are created, in two phases which are similar, since a page instance p can indirectly contain another page instance, via P_R . Here $P_{hash}[]$ is used to keep a mapping between ontology individuals and the page instances that will be used for them.

The CMS configuration process implemented in this algorithm is *reasoning-driven* because the page instances depend on what is entailed by the ontology, and in general a reasoner is needed to establish this.

The Linked Data CMS mapping also allows page instances to be used as *update views*, i.e. entities in the CMS can be mapped to entities in the ontology by looking at the browsing classes.

Finally, we can define the Linked Data CMS.

Definition 6.8 (Linked Data CMS). *A Linked Data CMS is the combination of an ontology $\mathcal{O} = \langle \mathcal{C}, \text{OP}, \text{DP}, \text{I}, \mathcal{A}, \mathcal{T} \rangle$, a configuration $\mathcal{C} = \langle B, lab \rangle$ and the resulting CMS $\mathcal{S} = \langle T, F, R, P \rangle$ once the Linked Data CMS mapping has been applied.*

The definitions presented show how a configuration \mathcal{C} is mapped to and CMS \mathcal{S} based on the entailments of \mathcal{O} ; relying on an ontology reasoner for instance checking and class subsumption checking. Once the Linked Data CMS mapping has been applied, the combination of \mathcal{O} , \mathcal{C} and \mathcal{S} is a Linked Data CMS.

6.2.3 Class Based Browsing Example

We now revisit the CMS entity example presented in Section 6.1.1 and show how this CMS structure can be created from an example ontology \mathcal{O} using our Linked Data CMS approach.

Algorithm 6.1 Linked Data CMS Mapping Algorithm

```

1: function LDCMS(ontology  $\mathcal{O} = \langle \mathbf{C}, \mathbf{OP}, \mathbf{DP}, \mathbf{I}, \mathbf{A}, \mathbf{T} \rangle$ , configuration  $\mathcal{C} = \langle B, lab \rangle$ )
2:    $T_{hash}[] \leftarrow \emptyset$  ▷ A hash table from classes to page templates, initially empty
3:    $P_{hash}[] \leftarrow \emptyset$  ▷ A hash table from individuals to page instances, initially empty
4:    $T \leftarrow \emptyset$  ▷ Sets of page templates, page instances, relationships and fields
5:    $P \leftarrow \emptyset$ 
6:    $R \leftarrow \emptyset$ 
7:    $F \leftarrow \{ \langle lab(p) \rangle \mid \langle b_C, B_{DP}, B_{OP}, b_\omega \rangle \in B, p \in B_{DP} \}$ 
8:   for all  $\langle b_C, B_{DP}, B_{OP}, b_\omega \rangle \in B$  do ▷ Initialise page templates.
9:      $T_{hash}[b_C] \leftarrow \langle lab(b_C), \{ \langle lab(p) \rangle : p \in B_{DP} \}, \emptyset \rangle$ 
10:  end for
11:  for all  $\langle b_C, B_{DP}, B_{OP}, b_\omega \rangle \in B$  do ▷ Finalise page templates and relationships
12:    for all  $p \in B_{OP}$  do
13:      for all  $b' \in B$  do
14:        if  $b'_C = b_\omega(p)$  then
15:           $r \leftarrow \langle lab(p), T_{hash}[b'_C] \rangle$ 
16:           $R \leftarrow R \cup \{r\}$ 
17:           $T_{hash}[b_C]_R \leftarrow T_{hash}[b_C]_R \cup \{r\}$ 
18:        end if
19:      end for
20:     $T \leftarrow T \cup \{T_{hash}[b_C]\}$ 
21:  end for
22:  end for
23:  for all  $\langle b_C, B_{DP}, B_{OP}, b_\omega \rangle \in B$  do ▷ Initialise page instances
24:    for all  $a$  such that  $\mathcal{O} \models b_C(a)$  do
25:       $P_{hash}[a] \leftarrow \langle lab(a), T_{hash}[b_C], \{ \langle lab(p), v \rangle : p \in B_{DP}, \mathcal{O} \models p(a, v) \}, \emptyset \rangle$ 
26:    end for
27:  end for
28:  for all  $\langle b_C, B_{DP}, B_{OP}, b_\omega \rangle \in B$  do ▷ Connect pages instances
29:    for all  $a$  such that  $\mathcal{O} \models b_C(a)$  do
30:      for all  $p \in B_{OP}$  do
31:        for all  $v$  such that  $\mathcal{O} \models p(a, v)$  do
32:           $r \leftarrow \langle lab(p), T_{hash}[b_\omega(p)] \rangle$ 
33:           $P_{hash}[a]_R \leftarrow P_{hash}[a]_R \cup \{ \langle r, P_{hash}[v] \rangle \}$ 
34:        end for
35:      end for
36:     $P \leftarrow P \cup \{P_{hash}[a]\}$ 
37:  end for
38:  end for
39:  return  $\langle T, F, R, P \rangle$ 
40: end function

```

The ontology \mathcal{O} (illustrated in Figure 6.3) consists of:

```

Person(may); name(may, "May"); dob(may, "1972-01-05");
Person(pam); name(pam, "Pam"); parentOf(may, pam);
Person(chris); name(chris, "Chris"); parentOf(may, chris);
rdfs:label(may, "May"); rdfs:label(pam, "Pam"); rdfs:label(chris, "Chris").

```

We have omitted the `rdfs:label` axioms for classes and properties, however we assume they have been defined in the same way as for the individuals.

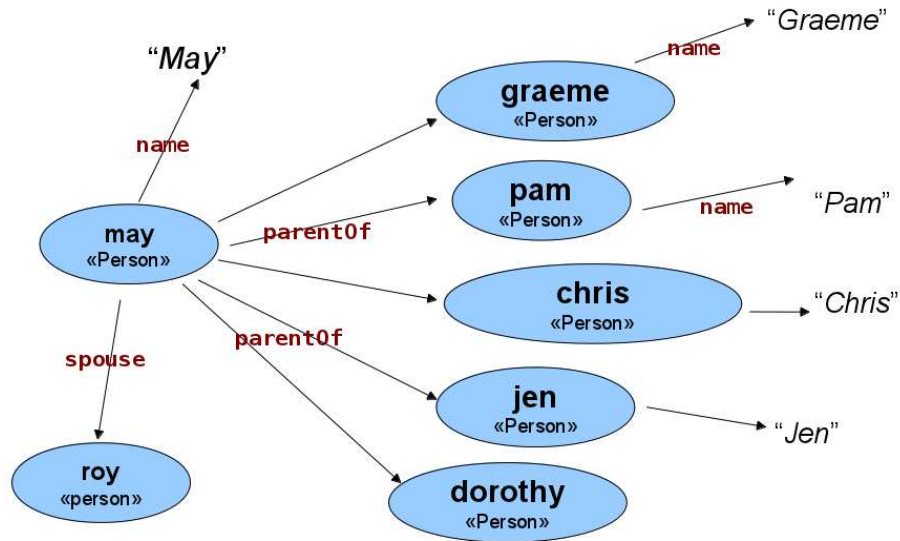


Figure 6.3: Example person ontology instances

The user can then define the configuration $\mathcal{C} = \langle B, lab \rangle$ by selecting base classes from the ontology and defining the set of browsing classes. The base classes are selected by deciding which of the classes in the ontology should be used to create page instances in the CMS, i.e., at what level in the class hierarchy of the ontology should the individuals be presented.

In this straightforward example we can define a browsing class for the `Person` class using the vocabulary from the ontology:

$$\begin{aligned}
 B &= \{ \langle \text{Person}, \{ \text{name}, \text{dob} \}, \{ \text{parentOf} \}, \{ \langle \text{parentOf}, \text{Person} \rangle \} \rangle \}; \\
 lab(x) &= l : \mathcal{O} \models \text{rdfs:label}(x, l).
 \end{aligned}$$

Although these labels are not the same as those used in Section 6.1.1, $lab(x)$ could easily be defined to produce labels based on the type of x , e.g., if x is a class, then $lab(x) = \text{concat}(l, \text{"s Display Page"}) : \mathcal{O} \models \text{rdfs:label}(x, l)$; where concat is the string concatenation function. However, for ease of presentation we use the simpler definition of $lab(x)$ in this example.

Once the configuration has been defined it can be validated against the class based browsing definitions in Section 6.2.1. The validation process is dependent on the particular implementation of the Linked Data CMS. However the validations that could be supported by an implementation fall into two main categories: (i) structural checks, e.g., have all of the properties assigned to the browsing classes been defined the ontology; and (ii) semantic checks, i.e., do the conditions in the class based browsing definitions hold. Many of these semantic checks could be performed by an ontology reasoner, however we leave this to future work.³

Now the Linked Data CMS mapping algorithm (Algorithm 6.1) is applied to \mathcal{O} and \mathcal{C} to produce a CMS \mathcal{S} . In the following steps we refer back to the steps in the mapping algorithm that are used to create each component.

First the fields are created (lines 4–10):

$$\begin{aligned} F &= \{name, dob\}; \\ name &= \langle \text{"name"} \rangle; \quad dob = \langle \text{"dob"} \rangle; \end{aligned}$$

A page template is then created for each browsing class in B and the relationships are created to connect these together (lines 11–22):

$$\begin{aligned} person &= \langle \text{"Person"}, \{name, dob\}, \{parentOf\} \rangle; \\ T &= \{person\}; \\ parentOf &= \langle \text{"parentOf"}, person \rangle; \\ R &= \{parentOf\} \end{aligned}$$

Then for each new page template, a page instance is created for each individual in the ontology that has a corresponding browsing class (lines 23–38):

$$\begin{aligned} P &= \{may, pam, chris\}; \\ may &= \langle \text{"may"}, person, may_F, may_R \rangle; \\ pam &= \langle \text{"pam"}, person, \{\langle name, \text{"Pam"} \rangle\}, \emptyset \rangle; \\ chris &= \langle \text{"chris"}, person, \{\langle name, \text{"Chris"} \rangle\}, \emptyset \rangle; \\ may_F &= \{\langle name, \text{"May"} \rangle, \langle dob, \text{"1972-01-05"} \rangle\}; \\ may_R &= \{\langle parentOf, pam \rangle, \langle parentOf, chris \rangle\}. \end{aligned}$$

Finally, the CMS \mathcal{S} is constructed (line 39):

$$\mathcal{S} = \langle T, F, R, P \rangle;$$

³Some details of our future work to validate configurations using meta ontology reasoning in OWL 2 are presented in Section 7.2.3.

In this example we have created the CMS entities in \mathcal{S} to reproduce the CMS structure described in Section 6.1.1, based on the ontology \mathcal{O} and configuration \mathcal{C} . This example illustrates that given an ontology and a fairly simple configuration, a significant CMS can be created in a straightforward manner. The ontology in this example was small (since we only used three individuals) however using our approach we can increase the number of page instances at no extra cost to the user, since the complexity of defining a configuration is only related to the number of classes chosen for display.

6.3 Drupal Linked Data CMS Implementation

We have implemented our Linked Data CMS approach using the Drupal platform. Drupal 7 was chosen because it has RDF support built into the core system, and due to the Drupal community offering a wide range of extension modules. Our Linked Data CMS approach is implemented as a Drupal module which builds on a number of existing semantic web based modules. In this section we provide an overview on how our approach has been implemented within Drupal. We also provide an overview of the system in Appendix F.

In our Drupal implementation we allow the user to specify the configuration in terms of browsing classes, fields (for datatype properties), and relationships (for object properties), following our class based browsing approach described in Section 6.2.1. In our Drupal module the configuration is specified by creating a set of data structures in PHP to represent Linked Data CMS configuration.⁴

An example field for the datatype property `ex:nick`, and relationship for the object property `ex:knows` are defined below:

```
$fields[] = array(
    'property' => 'ex:nick',
    'label'    => 'Nick name',
    'type'     => 'text',
);
$fields[] = array(
    'property' => 'ex:knows',
    'label'    => 'Knows',
    'type'     => 'relationship',
);
```

Both fields and relationships are specified in the `$fields[]` array, and are differentiated by the `type` field. The `property` field specifies the URI of the datatype or object property, and the `label` field specifies a human readable label describing the field or relationship.

Similarly browsing classes are specified using the `$browsing_classes[]` array:

```
$browsing_classes[] = array(
    'name' => 'person',
    'label' => 'People',
```

⁴PHP is the scripting language used to develop the server side components of Drupal.

```
'class' => 'ex:Person',
);
```

The `name` field defines an identifier for the browsing class used throughout the configuration, the `label` field is a human readable label describing the browsing class, and the `class` field specifies the base OWL class for the browsing class.

The fields (datatype properties) defined in the `$fields` array can be assigned to one or more browsing classes using the `$field_assignments` array. For example the `ex:nick` and `ex:dob` fields can be assigned to the 'person' browsing class as follows:

```
$field_assignments['person'] = array(
  'ex:nick',
  'ex:dob',
);
```

Similarly the relationships (object properties) defined in the `$fields` array can be assigned to one or more browsing classes by creating new items in the `$relationship_assignments` array for each browsing class:

```
$relationship_assignments['person'][] = array(
  'property' => 'ex:knows',
  'target'   => 'ex:Person',
);
```

The relationship assignment requires a property URI corresponding to a relationship field and a target class URI, which corresponds to a browsing class.

A complete example of a sample configuration for the Drupal Linked Data CMS module is also provided in Appendix G.

The configuration data structures are then used as the input to the Linked Data CMS mapping algorithm (Algorithm 6.1), which is also implemented as part of the module. The output from the mapping algorithm is then used to automatically create a set of Drupal entities, using the Views, Panels and SPARQL Views modules to represent the fields, relationships and page templates. At runtime the page instances are generated on the fly by the SPARQL Views module. The SPARQL Views module uses the Drupal entities created by our module to generate queries to the ontology via a SPARQL endpoint.

We configure Drupal with three main displays for each page template: (i) the *listings view*, which allows users to browse and search ontology individuals; (ii) the *details view*, which allows users to view all the fields and relationships for a particular individual; (iii) the *update view*, which allows users to create, update and delete individuals in the ontology.

6.3.1 Implementation using SPARQL Views

SPARQL Views (Clark, 2010) is an existing Drupal module, which builds on the SPARQL module and the well known Views modules. The SPARQL Views module allows users to map Drupal fields to RDF predicates and then build a view based on those fields. The field mappings are grouped into SPARQL Views Resource Types, each intended to represent a particular type

of RDF resource. SPARQL Views automatically generates a SPARQL query based on the RDF mapping and Views specification. The view can be configured by users selecting the appropriate *fields*, *filters* (usually based on URL parameters), *relationships* to other SPARQL Views resources and display configuration.

Figure 6.4: Drupal SPARQL Views specification, defining the configuration of the Drupal View using the fields and relationships for a particular browsing class.

However, as the number of RDF resources and predicates increases, the complexity of the task of defining and maintaining both the SPARQL Views Resources and Views becomes increasingly difficult. For example, for an RDF dataset with 10 types of RDF resources (or OWL/RDFS classes), each with an average of 10 properties, the user has to enter 100 mappings into SPARQL Views (even if these resource types share properties). For each of these 10 resource types, there is likely to be a corresponding view, with its own set of fields, filters, relationship and display configuration. The reason for this limitation is that SPARQL Views is a fairly general tool for building SPARQL queries and rendering the results within the CMS. In our approach we can exploit the fact that we limit ourselves to configuring the CMS in terms of class based browsing.

In essence our Drupal Linked Data CMS automatically configures a set of SPARQL Views Resources and Views based on an OWL ontology and user specification of how that ontology should be represented in Drupal, along with an additional set of pages allowing those resources to be maintained (create / update / delete). The browsing classes identified by the user, along with their associated datatype and object properties are mapped to resource types, fields and relationships respectively. A default SPARQL Views view is then created for each browsing

class (listings and details views), this view can then be modified by the user using the standard Views interface (Figure 6.4).

To handle the layout of pages in Drupal, we use the Panels module and automatically generate a Drupal page template to control the display of multiple views on a single page (Figure 6.5).



Figure 6.5: Drupal Page Template, defining the main content areas used to display the details of an ontology individual (the page instance).

6.3.2 Configuration and Maintenance

Our Drupal module uses the configuration to create an entire Drupal site using SPARQL Views, based on the ontology and configuration specified by the user. This approach also centralises the maintenance of the structure of the CMS with respect to the ontology, e.g., if a new browsing class is required, the user can update the configuration and then run the Linked Data CMS mapping to create the required Drupal entities. Additionally our approach can handle changes to the schema of the ontology. For example if a change in the ontology occurs, such as a domain/range, additional classes or a change of URIs, then the configuration can be used to synchronise Drupal with the ontology schema.

Drupal site administrators can also maintain the Drupal Linked Data CMS generated by the configuration in the same way as a regular Drupal site. More specifically once the configuration mapping has been applied, site administrators are free to change the labels, fields, RDF mappings, page layouts and so on using the administration interface provided by the various Drupal modules.

The configuration entities are modelled in our Drupal module by the classes shown in Figure 6.6. An `LDCMSConfiguration` object is created by the implementation of the Linked Data CMS mapping algorithm (Algorithm 6.1) in the Drupal Linked Data CMS module.

Briefly, the `LDCMSConfiguration` class consists of collections of browsing classes (resources), view specifications (views), fields and relationships (fields). The `LDCMSConfiguration` also contains tables to map browsing classes to views specifications (`baseClassToView`), and base classes to browsing classes (`baseClassToResource`).

An `RDFResource` corresponds to a browsing class in the configuration. Each `RDFResource` has a base class (`baseClass`), a set of fields (`fields`), and a set of relationships (`relationships`). An `RDFViewSpec` provides some additional information to the Drupal Views module to determine how a browsing class (`RDFResource`) should be displayed. An `RDFField` is used to represent a field or relationship, and are distinguished by their `fieldType` attribute.

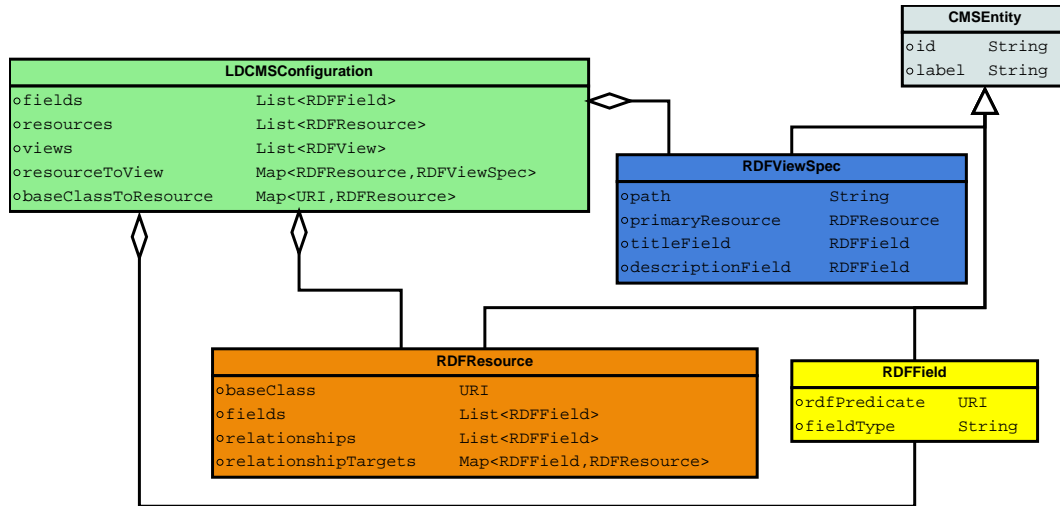


Figure 6.6: Drupal Linked Data CMS Configuration Entities

These entities are used by the Linked Data CMS module to provide metadata to the Drupal API and other Drupal modules at runtime for the Linked Data CMS.

6.4 Case Study: Cultural Heritage Linked Data CMS

We have used our Linked Data CMS approach to build a system that manages a repository of cultural heritage linked data based on our Drupal Linked Data CMS implementation (Taylor et al., 2013). The Hebridean Connections cultural repository is a repository about people, places historical events and artefacts in the Outer Hebrides. In the repository, the people's relationships to each other, their occupations, the places they have lived, events they have been involved in and even the historical artefacts they have interacted with have all been recorded (Beel et al., 2013).

6.4.1 Cultural Heritage Ontology

As an OWL ontology, the repository consists of approximately 32,000 individuals, 520 classes, 250 object properties, 55 datatype properties. The ontology schema uses an OWL 2 RL level of OWL expressivity, where most of the expressive power is used to express relationships between object properties.

This ontology makes heavy use of domain, range, functional, reflexive, symmetric and transitive property axioms, while having a relatively simple set of atomic classes. Using our reasoning driven configuration, the Linked Data CMS has been configured with 16 browsing

classes, with 106 datatype and object properties from the ontology. The browsing classes contain a total of 211 field assignments (datatype properties) and 118 relationship assignments (object properties).⁵

The browsing classes used for the ontology have been selected at a fairly high level close to the top of the classification hierarchy. Some examples of the browsing classes are: Person, Occupation, Residence, Business; each class subsumes a number of more specific classes which we have also used to configure search filters in the CMS. The browsing classes and their sub-classes appear in the ranges of the object properties used as relationships in the Linked Data CMS.

6.4.2 Drupal Linked Data CMS Configuration

The configuration is used to instantiate both the SPARQL Views resource types (and associated RDF mappings) and Views in Drupal.

Once the Drupal Linked Data CMS has been configured, 16 resource types are created, with a total of 329 fields between them (from the 211 field + 118 relationship assignments). The corresponding views are then configured based on these fields. Three types of views are created for each browsing class, a *listings view* (Figure 6.7), a *details view* (Figure 6.8) and an *update view*.

The screenshot shows the 'Hebridean Connections Demo' interface. At the top, there is a search bar with the text 'Search Hebridean Connections' and a dropdown menu for 'Subject Type' set to 'People'. Below the search bar is a 'Search' button. On the left, there is a sidebar with a 'CURIOUS' menu listing various categories like Boats, Buildings and Public Amenities, Businesses, etc. The main content area is titled 'People' and contains a search bar with the text 'Donald MacDonald', an 'Items per page' dropdown set to '20', and a 'Search' button. Below the search bar is a table with three columns: 'Title', 'Description', and 'Files'. The table lists several entries for 'Donald MacDonald' with their respective descriptions and file status.

Title	Description	Files
Donald MacDonald	Donald, son of Calum Macdonald and Margaret Macphail, 52a Balalain married Sheena Cadzow, Girvan. The family moved to Stornoway in 2001.	
Donald MacDonald	Donald born c1827 was a son of Malcolm and Christina nee MacLennan who settled at 6 Carishader after moving from Reel. He was married to Catherine nee Smith from 2 Ungeshader and...	
Donald MacDonald	Donald Macdonald, 38 North Tolsta married Marion Morrison, 16 Balalain and settled in Edinburgh. After school in Tolsta and the Nicolson Institute, Donald studied (and played...	
Donald MacDonald	Donald Macdonald, son of Murdo (Murchadh Buidhe) of 51 North Tolsta, married Christina Macdonald of 38 Tolsta and settled there. They were the parents of Catherine and Donald,...	
Donald MacDonald	Donald Macdonald (1921-1992), son of Alexander Macdonald and Mary A Macleod, 10 Keose Glebe, also known as Doi Alasdair Dhuinn, married Donaldina Mackenzie, 4 Keose. He served as...	true
Donald MacDonald	Donald Macdonald (1885), son of Alexander Macdonald and Ann Mackenzie, 13 Keose, who died aged 14 days.	

Figure 6.7: Listings View in Drupal Linked Data CMS, generated for each browsing class to list all page instances for that browsing class from the ontology.

The listings view provides an interface to search and browse the repository for a particular

⁵An *assignment* is an occurrence of a field or relationship being used in a browsing class.

resource type. The listing displays the *base fields* for multiple resources, allowing the user to browse through them. The base fields are specified as `dc:title` and `dc:description` in the Drupal Linked Data CMS. These base fields are used when a summary of an ontology entity is required.

The screenshot shows a web interface for a person's details. On the left is a sidebar menu titled 'CURIOS' with a list of categories like Boats, Buildings and Public Amenities, Businesses, Crofts and Residences, Gaelic Verse B R dachds, Historical Events, Landmarks and Archaeological Sites, Locations, Natural Landscape Features, Objects and Artefacts, Organisations, People, Resources, Story Reports and Traditions, Vehicles, and ALL SUBJECTS. The main content area is titled 'Dolina Macdonald' and contains a biographical paragraph: 'Dolina Macdonald, Doileag a Ghreusaich (1898-1990), was the daughter of Donald Macdonald and Peggy Macritchie from Geshader. Dolina was married to Finlay Macleod from 2 Enaclete, where they both lived and raised a family.' Below this is a 'Back to listing' link and a table of personal details: Subjectid (2742), English Name (Dolina Macdonald), Bk Reference (68), Born (1898-12-17), Date Of Death (1991-03-03), and Sex (Female). To the right of the table are two image galleries. The first gallery shows Dolina Macdonald and a young girl. The second gallery shows Dolina Macdonald with a group of people. On the far right, there are links for 'Associated With Oral Tradition', 'Child Of Donald Macdonald... Margaret Macrit...', 'Lived At 2 Enaclete 1 Geshader', 'Married Finlay Macleod', and 'Parent Of Dolly Alice Mac... Margaret Cather...'.

Figure 6.8: Details View in Drupal Linked Data CMS for a Person page, generated for the ‘person’ browsing class.

The details view provides an interface to view a single resource in detail. This view consists of four distinct regions (Figure 6.9), each displaying a particular view block. The summary region displays the base fields; the properties region displays a field listing (datatype properties); the image gallery is specifically for displaying a specific image field as a gallery on the web page; the relationships region displays a summary view for each relationship (object property). The relationships view displays the title of the related resource, and a linked to its corresponding page instance in the Linked Data CMS.

The update view allows users to create, update and delete individuals in the Hebridean Connections Linked Data repository. This functionally is not included in any existing Drupal module, so the Drupal Linked Data CMS provides a special set of pages and SPARQL Update (SPARQL, 2012) query generator for this purpose. The pages are built using data from the configuration and the existing SPARQL Views resource types.

The Linked Data CMS configuration allowed a fairly complex Drupal site for Hebridean Connections to be quickly developed by specifying a minimal configuration. Much of the configuration was generated using SPARQL queries against the Hebridean Connections ontology, since many of the browsing classes use all of the properties whose domain contain the browsing

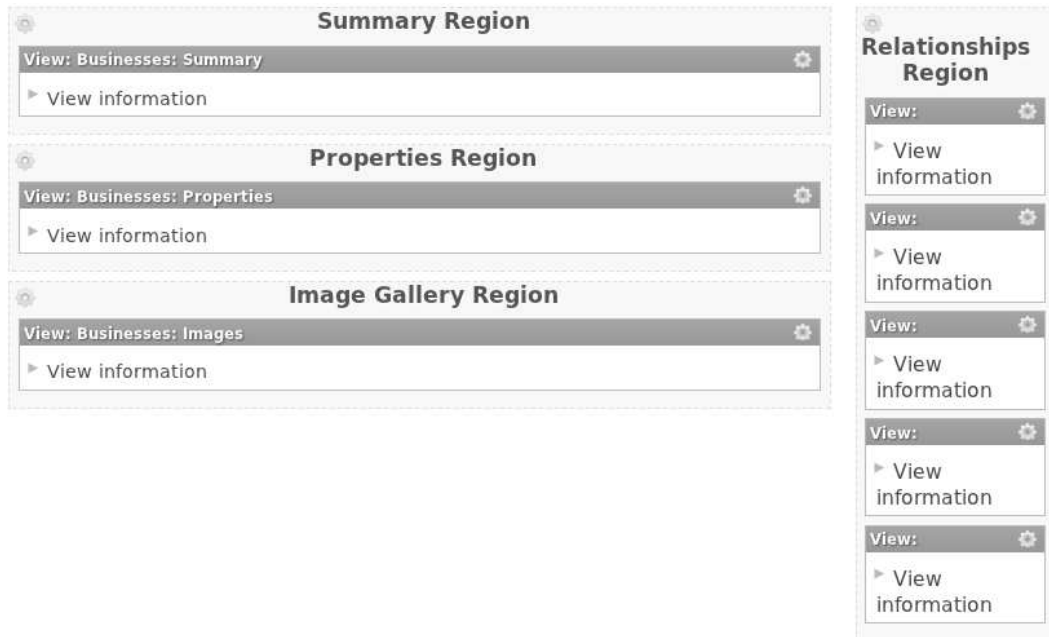


Figure 6.9: A Page Template Instance in Drupal Linked Data CMS, which controls the layout for the fields and relationships defined for a particular browsing class.

class. Once the Drupal site has been created, the configuration can also be used to maintain the site with respect to the ontology. For example, if changes occur in the ontology, then each effected resource type and view can be automatically updated by adding the changes to the configuration.

6.4.3 Querying and Reasoning Requirements

In this case study we used the Fuseki SPARQL server (Seaborne, 2011), since it can be integrated with Jena API reasoners (Jena, 2013) and supports SPARQL 1.1 (SPARQL, 2012). The Drupal Linked Data CMS uses SPARQL 1.1 for all communication with the endpoint (query and update services). This allows the selection of any SPARQL 1.1 compatible endpoint for the system.

The Hebridean Connections ontology schema is quite straightforward in terms of expressivity and fits within the OWL 2 RL profile. This feature means that we could alternatively make use of a rule extended DBMS for storage and querying, such as Oracle 11g (Steiner, 2010) or OWLIM (Bishop et al., 2011).

However for the purposes of this case study we have made use of the TrOWL ontology reasoning infrastructure (Section 2.3), using semantic approximation to provide high performance run time query answering. This allows us to also make use of our reusable infrastructure for semantic mashup (Section 5.1), so that our Linked Data CMS approach could be extended in future to include resources from folksonomy-based systems by making use of folksonomy search expansion (Chapter 4). The TrOWL reasoner is used both for the configuration and at query time for inference. By using the lightweight reasoning provided by TrOWL the Linked

Data CMS could scale to millions of pages (instances of browsing classes in the domain ontology) (Thomas et al., 2010a).

6.5 Summary

In this chapter, we have presented a general approach to automatically set up a traditional web content management system to manage semantic web data based on an OWL ontology and a user specification of the views of the ontology to be presented (the *configuration*) (Objective 3). The entities generated by the mapping algorithm (Section 6.2.2) rely on ontology entailments, which are used to automatically infer additional relationships between pages in the CMS. The validation of the user specification and the generation of the CMS are both *reasoning-driven*, in that they make essential use of ontology reasoning. In our approach we encourage the use of lightweight reasoning, which allows the configuration process and runtime querying in our implementation to scale to large domain ontologies.

The approach relies on using a particular method of managing ontology data in the content management system, called class based browsing. Using our approach the gap between linked data and popular the CMS tools currently in use on the web is greatly reduced, allowing ordinary web users to contribute directly to the semantic web using familiar CMS tools.

We also showed how our Linked Data CMS approach can be implemented in the Drupal CMS by building on top of some popular Drupal modules. An additional advantage of this implementation is that once the class based browsing has been configured in Drupal, users can still use all of the standard Drupal tools to customise the Linked Data CMS.

In our proof of concept case study (Section 6.4), we presented an instantiation of the Linked Data CMS with cultural heritage data. This structure of the Hebridean Connections ontology is used to configure the CMS with several browsing classes, fields and relationships. The relatively large amount of individuals in the ontology and relative complexity of the schema means that careful consideration must be taken over the runtime reasoning approach. We made use of lightweight reasoning, with the high performance OWL 2 QL query engine and semantic approximation provided by TrOWL, with the Fuseki SPARQL 1.1 endpoint.

Chapter 7

Discussion, Conclusions and Future Work

In this chapter we conclude by reviewing the objectives presented in Chapter 1 and discussing the extent to which they have been satisfied by the work in this thesis. We also discuss some future work motivated by this thesis.

7.1 Thesis Overview and Discussion

In this section we review each of the main contributions with respect to the thesis objectives and their wider impact on the semantic web.

We addressed the question posed by Problem I: Bridging the Web of Data (Section 1.1.1): “*Can semantic web applications be configured to reuse user-generated content from folksonomy-based tagging systems?*” with the specific research questions identified in Objectives 1–2 with our Folksonomy Search Expansion approach in Chapter 4 and our Reusable Infrastructure for Semantic Mashup in Chapter 5.

We also addressed the question posed by Problem II: Building with Semantic Tools (Section 1.1.2): “*Can existing web content management systems be exploited by to encourage user-generated content on the semantic web?*” in Objective 3 with our Linked Data Content Management Systems approach proposed in Chapter 6.

7.1.1 Folksonomy Search Expansion

In Chapter 4 we presented our Folksonomy Search Expansion approach which makes use of ontology reasoning to allow the reuse of tagged resources in semantic mashup applications, without bothering the users of the tagging systems with the complexity of an ontology-based approach. This approach can be configured to give high precision results using a selection of expansion methods based on the structured information in OWL ontologies. We investigated the application of various lightweight reasoning approaches that could be used to provide a basis for practical semantic web applications using our approach.

Objective 1 was to answer the research question: *Can semantic web applications be configured to reuse user-generated content from folksonomy-based tagging systems, without sacrificing the benefits of folksonomies by limiting the ease of user contribution?* The approach to achieve this objective should allow users to continue to use existing folksonomy tools to annotate web resources. Motivated by the existing work in Chapter 3, our approach to achieve this objective

was to make use of a query expansion approach, driven by domain ontologies to retrieve tagged resources based on ontology entities.

This objective is addressed in our approach by allowing web application developers to exploit the benefits of ontologies without bothering the users of the tagging systems with their rigidity. The objective was achieved by: (i) automatically associating keywords with classes, properties and individuals in domain ontologies, so that the approach can be applied to any domain ontology without requiring ontology entities to be manually indexed with keywords (Section 4.2.1); and (ii) by proposing a query expansion approach to expand the search keyword(s) generated by ontology-based semantic mashup applications to keyword sets based on the structure of domain ontologies (Section 4.2.2). These two features of our approach allow folksonomy users to still enjoy the benefits associated with the folksonomy-based approach, while proving semantic web developers with an approach to combine tagged resources with semantic web applications. We then performed experiments with our approach in a proof of concept evaluation, showing its practical application with a case study in the music domain (Section 4.6). These experiments showed that the proposed approach is feasible for a significant domain of linked data and that the existing annotation process used in folksonomy-based systems are adequate reuse in semantic web applications using our approach.

Objective 1.1 was to answer the research question: *Can the approach proposed in Objective 1 be configured to have high precision?* The approach to achieve this objective should make use of the structured data in ontologies to help address the limitations of folksonomy searching in our approach. With regard to the existing work presented in Chapter 3, we aimed to propose an approach that allows semantic web applications to reuse tagged resources, annotated with only straight-forward keyword-based metadata, with a high level of precision. To provide support for a wide variety of domains we allow query expansion based on user-defined ontologies that can make use of data held in both the TBox and ABox of the ontology.

This objective is addressed by proposing a number of expansion methods based on the core relationships contained in OWL ontologies (Section 4.2.2), which are closely related to query expansion techniques used in traditional web search engines. These expansion methods can be tailored by developers to suit the target domain and tagging systems (Section 4.2.3). We then illustrated the usefulness of our approach in terms of precision with a proof of concept evaluation, with our MusicMash2 case study semantic mashup application in the music domain (Section 4.6). To perform the experiments we made identical *top k* searches for music artists with YouTube and with our folksonomy search expansion implementation Taggr. The experiments showed that the proposed approach can be applied with high precision when a good quality domain ontology was available.

Objective 1.2 was to answer the research question: *Can the proposed approach in Objective 1 make use of ontology reasoning to support a wide range of ontologies, regardless of the expressivity of their schema?* The approach to achieve this objective should make use of light-weight reasoning to provide support for practical semantic web applications.

The objective has been addressed by making use of ontology entailments in our approach

(Section 4.3), and presenting the various levels of lightweight reasoning which can be applied with folksonomy search expansion (Section 4.4). We also have shown that these expansion methods can be efficient even for some very large ontologies in our proof of concept evaluation using our case study in the music domain and an approximation-based lightweight reasoning approach (Section 4.6.4).

We have shown how lightweight reasoning can be applied to reuse resources from folksonomy-based tagging systems in semantic web applications, by making use of our ontology-based folksonomy search expansion approach. We have showed that the combination of ontology, folksonomies and reasoning can form a basis for practical semantic web application. How to apply semantic web technologies to improve folksonomy-based systems and social networks has been a pressing issue for the semantic web community. Our approach also provides a platform reuse of user-generated content from folksonomy-based tagging systems. Future development of our approach could aim toward associating tagged resources with ontology entities based on the output of the folksonomy search expansion methods. This development could significantly lower the bar for the average web user to contribute to the semantic web, by utilising many of the user participation techniques perfected from their use in Web 2.0 applications.

It is worth noting that to be able to effectively apply Folksonomy Search Expansion by picking a high precision expansion method, the application developer is required to have a certain level of understanding of the tagging conventions used for a particular type of resource, or in a particular community (Section 4.2.3). In our evaluation we chose an *artist name*, *song title* combination for search expansion based on our observations of tagging trends in the YouTube music community. Additionally, the application developer may benefit from some deeper knowledge of the metadata ontologies used to perform the expansion in order to fully optimise their chosen expansion method. Our approach is also dependent on good quality ontologies being available for the application domain (Section 4.2.3). In our experiments we optimised our expansion method by viewing small subsets of the data, and then selecting the appropriate object property to achieve our intended search expansion.

In summary, our approach to reuse tagged resources from folksonomy-based systems in semantic web applications can be seen as a contribution to addressing the problem of user-generated content on the semantic web, as well as improving the performance of search in folksonomy-based systems.

7.1.2 Reusable Infrastructure for Semantic Mashup

In Chapter 5 we presented our Reusable Infrastructure for Semantic Mashup which provides an approach to building applications that can be configured to make use of folksonomy search expansion to reuse tagged resources in semantic web applications.

Objective 2 was to answer the research question: *Can a reusable infrastructure be configured to building applications that make use of the approach proposed in Objective 1?* The approach to achieve this objective should make use of lightweight reasoning to allow the development of

efficient and scalable applications and the infrastructure should be configurable to support the large variety of domains covered by Linked Data.

This objective is addressed by our reusable infrastructure for semantic mashup (Section 5.1). We provide a number of components in this infrastructure which can be used to develop applications that make use of folksonomy search expansion. We make use of an ontology search engine, ONTOSEARCH2 (Section 2.3), to automate the ontology-keyword association process that forms the basis of our folksonomy search expansion (Section 5.2); we make use of TrOWL to provide a variety of lightweight ontology reasoners and an ontology repository for developers to build practical efficient and scalable semantic web application(Section 5.2); we provide an implementation of folksonomy search expansion, Taggr, which is implemented on top of these components (Section 5.3); we demonstrate the usefulness of this infrastructure with a case study in the music domain (Section 5.4); and finally we provide a proof of concept evaluation our infrastructure with our case study mashup application MusicMash2 (Section 5.5) to show how an application in the music domain can be configured to use our infrastructure. In this case study we showed that the components of our infrastructure can be used to successfully support real world semantic mashup applications that reuse resources from folksonomy-based tagging systems in semantic web applications.

Our infrastructure provides folksonomy search transparently for users of mashup applications, and also allows semantic mashup developers can exploit the benefits of ontology enhanced folksonomy search without knowledge of the internal complexities of the ontology infrastructure or the related ontologies. Because we provide components offering lightweight reasoning, developers are free to use expressive domain ontologies in the proposed infrastructure. This is due to the approximation method used in TrOWL, which allows users to enjoy the benefits of an expressive ontology language, such as OWL 2 DL, while still having the query performance of a lightweight language such as OWL 2 QL. Also by using ONTOSEARCH2 to automatically extract implicit ontology metadata, we provide a platform for many other similar applications that make use of this type of metadata to be built using our infrastructure. The f-SPARQL fuzzy query engine used by Taggr to implement the search expansion provides Taggr with an elegant method of filtering metadata in our expansion methods.

To make our approach more widely applicable we would need to open our infrastructure up to provide advanced developers with the functionality to specify additional sources of ontology metadata other than those used by default in ONTOSEARCH2. This would allow the keyword association to be further optimised for user submitted ontologies. Additionally there may be many more useful expansion methods than those presented in Chapters 4 and therefore it would benefit developers to provide a method in our infrastructure to define new expansion methods, in addition to using those already provided by Taggr.

We have provided the tools for semantic web application developers to reuse tagged resources from folksonomy-based system by proposing a lightweight-reasoning based infrastructure that allows developers to configure a search expansion and target ontology method for domain specific semantic mashup applications. In addition our infrastructure also provides

developers with a high performance and scalable reasoning infrastructure for more general ontology reasoning and querying tasks.

7.1.3 Linked Data Content Management Systems

In Chapter 6 we presented our Linked Data CMS approach which allows existing CMS software to be configured to create a web site based on a group of ontology classes, using a technique which we call class based browsing. This approach makes use of a configuration which uses an ontology reasoner and SPARQL query engine to map ontological entities to CMS entities, and this configuration is then used to generate the CMS structure required to display a set of web pages based the ontology.

Objective 3 was to answer the research question: *Can traditional web content management systems be configured to maintain repositories of Linked Data?* The approach to achieve this objective should make use of domain ontologies to help automate the configuration of these tools, and must make use ontology reasoning so that any domain ontology could be used with the approach, regardless of the expressivity of its schema.

This objective is addressed with our Linked Data Content Management approach (Section 6.2). In this approach we present a method to configure existing CMS systems to manage Linked Data. We first show how the relevant features of content management systems can be modelled (Section 6.1); we then define a specific method of viewing ontology entities so that they can be browsed by human users, which we call class based browsing (Section 6.2.1); we then show how the CMS can be configured to manage linked data based on class based browsing and by making use of ontology reasoning, which also allows us to exploit both the explicit and implicit information contained in domain ontologies (Section 6.2.2); our approach is then implemented as an extension of the Drupal CMS (Section 6.3); finally we put our approach to the test with a case study in the cultural heritage domain (Section 6.4). This case study used a significantly large and complex dataset that is derived from a real world scenario of managing cultural heritage data. Our case study made use of lightweight reasoning to allow the system to be both efficient and scalable. It demonstrates that our reasoning based configuration approach can be applied to reuse existing CMS tools to manage large repositories of linked data.

By restricting our CMS–Linked Data integration to class based browsing we have provided users with a straightforward method for configuring existing content management systems to manage semantic web data. In this approach we have reused tools with a long and successful history in managing traditional web content, and applied them to the semantic web. This provides a much stronger integration between content management tools and semantic web data than existing approaches, which often offer only read *or* write capabilities. Once a Linked Data CMS has been configured, ordinary CMS users, such as those currently authoring blogs, wikis, etc., can contribute to Linked Data. Furthermore our implementation using the Drupal CMS targets a community of CMS users who already have a firm interest in semantic web and CMS integration and our formal description of CMS entities could be reused for further applications.

Although ontology schemas on the web tend to remain static, there may be cases when

the development of content in the CMS drives the development of the ontology, such as adding new properties or classes. At present our approach does not yet support the management of ontology TBox data. However, it may be possible to synchronise any changes from the class based browsing structure in the CMS (such as a user adding a new relationship to a browsing class), back to the ontology.

Our Linked Data CMS approach extends the current highly successful web content management paradigm for use in the semantic web, allowing ordinary CMS users to manage Linked Data content on the web.

7.2 Future Work

The work in this thesis has introduced some interesting new research questions regarding the next generation of work in this area. In this section we present three key areas of future work that have been inspired by this thesis.

7.2.1 Framework for Folksonomy Search Expansion

Our work on Folksonomy Search Expansion in Chapter 4 presents two new research questions that may help encourage further widespread use of our approach on the web: (i) Could expansion methods be parametrised in such a way that they can be defined by users using an existing semantic web language? (ii) Could machine learning techniques be applied to automatically select the more appropriate expansion method for a particular domain ontology?

Declarative Expansion Methods

Extending Taggr with further expansion methods is quite straightforward in our Java implementation. New expansion methods are implemented with a single Java class by extending an *AbstractExpansionMethod* class and implementing an interface. However, generally the difference between expansion method implementation classes is the input parameters and the generated SPARQL query pattern (Section 5.3.2).

A simplified SPARQL query for evaluating the IPVE expansion is shown in Figure 5.6. The SPARQL *basic graph pattern* (BGP) of the query (ignoring fuzzy thresholds) could be parameterised as follows (Figure 7.1):

```
?y os:hasKeyword #KEYWORD# .
?y #PROPERTY# ?x .
?x os:hasKeyword ?k .
```

Figure 7.1: SPARQL BGP template for IPVE expansion

Declaring a SPARQL BGP as a template with the appropriate parameterisation would allow users to define new expansion methods in a straightforward way. In this example the BGP with the parameters #KEYWORD# and #PROPERTY# represent the following expansion call for IPVE:

$$E(\#KEYWORD\#, IPVE(\#PROPERTY\#))$$

These user-defined expansion methods are likely to relate closely to the core relationships found in OWL ontologies (Definition 2.1) which are not already captured in the six proposed expansion methods in Chapter 4.

This property of the expansion methods raises the question of whether it is possible for new expansion methods to be defined declaratively rather than imperatively, using existing declarative semantic web languages such as RDF, OWL or SPARQL.

Automated Selection of Expansion Method

It may also be possible to apply machine learning techniques (Bishop and Nasrabadi, 2007) to automatically select the expansion method used to evaluate the folksonomy search expansion. This approach would rely on some pattern analysis in the target domain ontology to attempt to find the most effective expansion method. The selection of expansion methods could be refined over time by crowdsourcing (Quinn and Bederson, 2011) *relevant* or *not relevant* results (measuring precision) with crowdvoting.¹

7.2.2 Lightweight Rule Extension for RIF

Our reusable infrastructure for semantic mashup presented in Section 5.1 allows semantic mashup applications to be configured to make use of folksonomy search expansion and lightweight reasoning supported by current W3C standards (RDF, OWL, SPARQL). With the recent W3C standardisation of the Rule Interchange Format (RIF) (Kifer and Boley, 2010), the semantic web now has a basis for rule interchange, which raises the question of how our reusable infrastructure can be extended to also support lightweight RIF rule inference.

The combination of ontologies and rules is considered to be an important step forward for the semantic web, since many applications may require expressivity beyond that which OWL provides; e.g., to express constraints or to reason about closed-world knowledge. Logic programming rules can provide such expressivity and consequently, their combination with OWL DL is an active research area.

In particular, users may wish to draw inferences specific to their application which are either difficult to express or outwith the expressivity of OWL. For example, “*all rock bands dislike all boy bands*” is not straightforward to express in OWL but can be easily expressed by the following rule:

$$dislikes(x, y) \leftarrow RockBand(x), BoyBand(y).$$

Additionally, there limitations in the expressivity OWL and Description Logics in general. For example, “*if a band performed a concert with an artist whom they also dislike, then that band is upset*”, can be expressed by the following rule:

$$Upset(x) \leftarrow performedWith(x, y), dislikes(x, y).$$

¹Crowdvoting is the process of gathering (crowdsourcing) a group’s opinion on a particular topic.

The inferences derived from these sorts of rules could be particularly useful in semantic mashup and could be achieved by extending the reasoning component of our reusable infrastructure presented Chapter 5 with a lightweight rule extension.

A lightweight rule extension of our infrastructure could be achieved in one of two ways: (i) by extending the semantic approximation (Pan and Thomas, 2007) approach to support Description Logic Rules (DL-Rules) (Krötzsch et al., 2008) in RIF; or (ii) by providing a new semantic approximation method that targets OWL 2 RL (rather than OWL 2 QL) to make use of rule-extended databases to perform rule inference – due to the close relationship between OWL 2 RL and Horn-rules (Grosz et al., 2003a; Motik et al., 2009a).

7.2.3 Configuration Meta Ontology

Our Linked Data CMS approach presented in Section 6.2 makes use of a configuration to map ontology entities to entities found in typical traditional content management systems. This configuration can be thought of as a meta model for the target ontology that specifies how its entities map to Linked Data CMS entities (e.g., mapping OWL classes to browsing classes). This observation raises the question of whether the configuration could be represented directly in OWL 2.

OWL 2 offers some simple metamodeling capabilities (Motik et al., 2009b) that were not available in OWL DL since some of the restrictions on the separation between class, property and individual names have now been relaxed. This metamodelling support is achieved with punning, an extension to *SR**OIQ* (Horrocks et al., 2006), which makes it possible to use the same term to identify a class, individual or property simultaneously.

The punning approach allows for simple metamodeling in OWL 2 DL ontologies. For example a `Nissan Silvia` could be a class representing all Nissan Silvia cars ever manufactured; while `Nissan Silvia` could also simultaneously be an instance of the (meta)class of `Popular Japanese Sports Cars`. In this situation it is not desirable for each individual Nissan Silvia car to be in the class of Popular Japanese Sports Cars.

Our reasoning-driven configuration could also be represented using the OWL 2 punning approach (Golbreich and Wallace, 2010) by defining a meta ontology to represent the Linked Data CMS configuration (Section 6.2). This would allow the configuration to be defined using existing ontology modelling tools, such as Protégé (Knublauch et al., 2004).² This approach would give us the means to encode some of the configuration constraints as OWL axioms, allowing a reasoner to be used to automatically check the validity of a configuration by performing a consistency check of the configuration ontology.

Figure 7.2 shows an example of how a browsing class x could be defined. In this example, `a:Person` is used first as a class and then as an individual in an object property assertion axiom for the meta-property `cms:baseClass`; `foaf:knows` is also used as an object property and then as an individual.

The idea is that classes and properties from the domain ontology can be used as individuals

²Protégé is a popular OWL ontology editor: <http://protege.stanford.edu/>.

```
SubClassOf(a:Farmer a:Person)
ClassAssertion(a:Farmer a:Donald)
ObjectPropertyAssertion(foaf:knows a:Donald a:May)

ClassAssertion(cms:BrowsingClass _:x)
ObjectPropertyAssertion(cms:baseClass _:x a:Person)
ObjectPropertyAssertion(cms:hasRelationship _:x
    foaf:knows)
```

Figure 7.2: OWL 2 Punning: Linked Data CMS Configuration

when defining browsing classes using a class based browsing meta-ontology.

Appendix A

Acronyms and Abbreviations

BGP	SPARQL Basic Graph Pattern
CMS	Content Management System
Linked Data CMS	Linked Data Content Management System
LD-CMS	Linked Data CMS
CRUD	Create Retrieve Update Delete
DBMS	Database Management System
RDBMS	Relational Database Management System
DL	Description Logic
FSE	Folksonomy Search Expansion
ICE	Individual Class Expansion
IPE	Individual Property Expansion
CIE	Class Instance Expansion
IPVE	Individual Property Value Expansion
SCE	Sub Class Expansion
CSE	Class Sibling Expansion
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
J2EE	Java Platform Enterprise Edition
\mathcal{O}	An OWL (2) DL ontology

OWL	Web Ontology Language
OWL DL	Description Logic based fragment of OWL
OWL 2 DL	The second version of OWL DL
RDF	Resource Description Framework
RDFa	Resource Description Framework in Attributes
RDFS	RDF Schema; also written RDF(S)
RDF/XML	XML serialisation for RDF graphs
REST	REpresentational State Transfer
RIF	Rule Interchange Format
SPARQL	SPARQL Protocol and RDF Query Language
SPARQL 1.1	Latest SPARQL specification; including an UPDATE language
f-SPARQL	SPARQL extension supporting fuzzy queries
tf-idf	Term frequency–inverse document frequency
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Appendix B

Namespaces

Prefix	Namespace
	<code>http://example.org/example.owl#</code>
<code>ex</code>	<code>http://example.org/example.owl#</code>
<code>cms</code>	<code>http://example.org/ldcms.owl#</code>
<code>dc</code>	<code>http://purl.org/dc/terms/</code>
<code>foaf</code>	<code>http://xmlns.com/foaf/0.1/</code>
<code>mo</code>	<code>http://purl.org/ontology/mo/</code>
<code>owl</code>	<code>http://www.w3.org/2002/07/owl#</code>
<code>rdf</code>	<code>http://www.w3.org/1999/02/22-rdf-syntax-ns#</code>
<code>rdfs</code>	<code>http://www.w3.org/2000/01/rdf-schema#</code>
<code>skos</code>	<code>http://www.w3.org/2004/02/skos/core#</code>
<code>xsd</code>	<code>http://www.w3.org/2001/XMLSchema#</code>

The table above provides a list of namespaces used in the examples throughout this thesis. The prefixes in the Prefix column provide a way to abbreviate URIs using the namespaces shown in the Namespace column. For example the URI `http://xmlns.com/foaf/0.1/name` can be abbreviated as `foaf:name` by replacing the namespace part of the URI, `http://xmlns.com/foaf/0.1/`, with the prefix `foaf`. The namespace and local part of the URI are separated by a semi-colon. A blank prefix is given in the first row of the table, where for example `example:PhDStudent` is an abbreviation for `http://example.org/PhDStudent`.

Appendix C

OWL 2 Functional Syntax Mapping

In this appendix we present a reference mapping between the features of OWL 2 Functional-Style Syntax (Motik et al., 2009c) and DL syntax (Baader et al., 2003) used in the thesis. Table C.1 presents a mapping of OWL class expressions and OWL property expressions to DL class expressions and DL property expressions respectively, and Table C.2 presents a mapping of OWL axioms to DL axioms. For the model-theoretic semantics of DL axioms and class expressions we direct the reader to The Description Logic Handbook (Baader et al., 2003), and Motik et al. (2009b) for the OWL 2 direct semantics.

In the following mappings A denotes an atomic class, C is a class expression, P is a property expression, and o is an individual name.

OWL 2 Functional Syntax	DL Syntax
A owl:Thing owl:Nothing ObjectOneOf($o_1 \dots o_n$)	A \top \perp $\{o_1, \dots, o_n\}$
ObjectIntersectionOf($C_1 \dots C_n$) ObjectUnionOf($C_1 \dots C_n$) ObjectComplementOf(C)	$C_1 \sqcap \dots \sqcap C_n$ $C_1 \sqcup \dots \sqcup C_n$ $\neg C$
ObjectSomeValuesFrom($P C$) ObjectAllValuesFrom($P C$) ObjectHasValue($P o$)	$\exists P.C$ $\forall P.C$ $\exists P.\{o\}$
P owl:topObjectProperty ObjectInverseOf(P)	P U P^-

Table C.1: OWL 2 class and property expressions

OWL 2 Functional Syntax	DL Syntax
SubClassOf($C_1 C_2$)	$C_1 \sqsubseteq C_2$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$
DisjointClasses($C_1 \dots C_n$)	$C_1 \sqcap \dots \sqcap C_n \sqsubseteq \perp$
SubObjectPropertyOf($P_1 P_2$)	$P_1 \sqsubseteq P_2$
EquivalentObjectProperties($P_1 \dots P_n$)	$P_1 \equiv \dots \equiv P_n$
SymmetricObjectProperty(P)	$P \equiv P^-$
FunctionalObjectProperty(P)	$Func(P)$
TransitiveObjectProperty(P)	$Trans(P)$
ObjectPropertyDomain($P C$)	$domain(P) \sqsubseteq C$
ObjectPropertyRange($P C$)	$range(P) \sqsubseteq C$
SameIndividual($o_1 \dots o_n$)	$o_1 = \dots = o_n$
DifferentIndividuals($o_1 \dots o_n$)	$o_1 \neq \dots \neq o_n$
ClassAssertion($C o$)	$C(o)$
ObjectPropertyAssertion($P o_1 o_2$)	$P(o_1, o_2)$

Table C.2: OWL 2 axioms

Appendix D

Tag Ontology Overview

The Tag Ontology is an OWL DL ontology used by Taggr (Section 4.5) as a lightweight representation of a folksonomy. It represents the resources, tags, users, and tag assignments imported from external tagging systems. An overview of the main classes and properties in the Tag Ontology is provided below.

Class: `User`. A user in a tagging system.

Class: `TaggedResource`. A web resource that has been tagged in some tagging systems (Definition 2.12).

Class: `Tag`. A representation of tag in a folksonomy (Definition 2.8). Tags objects are specific to a particular tagging system, e.g., `http://www.flickr.com/photos/tags/Rock`. The literal value of a tag (“Rock” in this case) is associated with Tag individual using the `rdfs:label` property.

Class: `TagAssignment`. A tag assignment of a single Tag to a TaggedResource by a User.

Class: `Keyword`. An individual keyword for a Tag (Definition 2.5). For example the tag `http://www.flickr.com/photos/tags/Rock%20Music` has a lexical value of “Rock Music” and has two keywords “rock” and “music”. The literal values of these keywords are associated with Keyword individuals using the `rdfs:label` property.

Property: `hasTag`. The relationship between a TaggedResource and a TagAssignment.

Property: `hasKeyword`. The relationship between a Tag and a Keyword.

Property: `instanceOfTag`. The relationship between a TagAssignment and its tagging system-specific Tag.

Property: `taggedBy`. The relationship between a TagAssignment and the User who tagged it.

Appendix E

Taggr API Overview

As part of our reusable infrastructure for semantic mashup (Section 5.1) we have provided a simple Java client API for Taggr.

E.1 Overview

The API provides a set of Java classes that encapsulate querying the Taggr RESTful web service. More specifically, the API allows users to call the two main functions provided by Taggr: (i) the search function $S(k, expansionMethod)$ (Algorithm 4.1); (ii) the expansion function $E(k, expansionMethod)$ (Algorithm 4.2). These functions are known as *search query* and *expansion query* in the Taggr API.

E.2 Dependencies

Java SE 6 Java Standard Edition 6 or later is required. It is available from: <http://java.sun.com>

Jena 2.6 Jena RDF API version 2.6 or later is required. It is available from: <http://jena.sourceforge.net>

E.3 Class Summary

This section provides an overview of the main front end classes provided by the Taggr API. Some classes and methods have been omitted to simplify presentation.

E.3.1 Tagged Resource

This class represents tagged resources (Definition 2.12) in the tag ontology.

Constructor Summary

Resource(String uri)

Construct a new resource with the specified URI.

Method Summary

String getUri()

Returns the URI of this resource.

String getTitle()

Returns the title of this resource.

void setTitle(String title)

Sets the title of this resource.

Set<String> getTags()

Returns the set of tags for this resource.

void setTags(Set<String> tags)

Sets the tags for this resource.

E.3.2 Keyword Set

This class represents a keyword set (Definition 2.6). This class provides a simple wrapper around

`java.util.LinkedHashSet<String>`.

Constructor Summary

KeywordSet ()

Creates a new, empty keyword set.

KeywordSet(Collection<String> k)

Creates a keyword set containing the specified collection of keywords.

Method Summary

Methods are inherited from `java.util.LinkedHashSet`: <http://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html>

E.3.3 Expansion Query

This class executes a specified expansion method against a specified ontology in the TrOWL repository and returns a keyword group (Definition 2.7). This is an **abstract class** used for defining expansion methods.

Method Summary

String getExpansionMethodName()

Returns the expansion method for this query.

Set<String> getKeywords()

Returns the keywords (search terms) for this query.

Map<String, KeywordSet> query()

Sends the query and returns the results from Taggr. The returned object is a mapping from each resource URI identified by the expansion, to its keyword set.

void setOntology(String ontologyURI)

Sets the target ontology. If no target ontology is specified then all ontologies in the TrOWL repository will be used for expansion.

void setRetain(boolean retain)

Sets whether or not the expansion method should *retain* the original search terms in the expanded keyword sets.

void setSourceThreshold(float threshold)

Sets the fuzzy threshold of the keywords used to identify the domain source entity for expansion.

void setTargetThreshold(float threshold)

Sets the fuzzy threshold of the keywords for target resources identified by the expansion.

E.3.4 Search Query

This class uses the specified expansion method to search for tagged resources in Taggr's tag ontology, and returns a set of tagged resources with associated metadata. Optionally a search query can be run without using an expansion method.

Constructor Summary

SearchQuery(ExpansionQuery expansion)

Creates a new SearchQuery with the specified expansion method.

SearchQuery(Set<String> keywords)

Creates a new SearchQuery with the specified keywords and no expansion.

Method Summary

Set<TaggedResource> query()

Sends the search request query to the Taggr RESTful API and returns a set of tagged resources.

E.3.5 Sub Class Expansion

Sub Class Expansion (SCE) searches for classes which are identified by the set of specified keywords. Keywords relating to sub classes of these classes identified are then returned. This class extends **ExpansionQuery**.

Direct or indirect expansion can be performed. Direct expansion identifies only the classes which are asserted to be direct sub classes of the identified classes. Indirect expansion identifies any classes which are entailed to be a sub class of the identified class. Indirect expansion is performed by default.

This class can either be passed to **SearchQuery** or executed directly as an **ExpansionQuery**.

Constructor Summary

SubClassExpansion(Set<String> keywords)

Creates a new SubClassExpansion with the specified keywords for expansion.

Method Summary

void setDirect(boolean direct)

Specifies whether direct or indirect expansion should be performed.

*The remainder of the methods are inherited from **ExpansionQuery***

E.3.6 Individual Property Value Expansion

Individual Property Value Expansion (IPVE) uses the data present in the ABox of the ontology. The expansion identifies individuals for the set of specified keywords, and expands this by finding other related individuals via the specified property. Keywords related to these individuals

are returned.

The domain and range of the expansion can be specified. Domain refers to the type of the individuals identified by the specified set of keywords. Range refers to the type of the individuals identified via the specified property.

This expansion method can find relationships between objects directly or inversely. This can be specified with **setInverse(boolean)**. The default expansion is performed directly.

This class can either be passed to **SearchQuery** or executed directly as an **ExpansionQuery**.

Constructor Summary

IndividualPropertyValueExpansion(Set<String> keywords, String propertyURI)

Creates a new IndividualPropertyValueExpansion with the specified keywords and property.

Method Summary

void setInverse(boolean inverse)

Specifies whether this expansion should be inverse or not.

void setDomainClass(String classURI)

Specifies the URI of the domain class used in the expansion.

void setRangeClass(String classURI)

Specifies the URI of the range class of the expansion.

*The remainder of the methods are inherited from **ExpansionQuery***

Appendix F

Walkthrough: Drupal Linked Data CMS

In the following sections we provide some further details of our Linked Data CMS implementation in Drupal. In particular we show how the Drupal Linked Data CMS has been implemented in various Drupal 7 modules.

CURIOS: Linked Data CMS is a Drupal 7 module that provides the functionality to perform CRUD operations on RDF data stored in a triple store. The module makes use of a configuration that provides a mapping between OWL classes/properties and Drupal entities. This version of the module is intended for use with the Hebridean Connections case study (Section 6.4) for the CURIOS project (Beel et al., 2013).

F.1 Overview

The pages in the Linked Data CMS are divided up into two main categories. The *front end user interface* provides an interface for users to search and browse individuals in the repository, and allows privileged users to update individuals. The *back end user interface* provides the functionality for site administrators to maintain the structure of the CMS.

F.2 Requirements and Dependencies

The *CURIOS: Linked Data CMS* module depends on a number of other contributed Drupal modules. The majority of these are available as release from <http://drupal.org/>. The SPARQL, SPARQL Views and ARC2 projects have been forked specifically for this project and are available from <https://github.com/stuartraetaylor/>. A list of the core requirements and dependencies of the Linked Data CMS is available below.

CURIOS: Linked Data CMS 0.2.0 <https://github.com/stuartraetaylor/curios>

PHP 5.3+ <http://www.php.net/>

Jena Fuseki 1.0.0+ http://jena.apache.org/documentation/serving_data/

Drupal 7.23+ <http://drupal.org/download>

Entity API 7.x-1.2 <https://drupal.org/project/entity>

Views 7.x-3.7 <http://drupal.org/project/views>

CTools 7.x-1.3 <https://drupal.org/project/ctools>

Lightbox2 <http://drupal.org/project/lightbox2>

Libraries API 7.x-2.1 <https://drupal.org/project/libraries>

Views Litepager 7.x-3.0 https://drupal.org/project/views_litepager

Panels 7.x-3.x-dev <http://drupal.org/project/panels>

RDFx 7.x-2.x-dev <https://drupal.org/project/rdfx>

ARC2 PHP Library <https://github.com/stuartraetaylor/arc2-sparql11>

SPARQL sparql11-0.1.0 <https://github.com/stuartraetaylor/sparql>

SPARQL Views curios-0.1.0 https://github.com/stuartraetaylor/sparql_views

Installation and download instructions for the Linked Data CMS module can be found on the GitHub project page: <https://github.com/stuartraetaylor/curios>.

F.3 Front End User Interface

The front end user interface allows the user to search, view and update individuals in the Linked Data CMS repository.

F.3.1 Listings View

The listings view displays a summary of the individuals in a particular browsing class. It also allows users to search by a set of predefined fields, and filter the results by type (sub classes of the base class in the ontology).

Figure F.1 shows the listing view being filtered by the string “Stuart”. The filtering is performed by a full text search which indexes instances of the `dc:title` and `dc:description` properties in the triple store.

The screenshot shows a web application interface. At the top is a blue navigation bar with a 'Home' link. Below the navigation bar, on the left, is a sidebar with a 'Subject Types' section containing a list of categories: Boats, Buildings and Public Amenities, Businesses, Crofts and Residences, Gaelic Verseb Rdachds, Historical Events, Landmarks and Archaeological Sites, Locations, Natural Landscape Features, Objects and Artefacts, Organisations, People, Resources, Story Reports and Traditions, Vehicles, and ALLSUBJECTS. The 'People' category is selected. Below the sidebar is a 'User login' section with a 'Username' field and a red asterisk. The main content area is titled 'People' and includes a subtitle: 'From this page you can both browse and search the Hebridean Connections repository.' Below this is a 'Search People' section with a search input field containing 'Stuart', an 'Items per page' dropdown set to '20', and a 'Search' button. Below the search section is a table with three columns: 'Title', 'Description', and 'Files'. The table contains seven rows of data, each representing a person whose name contains 'Stuart'.

Title	Description	Files
Stuart McLean	Stuart McLean, Paisley, married Chirsty Ann Macphail from 19 Gravir; they lived in Lanarkshire until their emigration to Canada in 1957. The had five children: Alasdair 1944, Iain...	true
Stuart Donald Macdonald		
Charles Stuart Hunter	Charles Stuart Hunter was the first husband of Janet Harrower. There were two children from this marriage.	
Kenneth Stuart Macdonald	Kenneth, born in 1878 was the eldest child of Donald and Christina nee Maclean, emigrants from Lewis to Ontario. He was married to Mary Matheson Macivor Mackay, Algoma, Ontario.	
Stuart Macaulay	Stuart was a son of Neil John and Catherine nee Hutchison.	
Charles Stuart Hunter	Charles Stuart, son of Charles Hunter and Janet Harrower was unmarried.	
Stuart Taylor	PhD Student at the University of Aberdeen	

Figure F.1: Searching *people* using full-text search.

Figure F.2 shows the listing view being filtered by sub-type. Each item in the *Filter* drop down is a subclass of the browsing class’s base class in the ontology. In this example, the drop down allows the records to be filtered by subclasses of the `Boat` class.

The screenshot shows the 'Boats' section of the Hebridean Connections repository. On the left is a sidebar with 'Subject Types' including Boats, Buildings and Public Amenities, Businesses, Crofts and Residences, Gaelic Verse/Rdachs, Historical Events, Landmarks and Archaeological Sites, Locations, Natural Landscape Features, Objects and Artefacts, Organisations, People, Resources, Story Reports and Traditions, Vehicles, and ALL SUBJECTS. Below this is a 'User login' section with a 'Username' field. The main content area is titled 'Boats' and includes a search bar, a filter dropdown set to 'Fishing Boat', and an 'Items per page' dropdown set to '20'. A 'Search' button is also present. Below these controls is a table of boat records.

Title	Description	Files
Jessie	The Jessie SY1538 was owned by Andrew Mackay and skippered by Malcolm Mackay out of Valtos. 2 masts and lugsails; 8 oars; fishing lines.7 ton; 28.5' keel, crew of seven.	
Sir James Matheson	The Sir James Matheson SY515 was registered in 1871 by owner Kenneth Smith (Stornoway) under skipper Alexander McLeod of Carloway. However in 1869 she appears to have been sailing...	
Peggy	The Peggy SY680 was owned and skippered by Peter Macritchie out of Aird Uig. One mast and lugsail; fishing lines.1.5 tons, 14' keel, crew of four.	
Foam	The Foam SY1424 was owned by Kenneth Smith (Stornoway) and skippered by John Macleod out of Mangersta. 2 masts and lugsails; 8 oars; jib; fishing lines 7 ton, 25' keel, crew of...	
Lord Clyde	Lord Clyde SY253 was owned by Kenneth Smith (Stornoway) and skippered by Malcolm Macaulay out of Brenish. Two masts and lugs; fishing lines. 8 tons, 23' keel; crew of eight.	
Kitty Ann	The Kitty Ann SY1690 was owned by Hector Matheson and skippered by Malcolm Macdonald out of Valtos. 2 masts and lugsails; lines and creels. 7 ton; 30' keel; crew of eight.	
Scotch Lass	Boat - Scotch Lass Reg. 1953 No. SY 59734ft keel Ketch rig with Kelvin diesel 6615.3 ton 4 man crew Line fishing, nets and creels Owner/Skipper - George Clark Sold 1955 Article that...	true
Maggie	The Maggie SY1568 was owned by Hector Matheson and skippered by John Macaulay out of Valtos. Evidently she was bought from Murdo Maclean, merchant of Valtos, and was fishing in...	
	The Mary Ann SY807 was owned and skippered by Donald Maclean out of Mangersta. One mast	

Figure F.2: Filtering *boats* by sub-type.

F.3.2 Details View

The details view displays the full set of fields and relationships for a browsing class. In the Drupal Linked Data CMS these are divided into four distinct areas: summary, properties (field listing), relationships, and image gallery.

Figure F.4 and Figure F.5 show the details view for an Archaeological Site and Person records respectively. The person record shown in Figure F.5 has the view's four areas highlighted with a border to show how they are arranged on the screen.

The SPARQL queries in Figure F.3 are generated by the SPARQL Views module using the browsing classes defined in the Linked Data CMS module. Each query is used to provide the data to the details view in Figure F.4. Note that the queries related to relationships have been limited to only the *located at* relationship for illustration purposes.

```

# Summary area.
PREFIX hc:
  <http://www.hebrideanconnections.com/hebridean.owl#>
SELECT ?site_title ?site_description WHERE {
  ?site rdf:type hc:ArchaeologicalSite;
  hc:englishName ?site_title.
  OPTIONAL { ?site hc:description ?site_description }
FILTER (?site = hc:52246) }

# Properties area.
PREFIX hc:
  <http://www.hebrideanconnections.com/hebridean.owl#>
SELECT ?site_subject_id ?site_english_name
  ?site_gaelic_name
  ?site_period ?site_northing ?site_easting WHERE {
  ?site a hc:ArchaeologicalSite.
  OPTIONAL { ?site hc:subjectID ?site_subject_id }
  OPTIONAL { ?site hc:englishName ?site_english_name }
  OPTIONAL { ?site hc:gaelicName ?site_gaelic_name }
  OPTIONAL { ?site hc:period ?site_period }
  OPTIONAL { ?site hc:northing ?site_northing }
  OPTIONAL { ?site hc:easting ?site_easting }
FILTER (?site = hc:52246) }

# Example relationship (located at).
PREFIX hc:
  <http://www.hebrideanconnections.com/hebridean.owl#>
SELECT ?subject_id ?title ?type WHERE {
  ?site a hc:ArchaeologicalSite.
  OPTIONAL {
    ?site hc:locatedAt ?site_located_at.
    ?site_located_at hc:subjectID ?subject_id;
    hc:englishName ?title;
    a ?type. }
FILTER (?site = hc:52246) }

```

Figure F.3: SPARQL queries required to generate a page about an archaeological site.

subject types

- Boats
- Buildings and Public Amenities
- Businesses
- Crofts and Residences
- Gaelic Verseb Rdachds
- Historical Events
- Landmarks and Archaeological Sites
- Locations
- Natural Landscape Features
- Objects and Artefacts
- Organisations
- People
- Resources
- Story Reports and Traditions
- Vehicles
- ALL SUBJECTS

Dun Carloway Broch

Dun Carloway is a well-preserved broch dominating the village of Doune Carloway. It likely dates from the last centuries BC.



It is now thought that brochs were "status symbol" dwellings as much as they were defensive forts. It rises to 30 feet, and its external diameter is 47 feet. According to local tradition, it was occupied intermittently until the 17th century.

In the 16th century the Dun was attacked by Donald Cam Macaulay, who climbed up the outside on the points of daggers to throw burning heather inside to smoke out the Morrisons, who were sheltering inside after a cattle raid.

There is some evidence that another much smaller broch was to be found close by - on an islet across from the present-day Doune Braes Hotel.

[Back to listing](#)

Subjectid:	52246
English Name:	Dun Carloway Broch
Gaelic Name:	Dn Charlabhaigh
Period:	Iron Age (pre-Roman) (800-0 BC)
Northing:	941281
Easting:	119033

Associated With
[Donald Macaulay](#)

Located At
[UK Mainland](#)
[Isle of Lewis](#)
[Carloway](#)
[Scotland](#)
[Western Isles](#)
[Doune Carloway](#)
[Outer Hebrides](#)

User login

Username *

Password *

Figure F.4: Details view for an archaeological site. The page generated using a set SPARQL queries, required to display each area of the page.

Home

Home

Subject Types

- Boats
- Buildings and Public Amenities
- Businesses
- Crofts and Residences
- Gaelic Verseb Rdachds
- Historical Events
- Landmarks and Archaeological Sites
- Locations
- Natural Landscape Features
- Objects and Artefacts
- Organisations
- People
- Resources
- Story Reports and Traditions
- Vehicles
- ALL SUBJECTS



User login

Dolina Macdonald

Dolina Macdonald, Doileag a Ghreusaich (1898-1990), was the daughter of Donald Macdonald and Peggy Macritchie from Geshader. Dolina was married to Finlay Macleod from 2 Enaclete, where they both lived and raised a family.

[Back to listing](#)

Subjectid:	2742
English Name:	Dolina Macdonald
Bk Reference:	68
Born:	1898-12-17
Date Of Death:	1991-03-03
Sex:	Female

Associated With
[Oral Tradition](#)

Child Of
[Donald Macdonal...](#)
[Margaret Macrit...](#)

Lived At
[2 Enaclete](#)
[1 Geshader](#)

Married
[Finlay Macleod](#)

Parent Of
[Dolly Alice Mac...](#)
[Margaret Cather...](#)

ninaltest.local/crofts-and-residences/3097

Figure F.5: Details view of a person page instance. The relationships are displayed on the right most pane. The areas displayed in the centre pane from top to bottom are: summary, properties, image gallery.

F.3.3 Update Views

There are two update views provided by the Drupal Linked Data CMS: create and edit. The create view allows the creation of new individuals in the triple store using the browsing classes defined in the configuration. The edit view allows the user to update field and relationship values, this view also allows the user to delete the entire record. A record in this sense is the set of properties relating to the fields and relationships for a browsing class.

Figure F.6 shows the update view for creating a new record in the triple store. At this stage the name and description fields are displayed. Once the record is created, the user is forwarded to the edit record screen.

The SPARQL query in Figure F.7 is generated to create the new record when the user submits the form shown in Figure F.6. A new subject URI, in this case `hc:95543`, is generated each time a new record is created.

The screenshot shows a web interface titled 'Manage Records'. On the left is a sidebar with 'Record Types' (listing categories like Boats, Buildings and Public Amenities, Businesses, etc.) and 'Navigation' (with a link to 'Add content'). The main area has three buttons: 'Create New Record', 'Edit / Delete Record', and 'Admin Tools'. Below these, it says 'Complete the form below to create a new record.' and 'Create new: **Person**'. There is a text input for 'English Name *' containing 'Stuart Taylor' with a label '(hc:englishName)' to its right. Below that is a 'Description' section with a rich text editor toolbar (containing icons for source, undo, redo, bold, italic, underline, strikethrough, link, unlink, list, etc.) and a text area containing 'PhD Student at the University of Aberdeen.' with a label '(hc:description)' at the bottom.

Figure F.6: Creating a new *person* record.

```
# Insert new record.
PREFIX hc:
  <http://www.hebrideanconnections.com/hebridean.owl#>
INSERT DATA {
  hc:95543 a ;
    hc:englishName "Stuart Taylor";
    hc:description "PhD Student at the University of
      Aberdeen."; }
```

Figure F.7: SPARQL query generated to create a new person record.

Once a record has been created, the user can then update the values stored in the triple store. Figure F.8 shows the edit view for updating these values. Figure F.9 shows the SPARQL query generated by the Linked Data CMS module to update the *English Name* field.

Record Types

- Boats
- Buildings and Public Amenities
- Businesses
- Crofts and Residences
- Gaelic Verse
- Historical Events
- Historical/ Archaeological Sites
- Image Details
- Locations
- Natural Landscape Features
- Objects and Artefacts
- Organisations
- People
- Resources
- Sound Files
- Stories, Reports and Traditions
- Vehicles

Navigation

- Add content

Manage Records

Create New Record Edit / Delete Record Admin Tools

Edit the form below to update this record.

Edit: **Person**; ID: **95543**; Name: **Stuart Taylor**

English Name *

Stuart Rae Taylor (hc:englishName)

Description

PhD Student at the University of Aberdeen.

body (hc:description)

Figure F.8: Updating the newly created person instance.

```
# Update existing record.
PREFIX hc:
  <http://www.hebrideanconnections.com/hebridean.owl#>
DELETE {
  ?subject
    hc:englishName "Stuart Taylor";
}
INSERT {
  ?subject
    hc:englishName "Stuart Rae Taylor";
}
WHERE {
  ?subject ?p ?o ;
    a hc:Person .
  FILTER ( ?subject = hc:95543 ) }
```

Figure F.9: SPARQL query to update the relevant fields for the new person instance.

Figure F.10 shows the user updating the other fields for the person browsing class. Figure F.11 shows the user adding relationships to the record. The dropdown lists in the 'Add

Relationship' fieldset allows the user to select the relationships defined for the browsing class.

Figure F.12 show the person record updated with field and relationship values.

Fields / Attributes

Gaelic Name
 (hc:gaelicName)

Also Known As
 (hc:alsoKnownAs)

Date of Birth
 (hc:born) [Edit Date]

Date of Death
 (hc:dateOfDeath) [Edit Date]

Date of Marriage
 (hc:dateOfMarriage) [Edit Date]

Occupation
 (hc:occupation)

Sex
 (hc:sex)

Bk Reference
 (hc:bkReference)

Figure F.10: Updating field values for a person page instance.

Existing Relationships

Associated With:

Mary (633) [Delete]

Add Relationship

To add a relationship, select the relationship type from the drop down menu and enter the SubjectId of the record in the textbox.

Child Of [v] [Remove]

Emigrated To [v] [Add]

Recently Created Records

Delete Record

[Save Changes] [Save & View record] [Discard Changes]

Figure F.11: Updating relationships for a person page instance.

Figure F.12: Viewing the updated person page instance.

The record shown in Figure F.12 can be deleted via the edit page by expanding the Delete Record fieldset and then clicking the Delete Record button highlighted in Figure F.13. The SPARQL query in Figure F.14 is generated to delete the record when the user clicks the Delete Record button in the edit form.

Figure F.13: Deleting a person record from the edit view.

```
# Delete record.  
PREFIX hc:  
  <http://www.hebrideanconnections.com/hebridean.owl#>  
DELETE { ?subject ?p ?o }  
WHERE {  
  ?subject ?p ?o .  
  FILTER ( ?subject = hc:95543 ) }
```

Figure F.14: SPARQL query to delete the person record.

F.4 Back End User Interface

The back end user interface allows site administrators to re-configure the modules initially configured by the Linked Data CMS. This part of the Linked Data CMS module reuses the interfaces provided by the SPARQL Views, Views and Panels modules.

F.4.1 SPARQL Views Resource Types

The SPARQL Views resource types shown in Figure F.15 provide the RDF mappings for the set of fields and relationships for each browsing class. These mappings are used when generating the SPARQL queries for the listings, details and update views.

The Drupal fields shown in Figure F.16 are created using the fields and relationships defined for the browsing class. Figure F.17 shows the RDF mapping predicate mapping for the English Name field.

SPARQL Views resource types

[+ Add sparql views resource type](#)
[+ Import sparql views resource type](#)

LABEL	STATUS	OPERATIONS					
Subject (Machine name: hc_subject)	Custom	edit	manage fields	manage display	clone	delete	export
Boat (Machine name: hc_boat)	Custom	edit	manage fields	manage display	clone	delete	export
BuildingOrPublicAmenity (Machine name: hc_building_or_public_amenity)	Custom	edit	manage fields	manage display	clone	delete	export
Business (Machine name: hc_business)	Custom	edit	manage fields	manage display	clone	delete	export
CroftOrResidence (Machine name: hc_croft_or_residence)	Custom	edit	manage fields	manage display	clone	delete	export
GaelicVerseBrdachd (Machine name: hc_gaelic_verseb_rdachd)	Custom	edit	manage fields	manage display	clone	delete	export
HistoricalEvent (Machine name: hc_historical_event)	Custom	edit	manage fields	manage display	clone	delete	export
LandmarkOrArchaeologicalSite (Machine name: hc_landmark_or_archaeological_site)	Custom	edit	manage fields	manage display	clone	delete	export
Location (Machine name: hc_location)	Custom	edit	manage fields	manage display	clone	delete	export

Figure F.15: Listing of all resource types (browsing classes).

LABEL	MACHINE NAME	FIELD TYPE	WIDGET	OPERATIONS
+ dc:title	dc_title	Text	Text field	edit delete
+ dc:description	dc_description	Text	Text field	edit delete
+ Subjectid	hc_subject_id	Text	Text field	edit delete
+ EnglishName	hc_english_name	Text	Text field	edit delete
+ GaelicName	hc_gaelic_name	Text	Text field	edit delete
+ Period	hc_period	Text	Text field	edit delete
+ PeriodOfConstructionUse	hc_period_of_construction_use	Text	Text field	edit delete
+ SMR Record ID	hc_smr_record_id	Text	Text field	edit delete
+ Northing	hc_northing	Text	Text field	edit delete
+ Easting	hc_easting	Text	Text field	edit delete
+ AssociatedWith	hc_associated_with	Related Resource	Select list	edit delete
+ InformationObtainedFrom	hc_information_obtained_from	Related Resource	Select list	edit delete
+ HasFile	hc_has_file	Related Resource	Select list	edit delete
+ SourceOfInformationOn	hc_source_of_information_on	Related Resource	Select list	edit delete
+ ComprisesSubSite	hc_comprises_sub_site	Related Resource	Select list	edit delete
+ IsPartOfSite	hc_is_part_of_site	Related Resource	Select list	edit delete
+ LivedHere	hc_lived_here	Related Resource	Select list	edit delete

Figure F.16: Listing of fields and relationships for the Archaeological Site browsing class.

Maximum number of values users can enter for this field.
'Unlimited' will provide an 'Add more' button so the users can add as many values as they like.

Maximum length *

255

The maximum length of the field in characters.

ENGLISHNAME RDF MAPPING

RDF Predicates

hc:englishName

Enter a comma-separated list of predicates for *EnglishName* using CURIE syntax. For example: *foaf:familyName*, *foaf:lastName*

Attribute Type

property

For fields containing literals—things such as plain text, html, numbers—use the **property** attribute. For fields containing references to other things—urls and node references, for example—use the **rel** or **rev** attribute.

Datatype

Enter the datatype for *EnglishName* using CURIE syntax. For a list of common datatypes, see [XML Schema datatypes](#).

Save settings

Figure F.17: RDF mapping for *english name* field.

F.4.2 Views Specification

The views specification defines the fields that will be used in each part of the listing and details views (summary, properties, relationships, and image gallery areas). Each view shown in Figure F.18 corresponds to a page template generated by the mapping algorithm (Algorithm 6.1).

Figure F.19 shows the Views configuration screen generated by the Linked Data CMS module for the summary area. Figure F.20 shows the Views configuration screen generated for each field defined for the Archaeological Site browsing class. Figure F.21 shows the Views configuration screen for the *located in* relationship. Each of these Views configurations can be edited by the user using the standard Drupal Views module admin interface.

VIEW NAME	DESCRIPTION	TAG	PATH	OPERATIONS
Boats Display: <i>Block</i> In code Type: SPARQL Views: Hebridean Connections	Boats (hc_boats)	CURIOS		edit
Buildings and Public Amenities Display: <i>Block</i> In code Type: SPARQL Views: Hebridean Connections	Buildings and Public Amenities (hc_buildings_and_public_amenities)	CURIOS		edit
Businesses Display: <i>Block</i> In code Type: SPARQL Views: Hebridean Connections	Businesses (hc_businesses)	CURIOS		edit
Crofts and Residences Display: <i>Block</i> In code Type: SPARQL Views: Hebridean Connections	Crofts and Residences (hc_crofts_and_residences)	CURIOS		edit
Gaelic Verseb Rdachds Display: <i>Block</i> In code	Gaelic Verseb Rdachds (hc_gaelic_verseb_rdachds)	CURIOS		edit

Figure F.18: Listing of all views (page templates).

Displays

Master Listing **Summary** Properties Images Object: AssociatedWith edit view name/description

Object: InformationObtain... Object: HasFile Object: SourceOfInformati...

Object: ComprisesSubSite Object: IsPartOfSite Object: LivedHere Object: LocatedAt

Object: LocatedHere + Add

▼ Summary details

Display name: Summary clone summary

TITLE

Title: None

FORMAT

Format: Unformatted list | Settings

Show: Fields | Settings

FIELDS add

LandmarkOrArchaeologicalSite: URI (URI)

LandmarkOrArchaeologicalSite: dc:title

LandmarkOrArchaeologicalSite: dc:description

Search Link

FILTER CRITERIA add

SORT CRITERIA add

BLOCK SETTINGS

Block name: None

Access: None

HEADER add

FOOTER add

PAGER

Use pager: Display all items | All items

More link: No

▼ Advanced

CONTEXTUAL FILTERS add

LandmarkOrArchaeologicalSite: URI

LandmarkOrArchaeologicalSite: RDF type

RELATIONSHIPS add

NO RESULTS BEHAVIOR add

Global: Text area (No results)

EXPOSED FORM

Exposed form style: Basic | Settings

OTHER

Machine Name: summary

Comment: No comment

Use AJAX: Yes

Hide attachments in summary: No

Figure F.19: Views configuration interface for the summary area.

Displays

Master Listing Summary **Properties** Images Object: AssociatedWith edit view name/description

Object: InformationObtain... Object: HasFile Object: SourceOfInformati...

Object: ComprisesSubSite Object: IsPartOfSite Object: LivedHere Object: LocatedAt

Object: LocatedHere + Add

▼ Properties details

Display name: Properties clone properties

TITLE

Title: None

FORMAT

Format: Unformatted list | Settings

Show: Fields | Settings

FIELDS add

LandmarkOrArchaeologicalSite: URI (URI)

LandmarkOrArchaeologicalSite: Subjectid (Subjectid)

LandmarkOrArchaeologicalSite: EnglishName (English Name)

LandmarkOrArchaeologicalSite: GaelicName (Gaelic Name)

LandmarkOrArchaeologicalSite: Period (Period)

LandmarkOrArchaeologicalSite:

BLOCK SETTINGS

Block name: None

Access: None

HEADER add

FOOTER add

PAGER

Use pager: Display all items | All items

More link: No

▼ Advanced

CONTEXTUAL FILTERS add

LandmarkOrArchaeologicalSite: URI

LandmarkOrArchaeologicalSite: RDF type

RELATIONSHIPS add

NO RESULTS BEHAVIOR add

Global: Text area (No results)

EXPOSED FORM

Exposed form style: Basic | Settings

OTHER

Machine Name: properties

Comment: No comment

Use AJAX: Yes

Hide attachments in summary: No

Figure F.20: Views configuration interface for the properties area.

Displays

Master Listing Summary Properties Images Object: AssociatedWith edit view name/description

Object: InformationObtain... Object: HasFile Object: SourceOfInformati...

Object: ComprisesSubSite Object: IsPartOfSite Object: LivedHere Object: LocatedAt

Object: LocatedHere + Add

▼ **Object: LocatedAt details**

Display name: Object: LocatedAt clone object: locatedat

<p>TITLE</p> <p>Title: Located At</p> <p>FORMAT</p> <p>Format: Unformatted list Settings</p> <p>Show: Fields Settings</p> <p>FIELDS add</p> <p>(LocatedAt) Subject: Subjectid</p> <p>(LocatedAt) Subject: dc:title</p> <p>(LocatedAt) Subject: RDF type</p> <p>Global: Link Target Rewriter</p> <p>FILTER CRITERIA add</p> <p>SORT CRITERIA add</p>	<p>BLOCK SETTINGS</p> <p>Block name: None</p> <p>Access: None</p> <p>HEADER add</p> <p>FOOTER add</p> <p>PAGER</p> <p>Use pager:</p> <p>Display a specified number of items 100 items</p> <p>More link: No</p>	<p>▼ Advanced</p> <p>CONTEXTUAL FILTERS add</p> <p>LandmarkOrArchaeologicalSite: URI</p> <p>LandmarkOrArchaeologicalSite: RDF type</p> <p>RELATIONSHIPS add</p> <p>LandmarkOrArchaeologicalSite: LocatedAt</p> <p>NO RESULTS BEHAVIOR add</p> <p>EXPOSED FORM</p> <p>Exposed form style: Basic Settings</p> <p>OTHER</p> <p>Machine Name:</p> <p>object_hc_landmark_or_archaeological_site_hc_</p> <p>Comment: No comment</p> <p>Use AJAX: Yes</p>
--	--	--

Figure F.21: Views configuration interface for a relationship area.

F.4.3 Panels Layouts

The panels layout provides a template for displaying and positioning the individual views on the screen. The layout shown in Figure F.22 shows the areas defined for the details view.

Category

CURIOS

What category this layout should appear in. If left blank the category will be "Miscellaneous".

▼ Canvas

▼ Column

▼ Row

▼ Column

▼ Row

▼ Region

Summary Region

▼ Row

▼ Region

Properties Region

▼ Row

▼ Region

Image Gallery Region

▼ Column

▼ Row

▼ Region

Relationships Region

Live preview

Preview

Save

Figure F.22: Panels layout template for details pages.

F.4.4 Panels Pages

The Panels pages shown in (Figure F.23) define the *path* and template used to display the views for each browsing class. The views (Section F.18) are automatically populated into a Panels pages.

TYPE	NAME	TITLE	PATH	STORAGE	OPERATIONS
Custom	page-hc_people_panel_page	People	/people/largs	In code	Edit
Custom	page-hc_organisations_panel_page	Organisations	/organisations/largs	In code	Edit
Custom	page-hc_resources_panel_page	Resources	/resources/largs	In code	Edit
Custom	page-hc_vehicles_panel_page	Vehicles	/vehicles/largs	In code	Edit
Custom	page-hc_boats_panel_page	Boats	/boats/largs	In code	Edit
Custom	page-hc_buildings_and_public_amenities_panel_page	Buildings and Public Amenities	/buildings-and-public-amenities/largs	In code	Edit
Custom	page-hc_businesses_panel_page	Businesses	/businesses/largs	In code	Edit
Custom	page-hc_crofts_and_residences_panel_page	Crofts and Residences	/crofts-and-residences/largs	In code	Edit
Custom	page-hc_gaelic_verseb_rdachds_panel_page	Gaelic Verseb Rdachds	/gaelic-verseb-rdanchds/largs	In code	Edit
Custom	page-hc_historical_events_panel_page	Historical Events	/historical-events/largs	In code	Edit
Custom	page-hc_landmarks_and_archaeological_sites_panel_page	Landmarks and Archaeological Sites	/landmarks-and-archaeological-sites/largs	In code	Edit
Custom	page-hc_locations_panel_page	Locations	/locations/largs	In code	Edit

Figure F.23: List of all panels pages for the browsing classes.

Clone

Export

Disable

Add variant

Import variant

Reorder variants

Summary

Settings

Arguments

Access

Menu

Variants

Details

Listing

Summary

Get a summary of the information about this page.

Panel Page: Landmarks and Archaeological Sites (hc_landmarks_and_archaeological_sites)

Storage

In code

Status

Enabled

Disable

Path

</landmarks-and-archaeological-sites/largs>

Edit

Access

This page is publicly accessible.

Edit

Menu

Normal menu entry. Title: *Landmarks and Archaeological Sites*. Menu block: *Subject Types*.

Edit

Panel: Details

Clone

Export

Disable

Storage

In code

Status

Enabled

Disable

Selection rule

This panel will be selected if args exists.

Edit

Layout

Subject Page Layout

Change layout

Edit content

Preview

Panel: Listing

Clone

Export

Disable

Storage

In code

Figure F.24: Panel page configuration for a single browsing class show the two display types.

There are two variants of each panels page shown in Figure F.24: the listing page and the details page. The details page shown in Figure F.25 uses the layout from Section F.4.3 to populate the views described in Section F.4.2 for each browsing class.

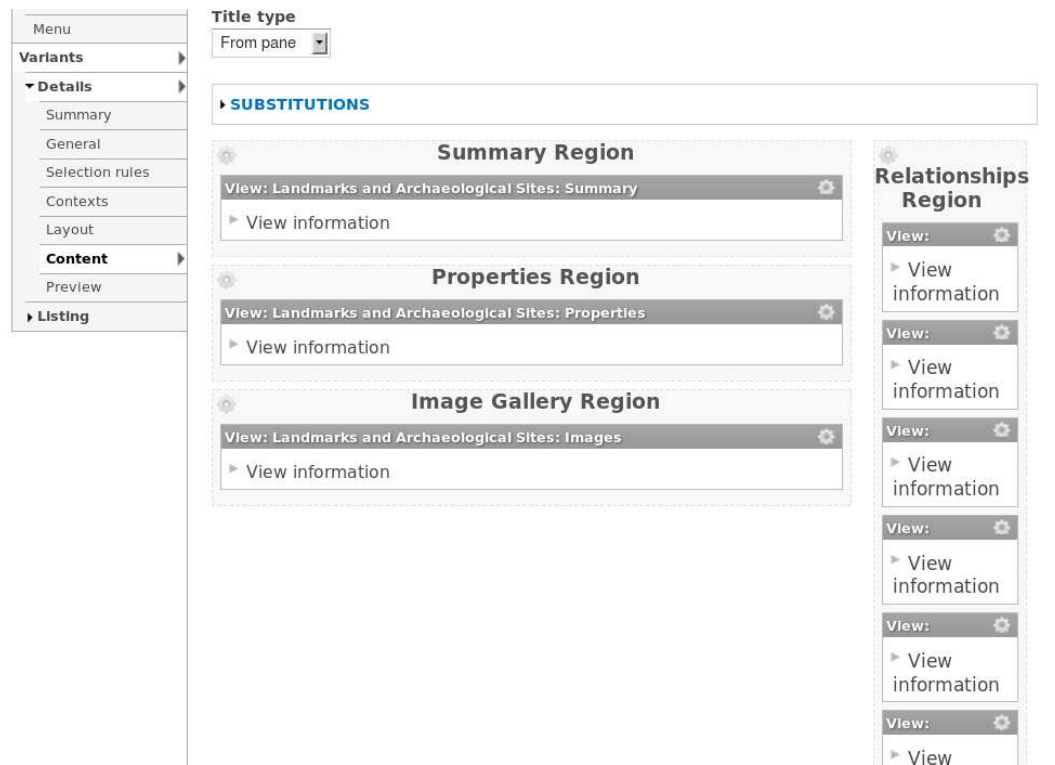


Figure F.25: Details display configuration, showing the placement of the views using the panels layout template.

Appendix G

Example Drupal Linked Data CMS Configuration

This appendix provides an example of how a configuration can be defined in the Drupal Linked Data CMS module described in Section 6.3.

In this example, a set of fields and relationships are defined in the `$fields[]` array, then the browsing classes are defined with their base class in the `$browsing-classes[]` array. The fields and relationships are then assigned to each browsing class using the `$field_assignments[]` and `$relationship_assignments[]` arrays.

```
// Namespaces used in the configuration.
$namespaces = array(
  'ex' => 'http://example.org/example.owl#',
  'dc' => 'http://purl.org/dc/terms/',
);

// Fields and relationships.
$fields[] = array(
  'property' => 'dc:title',      // OWL datatype property.
  'label'     => 'Title',        // Human readable label.
  'type'      => 'text',         // Type of field (text or
                                relationship).
);

$fields[] = array(
  'property' => 'dc:description',
  'label'     => 'Description',
  'type'      => 'text',
);

$fields[] = array(
  'property' => 'ex:nick',
  'label'     => 'Nick name',
  'type'      => 'text',
);
```

```

$fields[] = array(
  'property' => 'ex:dateOfBirth',
  'label'     => 'DOB',
  'type'      => 'text',
);
$fields[] = array(
  'property' => 'ex:knows',
  'label'     => 'Knows',
  'type'      => 'relationship',
);
$fields[] = array(
  'property' => 'ex:worksAt',
  'label'     => 'Works At',
  'type'      => 'relationship',
);
$fields[] = array(
  'property' => 'ex:employs',
  'label'     => 'Employs',
  'type'      => 'relationship',
);

// Browsing classes.
$browsing_classes[] = array( // Person browsing class.
  // Name to identify this browsing class.
  'name'  => 'person',
  // Human readable label used in Drupal.
  'label' => 'People',
  // Corresponding class in the ontology (base class).
  'class' => 'ex:Person',
);
$browsing_classes[] = array(
  'name'  => 'business',
  'label' => 'Businesses',
  'class' => 'ex:Business',
);

// Assignment of fields to browsing classes.
$field_assignments['person'] = array(
  'dc:title',
  'dc:description',
  'ex:nick',
);

```

```

$field_assignments['business'] = array(
    'dc:title',
    'dc:description',
);

// Assignment of relationships to browsing classes.
$relationship_assignments['person'][] = array(
    'property' => 'ex:knows',
    'target'    => 'ex:Person',
);
$relationship_assignments['person'][] = array(
    'property' => 'ex:worksAt',
    'target'    => 'ex:Business',
);
$relationship_assignments['business'][] = array(
    'property' => 'ex:employs',
    'target'    => 'ex:Person',
);

```

Bibliography

- Adida, B., an Manu Sporny, I. H., and Birbeck, M. (2012a). RDFa 1.1 primer. W3C Recommendation, <http://www.w3.org/TR/xhtml-rdfa-primer/>.
- Adida, B., Birbeck, M., McCarron, S., and Pemberton, S. (2012b). HTML+RDFa 1.1 – support for RDFa in HTML4 and HTML5. W3C Working Draft 11 September 2012, <http://www.w3.org/TR/rdfa-in-html/>.
- Adrian, B., Hees, J., Herman, I., Sintek, M., and Dengel, A. (2010). Epiphany: Adaptable RDFa Generation Linking the Web of Documents to the Web of Data. In Cimiano, P. and Pinto, H. S., editors, *EKAU*, volume 6317 of *Lecture Notes in Computer Science*, pages 178–192. Springer.
- Altheim, M. and McCarron, S. (2001). XHTMLTM 1.1 - module-based XHTML. W3C recommendation, W3C. <http://www.w3.org/TR/2001/REC-xhtml11-20010531>.
- Amann, B. and Fundulaki, I. (1999). Integrating Ontologies and Thesauri to Build RDF Schemas. In *ECDL*, Lecture Notes in Computer Science, pages 234–253. Springer.
- Angeletou, S., Sabou, M., and Motta, E. (2009). Improving Folksonomies using Formal Knowledge: A Case Study on Search. In *the Proc. of the 4th Asian Semantic Web Conference (ASWC2009)*.
- Ankolekar, A., Krötzsch, M., Tran, T., and Vrandečić, D. (2007). The two cultures: mashing up web 2.0 and the semantic web. In *Proc. of the 16th international World Wide Web conference (WWW2007)*, pages 825–834.
- Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., and Simperl, E. P. B., editors (2009). *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, volume 5554 of *Lecture Notes in Computer Science*. Springer.
- Artale, A., Calvanese, D., Kontchakov, R., and Zakharyashev, M. (2009). The DL-Lite Family and Relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69.
- Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. G. (2007). DBpedia: A Nucleus for a Web of Open Data. In Aberer, K., Choi, K.-S., Noy, N. F., Allemang, D., Lee, K.-I., Nixon, L. J. B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer*

- Science*, pages 722–735. Springer.
- Baader, F. (2003). Terminological Cycles in a Description Logic with Existential Restrictions. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 325–330, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the EL Envelope. In Kaelbling, L. P. and Saffiotti, A., editors, *IJCAI*, pages 364–369. Professional Book Center.
- Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.
- Baader, F., Horrocks, I., and Sattler, U. (2002). Description Logics for the Semantic Web. *KI – Künstliche Intelligenz*, 16(4):57–59.
- Battle, R. and Benson, E. (2008). Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *J. Web Sem.*, 6(1):61–69.
- Bechhofer, S., v. Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and eds., L. A. S. (2004). OWL Web Ontology Language Reference. W3C Recommendation, <http://www.w3.org/TR/owl-ref/>.
- Beel, D., Webster, G., Taylor, S., Jekjantuk, N., Mellish, C., and Wallace, C. (2013). CURIOS: Connecting Community Heritage through Linked Data. Presented at Digital Economy 2013: Open Digital (DE2013).
- Benjamins, R. V., Davies, J., Baeza-Yates, R., Mika, P., Zaragoza, H., Greaves, M., Gomez-Perez, J. M., Contreras, J., Domingue, J., and Fensel, D. (2008). Near-Term Prospects for Semantic Technologies. *IEEE Intelligent Systems*, 23(1):76–88.
- Berners-Lee, T. (1997). Realising the Full Potential of the Web. W3C Document, <http://www.w3.org/1998/02/Potential.html>.
- Berners-Lee, T. (1998). Semantic Web Road Map. W3C Design Issues. <http://www.w3.org/DesignIssues/Semantic.html>.
- Berners-Lee, T. (2009). Linked-data design issues. W3C design issue document. <http://www.w3.org/DesignIssue/LinkedData.html>.
- Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34–43.
- Bing (2011). Introducing Schema.org: Bing, Google and Yahoo Unite to Build the Web of Objects. <http://www.bing.com/community/site.blogs/b/search/archive/2011/06/02/bing-google-and-yahoo-unite-to-build-the-web-of-objects.aspx>. Accessed: 2012-10-19.
- Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):33–42.
- Bishop, C. M. and Nasrabadi, N. M. (2007). *Pattern Recognition and Machine Learning*. *J. Electronic Imaging*, 16(4):049901.

- Bizer, C., Heath, T., and Berners-Lee, T. (2009a). Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22.
- Bizer, C., Jentzsch, A., and Cyganiak, R. (2011). State of the LOD Cloud. <http://www4.wiwi.fu-berlin.de/locloud/state/>. Accessed: 2012-10-16.
- Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009b). DBpedia - A crystallization point for the Web of Data. *J. Web Sem.*, 7(3):154–165.
- Bos, B., Lie, H. W., Lilley, C., and Jacobs, I. (2008). Cascading style sheets, level 2. W3C recommendation, W3C. <http://www.w3.org/TR/2008/REC-CSS2-20080411/>.
- Brickley, D. and Guha, R. (2000). Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommendation, <http://www.w3.org/TR/rdf-schema>.
- Brickley, D. and Miller, L. (2010). FOAF Vocabulary Specification 0.97. Namespace document: <http://xmlns.com/foaf/spec/20100101.html>.
- Brin, S. and Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117.
- Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., , and Rosati, R. (2005). DL-Lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*.
- Calvanese, D., Giacomo, G. D., Lenzerini, M., Rosati, R., and Vetere, G. (2004). DL-Lite: Practical Reasoning for Rich DLs. In *Proc. of the DL2004 Workshop*.
- Clark, K. G., Feigenbaum, L., and Torres, E. (2008). SPARQL protocol for RDF. W3C Recommendation, <http://www.w3.org/TR/rdf-sparql-protocol/>.
- Clark, L. (2010). SPARQL Views: A Visual SPARQL Query Builder for Drupal. In Polleres, A. and Chen, H., editors, *ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- Corlosquet, S., Delbru, R., Clark, T., Polleres, A., and Decker, S. (2009). Produce and Consume Linked Data with Drupal! In Bernstein, A., Karger, D. R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., and Thirunarayan, K., editors, *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 763–778. Springer.
- Cyganiak, R. (2011). The Linking Open Data cloud diagram. <http://richard.cyganiak.de/2007/10/loclod/>. Accessed: 2012-10-16.
- DCMI (2003). Dublin Core Metadata Element Set, Version 1.1: Reference Description. DCMI Recommendation <http://dublincore.org/documents/dces/>.
- de Bruijn, J. (2010). RIF RDF and OWL compatibility. W3C Recommendation, <http://www.w3.org/TR/rif-rdf-owl/>.
- Dixon, S. and Jacobson, K. (2011). LinkedBrainz - A project to provide MusicBrainz NGS as Linked Data. <http://linkedbrainz.c4dmpresents.org/>. Accessed: 2013-12-10.
- Dowling, W. F. and Gallier, J. H. (1984). Linear-Time Algorithms for Testing the Satisfiability

- of Propositional Horn Formulae. *J. Log. Program.*, 1(3):267–284.
- drupal.org (2001). About Drupal. <http://drupal.org/about>. Accessed: 2012-10-12.
- drupal.org (2005). Views – Drupal module. <http://drupal.org/project/views>. Accessed: 2012-10-12.
- drupal.org (2007). Resource Description Framework (RDF) – Drupal module. <http://drupal.org/project/rdf>. Accessed: 2012-10-12.
- drupal.org (2011). Schema.org – Drupal module. <http://drupal.org/project/schemaorg>. Accessed: 2012-10-19.
- Echarte, F., Astrain, J. J., Córdoba, A., and Villadangos, J. E. (2007). Ontology of Folksonomy: A New Modelling Method. In Handschuh, S., Collier, N., Groza, T., Dieng, R., Sintek, M., and de Waard, A., editors, *SAAKM*, volume 289 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database and some of its Applications*. MIT Press, Cambridge, MA.
- Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine.
- Fishkin, R. (2011). SEOMoz: search ranking factors. <http://www.seomoz.org/article/search-ranking-factors>. Accessed: 2012-10-27.
- Gangemi, A. (2005). Ontology Design Patterns for Semantic Web Content. In Gil, Y., Motta, E., Benjamins, V. R., and Musen, M. A., editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 262–276. Springer.
- Garshol, L. M. (2004). Metadata? Thesauri? Taxonomies? Topic Maps! Making Sense of it all. *Journal of Information Science*, 30(4):378–391.
- Golbreich, C. and Wallace, E. K. (2010). OWL 2 Web Ontology Language: New Features and Rationale. W3C Recommendation, <http://www.w3.org/TR/owl2-new-features/>.
- Greaves, M. (2007). Semantic Web 2.0. *IEEE Intelligent Systems*, 22(2):94–96.
- Grosof, B., Volz, R., Horrocks, I., and Decker, S. (2003a). Description Logic Programs: Combining Logic Programs with Description Logics. In *Proc. of WWW 2003*, pages 48–57.
- Grosof, B. N., Horrocks, I., Volz, R., and Decker, S. (2003b). Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57.
- Gruber, T. (2007). Ontology of Folksonomy: A Mash-up of Apples and Oranges. *Int’l Journal on Semantic Web & Information Systems*, 3(2).
- Guha, R. (2011). Introducing schema.org: Search engines come together for a richer web. <http://googleblog.blogspot.co.uk/2011/06/introducing-schemaorg-search-engines.html>. Accessed: 2012-10-19.
- Guo, Y., Pan, Z., and Heflin, J. (2005). LUBM: A Benchmark for OWL Knowledge Base

- Systems. *Journal of Web Semantics*, 3(2):158–182.
- Halpin, H. and Davis, I. (2007). GRDDL Primer. W3C working group note, W3C. <http://www.w3.org/TR/grddl-primer/>.
- Halpin, H., Robu, V., and Shepherd, H. (2007). The complex dynamics of collaborative tagging. In *Proc. of the 16th International World Wide Web Conference (WWW2007)*, pages 211–220.
- Hayes, P. (2004). RDF Semantics. Technical report, W3C. W3C Recommendation, <http://www.w3.org/TR/rdf-mt/>.
- Hickson, I. (2012). HTML Microdata. W3C Working Draft 29 March 2012, <http://www.w3.org/TR/2012/WD-microdata-20120329/>.
- Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S., editors (2009). *OWL 2 Web Ontology Language: Primer*. W3C Recommendation. Available at <http://www.w3.org/TR/owl2-primer/>.
- Hitzler, P. and Vrandečić, D. (2005). Resolution-Based Approximate Reasoning for OWL DL. In *Proc. of the 4th International Semantic Web Conference (ISWC2005)*.
- Horrocks, I., Kutz, O., and Sattler, U. (2006). The Even More Irresistible SROIQ. In *KR*, pages 57–67.
- Horrocks, I., Sattler, U., and Tobies, S. (2000). Practical Reasoning for Very Expressive Description Logics. In *Logic Journal of the IGPL*, pages 239–263.
- Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. (2006). Information Retrieval in Folksonomies: Search and Ranking. In Sure, Y. and Domingue, J., editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 411–426. Springer.
- Hyatt, D. and Hickson, I. (2012). HTML 5. W3C candidate recommendation, W3C. <http://www.w3.org/TR/2012/CR-html5-20121217/>.
- International, E. C. M. A. (1999). *ECMA-262: ECMAScript Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, third edition.
- Jackson, C. and Wang, H. J. (2007). Subspace: secure cross-domain communication for web mashups. In *Proc. of 16th International World Wide Web Conference (WWW2007)*, pages 611–620.
- Järvelin, K. and Kekäläinen, J. (2000). IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48.
- Jena, A. (2013). Reasoners and rule engines: Jena inference support. <http://jena.apache.org/documentation/inference/>. Accessed: 2013-05-27.
- Jones, K. S. (1991). Notes and References on Early Automatic Classification Work. *SIGIR Forum*, 25(1):10–17.
- joomla.org (2005). What is Joomla? <http://www.joomla.org/about-joomla.html>. Accessed:

- 2012-10-12.
- JSR-000221 (2011). JSR-000221 JDBC API Specification 4.1.
- Kekäläinen, J. and Järvelin, K. (1998). The Impact of Query Structure and Query Expansion on Retrieval Performance. In *SIGIR*, pages 130–137. ACM.
- Kifer, M. and Boley, H. (2010). RIF Overview. W3C Recommendation, <http://www.w3.org/TR/rif-overview/>.
- Klir, G. J. and Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice-Hall.
- Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. <http://www.w3.org/TR/rdf-concepts/>. Series Editor: Brian McBride.
- Knublauch, H., Fergerson, R. W., Noy, N. F., and Musen, M. A. (2004). The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In McIlraith, S. A., Plexousakis, D., and van Harmelen, F., editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer.
- Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., and Lee, R. (2009). Media Meets Semantic Web - How the BBC Uses DBpedia and Linked Data to Make Connections. In Aroyo et al. (2009), pages 723–737.
- Korfhage, R., Rasmussen, E. M., and Willett, P., editors (1993). *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*. ACM.
- Krötzsch, M., Rudolph, S., and Hitzler, P. (2008). Description Logic Rules. In Ghallab, M., Spyropoulos, C. D., Fakotakis, N., and Avouris, N., editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications (FAIA)*, pages 80–84. IOS Press.
- Krötzsch, M., Vrandečić, D., and Völkel, M. (2006). Semantic MediaWiki. In Cruz, I. F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 935–942. Springer.
- Lambrix, P., Dragisic, Z., and Ivanova, V. (2012). Get My Pizza Right: Repairing Missing is-a Relations in *ALC* Ontologies. In Takeda, H., Qu, Y., Mizoguchi, R., and Kitamura, Y., editors, *JIST*, volume 7774 of *Lecture Notes in Computer Science*, pages 17–32. Springer.
- Lee, M.-C., Tsai, K. H., and Wang, T. I. (2008). A practical ontology query expansion algorithm for semantic-aware learning objects retrieval. *Computers & Education*, 50(4):1240–1257.
- Lin, H., Davis, J., and Zhou, Y. (2009). An Integrated Approach to Extracting Ontological Structures from Folksonomies. In Aroyo et al. (2009), pages 654–668.
- Manaf, N. A. A., Bechhofer, S., and Stevens, R. (2010). A Survey of Identifiers and Labels in OWL Ontologies. In Sirin, E. and Clark, K., editors, *OWLED*, volume 614 of *CEUR*

- Workshop Proceedings*. CEUR-WS.org.
- Manola, F. and Miller, E. (2004). RDF Primer. W3C Recommendation, <http://www.w3.org/TR/rdf-primer/>.
- mediawiki.org (2002). What is MediaWiki? http://www.mediawiki.org/wiki/How_does_MediaWiki_work%3F. Accessed: 2012-10-12.
- Mika, P. (2005). Ontologies are us: A unified model of social networks and semantics. In *4th International Semantic Web Conference (ISWC 2005)*.
- Miles, A., Matthews, B., Wilson, M. D., and Brickley, D. (2005). SKOS Core: Simple Knowledge Organisation for the Web. In *Proc. International Conference on Dublin Core and Metadata Applications, Madrid, Spain*.
- Miller, G. (1995). WordNet: A Lexical Database for English. *Comm. ACM*, 38(11).
- Miller, G. A., Chodorow, M., Landes, S., Leacock, C., and Thomas, R. G. (1994). Using a Semantic Concordance for Sense Identification. In *HLT*. Morgan Kaufmann.
- Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. (2009a). OWL 2 Web Ontology Language – Profiles. Technical report, W3C.
- Motik, B., Patel-Schneider, P. F., and Grau, B. C. (2009b). OWL 2 Web Ontology Language: Direct Semantics. W3C Recommendation, <http://www.w3.org/TR/owl2-direct-semantics/>.
- Motik, B., Patel-Schneider, P. F., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., and Smith, M. (2009c). OWL 2 web ontology language: Structural specification and functional-style syntax. W3C Recommendation, <http://www.w3.org/2007/OWL/draft/owl2-syntax/>.
- Motik, B., Shearer, R., and Horrocks, I. (2009d). Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228.
- Navigli, R. and Velardi, P. (2003). An analysis of ontology-based query expansion strategies. In *Workshop on Adaptive Text Extraction and Mining, (Cavtat Dubrovnik, Croatia, Sept 23)*.
- OWL Working Group, W. (2009). OWL 2 Web Ontology Language: Document Overview. W3C Recommendation. Available at <http://www.w3.org/TR/owl2-overview/>.
- Pan, J., Taylor, S., and Thomas, E. (2009a). MusicMash2: Mashing Linked Music Data via An OWL DL Web Ontology. In *the Proc. of the Web Science Conference 2009 (WebSci09)*.
- Pan, J., Taylor, S., and Thomas, E. (2009b). Reducing Ambiguity in Tagging Systems with Folksonomy Search Expansion. In *the Proc. of the 6th European Semantic Web Conference (ESWC2009)*.
- Pan, J. Z., Stamou, G., Stoilos, G., Taylor, S., and Thomas, E. (2008). Scalable Querying Services over Fuzzy Ontologies. In *the Proc. of the 17th International World Wide Web Conference (WWW2008)*.
- Pan, J. Z., Taylor, S., and Thomas, E. (2007). MusicMash2: Expanding Folksonomy Search

- with Ontologies. In *the Proc. of SAMT2007 Workshop on Multimedia Annotation and Retrieval enabled by Shared Ontologies (MARESO)*.
- Pan, J. Z. and Thomas, E. (2007). Approximating OWL-DL Ontologies. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, pages 1434–1439.
- Pan, J. Z., Thomas, E., Ren, Y., and Taylor, S. (2012). Exploiting Tractable Fuzzy and Crisp Reasoning in Ontology Applications. In *IEEE Computational Intelligence Magazine*.
- Pan, J. Z., Thomas, E., and Sleeman, D. (2006). ONTOSEARCH2: Searching and Querying Web Ontologies. In *Proc. of WWW/Internet 2006*, pages 211–218.
- Pan, J. Z., Thomas, E., and Zhao, Y. (2009c). Completeness Guaranteed Approximations for OWL-DL Query Answering. In *International Workshop on Description Logics (DL2009)*.
- Parsia, B., Sirin, E., and Kalyanpur, A. (2005). Debugging OWL ontologies. In Ellis, A. and Hagino, T., editors, *WWW*, pages 633–640. ACM.
- Passant, A. (2007). Using Ontologies to Strengthen Folksonomies and Enrich Information Retrieval in Weblogs. In *Proc. of 2007 International Conference on Weblogs and Social Media (ICWSM2007)*.
- Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, W3C. W3C Recommendation, <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>.
- Prud’hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C Recommendation, <http://www.w3.org/TR/rdf-sparql-query/>.
- Qiu, Y. and Frei, H.-P. (1993). Concept Based Query Expansion. In Korfhage et al. (1993), pages 160–169.
- Quinn, A. J. and Bederson, B. B. (2011). Human computation: a survey and taxonomy of a growing field. In Tan, D. S., Amershi, S., Begole, B., Kellogg, W. A., and Tungare, M., editors, *CHI*, pages 1403–1412. ACM.
- Raimond, Y., Abdallah, S. A., Sandler, M. B., and Giasson, F. (2007). The Music Ontology. In Dixon, S., Bainbridge, D., and Typke, R., editors, *ISMIR*, pages 417–422. Austrian Computer Society.
- Raimond, Y. and Sandler, M. B. (2008). A Web of Musical Information. In Bello, J. P., Chew, E., and Turnbull, D., editors, *ISMIR*, pages 263–268.
- Rayfield, J. (2012). Dynamic Semantic Publishing. In Maass, W. and Kowatsch, T., editors, *Semantic Technologies in Content Management Systems*, pages 49–64. Springer.
- Rector, A. and Rogers, J. (2006). Ontological and Practical Issues in Using a Description Logic to Represent Medical Concept Systems: Experience from GALEN. In *IN REASONING WEB, SECOND INTERNATIONAL SUMMER SCHOOL, TUTORIAL LECTURES*, pages 197–231.
- Ren, Y., Pan, J. Z., and Zhao, Y. (2010). Soundness Preserving Approximation for TBox Reasoning. In *the Proc. of the 25th AAAI Conference Conference (AAAI2010)*.

- Reynolds, D. (2010). OWL 2 RL in RIF. W3C Working Group Note, <http://www.w3.org/TR/2010/NOTE-rif-owl-rl-20100622/>.
- Rockley, A., Kostur, P., and Manning, S. (2002). *Managing Enterprise Content: A Unified Content Strategy*. Pearson Education.
- Roussey, C., Pinet, F., Kang, M.-A., and Corcho, O. (2011). an Introduction to Ontologies and Ontology Engineering. *Advanced Information and Knowledge Processing*, 1:9–38. Ontologies in Urban Development Projects.
- Ruotsalo, T. (2012). Domain Specific Data Retrieval on the Semantic Web. In Simperl, E., Cimiano, P., Polleres, A., Corcho, Ó., and Presutti, V., editors, *ESWC*, volume 7295 of *Lecture Notes in Computer Science*, pages 422–436. Springer.
- Salton, G. and McGill, M. J. (1983). *Introduction to modern information retrieval*. McGraw-Hill.
- Salton, G., Wong, A., and Yang, C. S. (1975). A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620.
- schema.rdfs.org (2011). About: schema.rdfs.org. <http://schema.rdfs.org/about.html>. Accessed: 2012-10-19.
- Seaborne, A. (2011). Fuseki: serving RDF data over HTTP. http://jena.apache.org/documentation/serving_data/. Accessed: 2012-10-27.
- Selman, B. and Kautz, H. (1996). Knowledge Compilation and Theory Approximation. *J. ACM*, 43(2):193–224.
- Shadbolt, N., O’Hara, K., Berners-Lee, T., Gibbins, N., Glaser, H., Hall, W., and m. c. schraefel (2012). Linked open government data: Lessons from data.gov.uk. *IEEE Intelligent Systems*, 27(3):16–24.
- Silverstein, C., Henzinger, M. R., Marais, H., and Moricz, M. (1999). Analysis of a Very Large Web Search Engine Query Log. *SIGIR Forum*, 33(1):6–12.
- softpedia.com (2011). Flickr Boasts 6 Billion Photo Uploads. <http://news.softpedia.com/news/Flickr-Boasts-6-Billion-Photo-Uploads-215380.shtml>. Accessed: 2012-10-19.
- Sowa, J. F. (1984). Conceptual graphs. In *Information Processing in Mind and Machine*, pages 39–44. Addison-Wesley.
- Sowa, J. F. (1987). Semantic Networks. In *Encyclopedia of Artificial Intelligence 2*. John Wiley & Sons.
- SPARQL (2012). SPARQL 1.1 overview. W3C Working Draft, <http://www.w3.org/TR/sparql11-overview/>.
- SQL:2011 (2011). Information technology – Database languages – SQL.
- Steiner, J. (2010). Information Management With Oracle Database 11g Release 2 – White Paper.

- Stevenson, M. (2002). Combining disambiguation techniques to enrich an ontology. In *Proceedings of the Machine Learning and Natural Language Processing for Ontology Engineering Workshop*.
- Straccia, U. (2005). Towards a fuzzy description logic for the semantic web. In *Proceedings of the 2nd European Semantic Web Conference*.
- Straccia, U. (2006). Answering Vague Queries in Fuzzy DL-Lite. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, pages 2238–2245.
- Tait, E., MacLeod, M., Beel, D., Wallace, C., Mellish, C., and Taylor, S. (2013). Linking to the past: An Analysis of Community Digital Heritage Initiatives. *Aslib Proceedings, Vol. 65 Iss: 6, pp.564 - 580*.
- Taylor, S., Jekjantuk, N., Mellish, C., and Pan, J. Z. (2013). Reasoning Driven Configuration of Linked Data Content Management Systems. In *the Proc. of the 3rd Joint International Semantic Technology Conference (JIST2013)*.
- Taylor, S., Pan, J. Z., and Thomas, E. (2008). MusicMash2: A Web 2.0 Application Built in a Semantic Way. In *the Proc. of the 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE2008)*.
- ter Horst, H. J. (2005). Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *J. Web Sem.*, 3(2-3):79–115.
- Thomas, E., Pan, J. Z., and Ren, Y. (2010a). TrOWL: Tractable OWL 2 Reasoning Infrastructure. In *the Proc. of the Extended Semantic Web Conference (ESWC2010)*.
- Thomas, E., Pan, J. Z., and Sleeman, D. (2007). ONTOSEARCH2: Searching Ontologies Semantically. In *Proc. of OWL: Experiences and Directions (OWL-ED2007)*.
- Thomas, E., Pan, J. Z., Taylor, S., Jekjantuk, N., and Ren, Y. (2009). Semantic Advertising for Web 3.0. In *the Proc. of the 2nd Future Internet Symposium (FIS2009)*.
- Thomas, E., Pan, J. Z., Taylor, S., and Ren, Y. (2010b). Lightweight Reasoning and the Web of Data for Web Science. In *the Proc. of the 2nd International Conference on Web Science (WebSci 2010)*.
- Towell, G. G. and Voorhees, E. M. (1998). Disambiguating Highly Ambiguous Words. *Computational Linguistics*, 24(1):125–145.
- Vechtomova, O. (2009). Query Expansion for Information Retrieval. In Liu, L. and Özsu, M. T., editors, *Encyclopedia of Database Systems*, pages 2254–2257. Springer US.
- Voorhees, E. M. (1993). Using WordNet to Disambiguate Word Senses for Text Retrieval. In Korfhage et al. (1993), pages 171–180.
- Voorhees, E. M. (1998). Using WordNet for Text Retrieval. In Fellbaum, C., editor, *WordNet: An Electronic Lexical Database*, pages 285–303. MIT Press, Cambridge, MA.

- wordpress.com (2012). Stats – WordPress.com. <http://en.wordpress.com/stats/>. Accessed: 2012-10-19.
- wordpress.org (2003). About Wordpress. <http://wordpress.org/about/>. Accessed: 2012-10-12.
- Yahoo! (2011). Introducing schema.org: A Collaboration on Structured Data. <http://www.ysearchblog.com/2011/06/02/introducing-schema-org-a-collaboration-on-structured-data/>. Accessed: 2012-10-19.