



Master Degree in Computer Science

Bootstrapping the Web of Data with Drupal

by

Stéphane Jean Joseph Corlosquet

This thesis is submitted to the National University of Ireland, Galway, in fulfillment of the award of M.Sc. in Computer Science.

Supervisor: Dr. Axel Polleres
Digital Enterprise Research Institute
National University of Ireland, Galway

Director: Prof. Dr. Stefan Decker
Digital Enterprise Research Institute
National University of Ireland, Galway

Galway, July 31th, 2009

Contents

1	Introduction	2
2	Preliminaries	7
2.1	History and Pre-Existing Work on Content Management Systems . . .	7
2.2	Semantic Web Technologies	9
2.2.1	RDF	9
2.2.2	Ontologies and Vocabularies	12
2.2.3	Querying RDF with SPARQL	16
2.3	RDF serialization formats	19
2.3.1	RDF/XML	19
2.3.2	RDFa	19
2.3.3	Notation3	20
2.3.4	Turtle	21
2.3.5	N-Triples	21
2.4	Vocabulary publishing on the Semantic Web	21
2.4.1	The value of vocabularies	22
2.4.2	Current approaches to vocabulary publishing	23
2.5	Linked Data	24
2.6	Content Management Systems and RDF support	25
2.6.1	Content Management Systems	25
2.6.2	CMSs and RDF support	27
2.6.3	Details on Drupal	27
3	Solutions to connect Drupal to the Web of Data	34
3.1	From Content Models to Site Vocabularies	36
3.1.1	Building a Site Vocabulary	36
3.1.2	Adhering to Linked Data principles	38
3.2	Mapping Content Models to Existing Ontologies	38
3.2.1	External vocabulary importer module	39
3.2.2	External ontology search service	39
3.2.3	Mapping process	40
3.2.4	User experience	42
3.3	Exposing and Consuming Linked Data with SPARQL	43
3.3.1	Exposing RDF data with a SPARQL endpoint	44

3.3.2	Consuming Linked Data by lazy loading of remote RDF resources	45
4	User Evaluation and Adoption of the Implemented Solutions	46
4.1	Usability	46
4.2	Adoption	47
5	Minimal RDF Support for Drupal Core	49
5.1	RDF Schema proposal	49
5.2	Implementation for Drupal Core 7	51
5.2.1	RDF Mapping Definition	51
5.2.2	RDFa output	52
6	Neologism: Easy RDFS vocabulary publishing	54
6.1	Architecture	54
6.2	User experience	55
6.3	Adoption	56
7	Conclusions and Outlook	58
A	Namespaces	68
B	Drupal RDF Content Construction Kit evaluation	69
B.1	Introduction	69
B.2	Content Construction Kit	69
B.3	Setting up the content model	70
B.3.1	Content types	70
B.3.2	Fields	71
B.4	Map the Content Model to RDF terms	72
B.5	Drupal RDF Content Construction Kit evaluation - Questionnaire . . .	73
B.6	Drupal RDF Content Construction Kit evaluation - RDF Mappings . .	74

List of Figures

2.1	Basic RDF statements.	11
2.2	Basic RDF graph.	11
2.3	RDF Graph merge	12
2.4	Blank node graph	12
2.5	RDFS graph representing a simple hierarchy of classes.	14
2.6	Graph representing RDFS entailment rules.	15
2.7	Notation 3 classification (from [12]).	20
2.8	The Linking Open Data cloud (2009).	24
2.9	Web Content Management Systems comparison (from [60]).	27
2.10	Hierarchy of contributors on a typical Drupal site	30
2.11	User profile page built with Drupal's CCK.	31
2.12	Administration page of the Person content type in Drupal's CCK.	32
2.13	List of fields of the Person content type in Drupal's CCK.	32
2.14	Defining constraints on the gender field in Drupal's CCK.	33
3.1	Graph of a set of restrictions on a class.	37
3.2	RDF mappings management through the Drupal interface: RDF class mapping (left) and RDF property mapping (right).	43
3.3	Form for importing an external vocabulary (left) and confirmation message after a vocabulary import (right).	43
3.4	Extended example in a typical Linked Data eco-system.	44
3.5	A list of SPARQL results (left) and an RDF SPARQL Proxy profile form (right).	45
4.1	Comparison between initial setup and RDF mappings.	47
4.2	Evolution of the number of installations of RDF CCK since its release.	48
5.1	Drupal RDF Schema proposal.	50
6.1	A vocabulary page in Neologism, as it appears to an authenticated user.	55
6.2	A form for editing a class.	56
6.3	The vocabulary overview diagram.	57

Abstract

Content Management Systems (CMS), blogging tools, and other Internet applications such as wikis are the basis for the explosion of web content available on the Internet today. Each of these system implements its own structure for processing and storing the information depending on the specific requirements and platform it was built for, thus locking the information in a data silo. One of the main problems with so much disparate data appears when it comes to sharing, reusing, merging or exporting this data across various platforms or applications. The lack of unified semantics for describing this data make these processes rather cumbersome and operose. The Resource Description Framework (RDF) provides a solution to solve this problem with a unified, standardized language to describe resources on the Web in a machine-readable format, therefore enabling the smooth exchange of information and knowledge on the Semantic Web.

This thesis describes a set of tools which expose the huge amounts of Web content residing in CMS systems to the Web of Data. We implement our approach in one of the most popular CMS systems nowadays, Drupal, where we enable site administrators to export their site content to the Web of Data without requiring extensive knowledge on Semantic Web technologies. This becomes possible by mapping the inherent site structure to a lightweight ontology that we call the Site Vocabulary, and exposing site data directly in RDF. Of itself, this simple solution would not link to existing Semantic Web data, since site vocabularies exist decoupled from the widely used vocabularies in the Linked Data cloud. To close this gap, we suggest an easy-to-use and fail-safe ontology import and reuse mechanism allowing site administrators to link their site vocabulary, and thus their site data, to existing widely used vocabularies on the Web. Moreover, we make all this data available in a SPARQL endpoint, allowing other parties on the Internet to execute complex queries against the content of a site. Reversely, taking advantage of the existing Linked Data on the Web, we allow site administrators to include data from remote SPARQL endpoints on their site. Finally, to cater to those willing to define their own vocabularies, we propose a tool for authoring and publishing RDF vocabularies online.

We believe that our approach will help to bootstrap the Web of Data by leveraging the huge amounts of information contained in CMSs. In approaching site administrators rather than end users, we tackle the problem of adding semantics where we consider it easiest: Semantics are fixed at design time based on the site structure and, as a result, users entering data produce Linked Data automatically without extra burden.

Acknowledgements

First of all, I want to give my sincere gratitude to Dr. Axel Polleres who supervised me and provided me with his wise guidance during the course of my Masters at DERI Galway. I would like to thank my colleagues Nuno Lopes, Thomas Krennwallner, Dr. Antoine Zimmerman and Dr. Gergely Lukácsy for their expertise and entertaining conversations on Description Logics, OWL, emacs and Latex. Thanks to Dr. Alexandre Passant, Dr. Michael Hausenblas, Dr. Bart Gruzalski and Nuno Lopes for their valuable feedback on my thesis. Thanks to VinhTuan Thai, Tudor Groza, Sheila Kinsella, Aidan Hogan, Knud Möller, Laura Dragan, Gabriela Vulcu, Wassim Derguech, Georgeta Bordea and Jürgen Umbrich who helped with the evaluation of some of the solutions presented in this thesis.

Let me thank, in no particular order, Cosmin Basca, Stefan Bischof, Dr. James Cooley, Renaud Delbru, Andrew Gallagher, Benjamin Heitmann, Dr. Andreas Harth, Richard Cyganiak, Dr. John Breslin, Deirdre Lee, Julie Letierce, Peyman Nasirifard, Alexander Schutz, Thomas Schandl and Sergio Fernández who all participated in one way or another in the success of my Master at DERI Galway. I want to also thank DERI at large, post-graduate students, post-doc researchers and related staff for their kind aid and advice.

A special thanks to Dr. Eyal Oren, Dr. John Breslin and Prof. Dr. Stefan Decker who gave me the opportunity to work at DERI and without whom I would have not have been able to experience this great journey.

I am grateful to Tim Clark who invited me to visit Harvard's Initiative in Innovative Computing, as well as Sudeshna Das for supervising me during my stay in Cambridge. I want to thank Marco Neumann for giving me the opportunity to speak at the New York City Semantic Web Meetup, as well as Lee Feigenbaum, Lalana Kagal and Oshani Seneviratne for arranging the various meetups at MIT. I would like to thank Sir Tim Berners-Lee and his group for their feedback on the solutions presented in this thesis. Moreover, I want to thank all the people who made my journey in the US a great one: Benjamin Melançon, Kathleen Murtagh and the Agaric Design team, Jonathan Hendler, Cameron Freer, Eric Prud'hommeaux, Greg Rundlett, Patty Kopperl and Morton Swimmer.

I would like to thank the Drupal developers who participated in the RDF code sprint, namely, Florian Loretan, Rolf Guescini, Benjamin Melançon, Stefan Freudenberg, Frédéric G. Marand, Mark Birbeck, John Morahan and Dr. John Breslin. Many thanks to the Drupal community at large from which I have learnt a great deal and who provided a solid, practical testbed for the tools described in this thesis.

Finally I would like to thank my wife Diliny De Alwis Corlosquet for her constant support and great patience when I had to work overnight, on week-ends or on holidays, and my family Hervé, Martine, Thérèse and Jérôme Corlosquet for their belief in me and aid despite the distance between us.

The work presented in this thesis has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and the European FP6 project in-Context (IST-034718).

Chapter 1

Introduction

Since the late 90ies and early 2000s a paradigm shift has taken place in Web publishing towards a separation of data (content) and structure (mainly layout). The first ideas to have the data represented in a way which allows its reuse in various ways and not only HTML, emerged in parallel in various systems such as Typo3 (1997), Plone (1999), WebML [21]. These open source systems and their commercial counterparts are nowadays typically subsumed under the common term *Content Management Systems* (CMS).

While it is worthwhile to mention that the first of these systems appeared at around the same time as Semantic Web technologies emerged, with Resource Description Framework RDF [41] being standardized in 1999, the development of CMS and Semantic Web technologies have gone largely separate paths. Semantic Web technologies have matured to the point where they are increasingly being deployed on the Web, hereby forming a Data Web or Web of Data. But the HTML Web still dwarfs this emerging Web of Data and – boosted by technologies such as CMS – is still growing at much faster pace than the Semantic Web.

This is a pity, since actually both worlds could significantly benefit from each other. Particularly, large amounts of RDF data can now be accessed over the Web as *Linked Data* and built into Web applications [31]. Linked Data is a method for exposing connected data on the Web via dereferenceable URIs. This data is used by a variety of clients and in RDF data mashups that integrate information from various sources, search engines that allow structured queries across multiple sites and datasets, and data browsers that present a site’s content in new ways. The task of “RDFizing” existing web sites, which contain structured information such as events, personal profiles, ratings, tags, and locations, is important and of high potential benefit. While the Web features some huge websites with millions of pages and users, a lot of the Web’s interest and richness is in the “long tail”, in smaller special-interest websites.

Despite the fact that the need for dynamic inclusion and export of structured data in websites is widely recognized readily available support for exporting and consuming Linked Data in CMS systems – especially for non-experts – is still lacking. At present, RSS feeds are the only agreed way to share and syndicate data across websites. However, the RSS format is very poor and limited when it comes to semantics. It was

designed to carry news information, but it is today being employed for carrying other types of information due to the fact that users are unaware of compelling alternatives such as Linked Data. We believe RDF is a good candidate for improving interoperability between sites for the following reasons: RSS can only carry a flat list of news item, while RDF can express any structured data and – by RDFS and OWL – even describe its own structure. Moreover, it offers a structured query language and protocol in order to extract and retrieve selected data that match a set of criteria, whereas with RSS, one first needs to fetch an entire feed and then process it locally to extract the needed information.

Hypothesis It is remarkable that although most common CMS system support RSS, RDF is still being largely ignored as a much richer potential syndication format; especially, since CMSs are typically built on a structured model of the domain of the site which is reflected in both the underlying database, but – also and often more accurately – in the content types defined in the CMS system itself by a site administrator. It is reasonable to assume that this structured models map naturally to RDFS and OWL. Additionally, the recently standardized RDFa [1] format could support the exposure of structured data on CMS by allowing RDF to be embedded directly into the exposed HTML pages, as opposed to a separate document (like in RSS). Hence RDFS, OWL and RDFa seem to be more adequate means to expose machine-readable Linked Data on Web sites. Likewise, the consumption and aggregation of Linked Data on a CMS site offer new possibilities further described in this thesis. Approaching site administrators of widely used CMSs with easy-to-use tools to enhance their site with Linked Data will not only be to their benefit, but also significantly boost the Web of Data.

Approach We present several modules for Drupal – a state-of-the art CMS which has been gaining popularity recently by its rapidly growing user community, openness and modular design – to enable Linked data exposure and consumption.

Contributions

This section lists the various contributions which were made in the course of this thesis.

Drupal Modules

Several modules have been contributed to the Drupal community and are listed below.

RDF CCK (<http://drupal.org/project/rdfcck>) is an extension for Drupal which allow to link back this content model to the Web of Data. After installing this module on an existing Drupal site, the content types and fields typically defined by a Drupal site administrator using Drupal's Content Construction Kit (CCK) will be automatically exposed in terms of classes and properties of a canonical OWL ontology, the *site vocabulary*. The site data itself can become

available as RDFa embedded in the live website following Linked Data principles. While this off-the-shelf solution already makes any Drupal site which installs our module amenable to Semantic Web tools such as RDF crawlers, search engines, and SPARQL engines to search and query the site, this is only a first step.

RDF external vocabulary importer (<http://drupal.org/project/evoc>)
For better linkage to the Web of data, we enable mappings of the site vocabulary to existing, widely used ontologies. To this end, this module extends CCK by means to import existing vocabularies and map the local terms to external ones. To keep the burden low for site administrators who normally have little interest in learning details of RDF and OWL, we support safe reuse, i.e. in linking content types and fields to existing classes and properties we avoid modifications of existing ontologies.

RDF SPARQL Endpoint (http://drupal.org/project/sparql_ep) Upon simple installation of this module, the site data can – on top of the RDFa data on the HTML pages exposed by the CMS – additionally be exposed with a standard SPARQL query interface, following the standard SPARQL protocol. This module is based on the PHP ARC2 library.¹

RDF SPARQL Proxy (<http://drupal.org/project/rdproxy>) We allow administrators to dynamically integrate data from other RDF enhanced Drupal sites or Linked Data producers by an RDF SPARQL Proxy module.

Neologism (<http://neologism.googlecode.com/>) Neologism is an online vocabulary editor and publishing system based on Drupal,² implemented in PHP and ActionScript, and reduces the time required to create, publish and modify RDF vocabularies.

Drupal Core A list of patches to integrate minimal RDF support in Drupal core 7 is available at http://drupal.org/project/issues/search/drupal?issue_tags=RDF.

Results Dissemination and Community Work

Part of the work described in this thesis involved some dissemination tasks to raise awareness about the Semantic Web technologies and make these solutions available to and used by as many people as possible.

Talks

- *Social Networking Techniques and Potential* presentation at DrupalCon Barcelona on September 21st, 2007 (<http://barcelona2007.drupalcon.org/node/438>). Report on the session available at <http://openspring.net/blog/2007/09/24/great-success-of-the-semantic-web-at-drupalcon>.

¹<http://arc.semsol.org/>

²<http://drupal.org/>

- *SIOC, the Semantic Web and Drupal* at the Drupal Austria Meetup on March 12th, 2008 in Vienna (<http://groups.drupal.org/node/9466>).
- *Drupal and the Semantic Web: the Neologism project* presentation at DrupalCon Szeged on August 30th, 2008 (<http://szeged2008.drupalcon.org/program/sessions/drupal-and-semantic-web-neologism-project>).
- New York City Semantic Web Meetup on February 26th, 2009 (<http://semweb.meetup.com/25/calendar/9630332/>). “Drupal Semantics” presentation the RDF CCK and Evoc modules. Video available at <http://www.vimeo.com/4427060>, post event interview at <http://www.vimeo.com/5024015> and a blog post from a member of the audience at <http://semanticalley.com/2009/02/27/144/>.
- New York City DrupalCamp on February 28th, 2009 (<http://groups.drupal.org/node/18467>). “Drupal Semantics” presentation the RDF CCK and Evoc modules.
- *RDF Bird of a Feather* gatherings at the DrupalCon Washington DC on March 5th/6th, 2009 (<http://dc2009.drupalcon.org/>). Wrap up including slides available at <http://groups.drupal.org/node/19927>, list of participants and use cases at <http://groups.drupal.org/node/19734>.
- Cambridge Semantic Web gathering at MIT Stata center on March 10th, 2009 (http://esw.w3.org/topic/CambridgeSemanticWebGatherings/Meeting/2009-03-10_Gathering). Presentation on Drupal, RDF CCK and Evoc.
- Decentralized Information Group meeting at MIT on March 26th, 2009. Presentation of Drupal, RDF CCK, Evoc and Neologism to Tim Berners-Lee and his group.
- *RDF in Drupal Core code sprint* (<http://groups.drupal.org/node/21469>) organized at DERI on June 11th-14th 2009 gathered nine Drupal developers to work on integrating a lightweight RDF API in Drupal core.
- Presentation on RDF in Drupal at the *IKS Requirements Workshop* (<http://www.iks-project.eu/requirements-workshop>) in Salzburg, Austria on May 28-29th, 2009, interview at <http://www.vimeo.com/5214999>.

Online discussions and advocacy

- Manager of the Semantic Web group on Drupal.org: <http://groups.drupal.org/semantic-web> created in September 2007.
- *RDFa in Drupal - examples and use cases* screencast (<http://groups.drupal.org/node/20167> which was shown at the DrupalCon DC 2009 on March 10th.

- *Drupal Voices 18: Stéphane Corlosquet on the Semantic Web & Drupal* interview with Lullabot available at <http://www.lullabot.com/drupal-voices/drupal-voices-18-st%C3%A9phane-corlosquet-semantic-web-drupal>.
- *Drupal RDF Schema proposal* discussion available at <http://groups.drupal.org/node/9311>.
- *A roadmap for RDFa in Drupal 7* discussion available at <http://groups.drupal.org/node/16597>.

Publications

Workshop papers

- Cosmin Basca, Stéphane Corlosquet, Richard Cyganiak, Sergio Fernández and Thomas Schandl, Neologism: Easy Vocabulary Publishing. In Proceedings of the Workshop on Scripting for the Semantic Web Workshop at ESWC2008, Tenerife, Spain, 2008. Available at <http://www.semanticscripting.org/SFSW2008/papers/10.pdf>
- Stéphane Corlosquet, Richard Cyganiak, Axel Polleres and Stefan Decker, RDFa in Drupal: Bringing Cheese to the Web of Data. In 5th Workshop on Scripting and Development for the Semantic Web at ESWC2009, Crete, Greece. Available at http://www.semanticscripting.org/SFSW2009/short_3.pdf

Technical Reports

- Stéphane Corlosquet, Richard Cyganiak, Stefan Decker, Axel Polleres Semantic Web Publishing with Drupal. DERI, 2009. Available at <http://www.deri.ie/fileadmin/documents/DERI-TR-2009-04-30.pdf>

Structure

The rest of this thesis is structured as follows. We will begin by introducing some preliminaries in Chapter 2. These will be useful to understand the details of the solutions we implemented and which are further detailed in Chapter 3. We will also look at the adoption of our solutions, along with a user evaluation in Chapter 4. To facilitate the adoption of our approach, a minimal RDF support for Drupal core will be proposed in Chapter 5. The Neologism project, a more advanced solution for RDF vocabulary publishing, will be presented in Chapter 6. Finally, we will conclude in Chapter 7 with an outlook on future work and a use case for our approach.

Chapter 2

Preliminaries

In this chapter, we will first set our approach into context with some history and related work in Section 2.1 – as we will see the Semantic Web and CMSs have crossed paths various times in the past years, still we emphasize the complementary nature of our approach. Sections 2.2 and 2.4 will describe some of the technologies and aspects of the Semantic Web relevant for this thesis. Finally in Section 2.6, we will look at various Content Management Systems and their relation to RDF.

2.1 History and Pre-Existing Work on Content Management Systems

Staab et al. [57] proposed – at a stage where CMSs as we know them today were still in their infancy – what could be conceived as an ontology based CMS. The idea was to separate Web Content from domain models, very similar to the core ideas that made the success of CMSs. Similarly, Ontowebber [36] promoted the creation of Websites out of ontologies with a system built on a native RDF repository. These approaches though focus on ontologies and ontology design themselves as a backbone to organise data on Web sites. On the contrary, CMSs typically support very lightweight structuring of information items supported by user interfaces limited to the needed functionality to structure data for Web presentation, rather than full ontology editing tools. Moreover, current CMS typically rely on off-the-shelf relational databases to store data on the backend, rather than RDF repositories. Our module for Drupal thus has a very orthogonal goal: rather than building an ontology-based CMS we aim to extract and link ontologies from the content models and within the tools that typical site administrators are familiar with nowadays. We believe that this strategy – taking users from where they are in an unobtrusive way – to be the key enabler to leverage Semantic Web technologies with low entry barriers.

Somewhat closer to our approach are the ideas presented by Stojanovic et al. [58], where a relational database schema underlying a CMS is mapped to RDF Schema or Frame Logic. KnoBot[6] is a CMS entirely based on RDF by Reto Bachmann-Gmür. A more recent approach of *Database-level RDF exporters* – Triplify [4] – follows

a similar path, providing a generic mapping tool for lifting relational databases into Linked Data by dedicated mappings from a relational Schema to RDF. It requires understanding of database technologies and of the application's internal database schema. It should be mentioned that Triplify even provides some predefined mappings to wrap Drupal sites' backend databases into RDF. Again, our approach is significantly different. Due to the flexible nature of Drupal the underlying database used to store the data underlying a CMS based site does not necessarily reflect the sites content model and its constraints. Thus, the relational schema might vary between different versions of Drupal on the one hand, and on the other hand, the effects of changing the content model to the database schema underneath by the site administrator are not always obvious. Actually, a good part of the success of CMSs is grounded precisely in the fact that site administrators do not need to delve into details of the underlying database system or schema. Our approach works on a more abstract level directly in the UI/API the site administrator is used to, where all the information about the content model is available, which we believe to model the information structure more adequately than the underlying database schema. The administrator does not need to know anything about the underlying database to create mappings to existing RDF vocabularies or expose RDFa on her site.

SIOC exporters and similar fixed-schema RDF generators cannot deal with the case where the site schema and its mapping into RDF terms is defined by the site administrator at setup time. *Semantic MediaWiki*¹ and similar systems such as the *Semantic Forms* extension² or UfoWiki [50] address a different use case: all content authors, rather than just the site administrator, collaboratively define the structure of the site. We address the common case where the site's basic structure should not be changed by individual content authors.

A main difference to Semantic Wikis is that these typically provide semantic annotations at editing, i.e. by the content providers themselves, whereas in CMS the administrator and content provider roles are typically more clearly separated. We aim at providing semantic annotations at design time for the site administrator, but - additionally - also enabling posteriori annotation of existing Drupal sites, which becomes possibly by the flexibility of Drupal's module system. In principle, our approach is also applicable to Semantic Wikis with some adaptations, but we envision with targeting CMS a much broader audience. It is worth highlighting the survey [53] carried out by the W3C RDB2RDF Incubator Group which includes a more exhaustive list of approaches for mapping of relational databases to RDF.

Finally, none of the above works provide an all-in-one solution for exporting, aggregating and mapping Linked Data from within a commonly used CMS. This is what distinguishes our work which is tailored for easy of use.

As for pre-existing work specifically in Drupal, a recent paper [23] describes the related SCF (Science Collaboration Framework) Node Proxy architecture. This module, developed specifically for biomedical applications, enables RDF from defined SPARQL queries to be mapped to specific Drupal content types. These mappings must be generated individually per content type - Node Proxy is not a general RDF-to-Drupal

¹<http://semantic-mediawiki.org/wiki/SemanticMediaWiki>

²http://www.mediawiki.org/wiki/Extension:Semantic_Forms

mapping facility, it does not support CCK, and it is read-only (into Drupal from Linked Data). Nonetheless, it was a significant first step along the path we develop further and more generally here. The Node Proxy architecture is currently used to retrieve Gene information from a common RDF store for StemBook,³ an open access review of stem cell biology. It will be superseded in the SCF Drupal distribution by the much more general and fully-featured modules we describe here. We will provide more details on the SCF use case in Section 4.

2.2 Semantic Web Technologies

2.2.1 RDF

The core technology on which the Semantic Web is based is the Resource Description Framework [39], a metadata model language for describing resources on the web. RDF became a W3C recommendation in 2004. As opposed to HTML which is tailored for displaying information for the human, RDF is designed to be read and understood by machines, so that independent parties can exchange data and seamlessly interoperate in an open world assumption. Typical applications of RDF range among other things from describing books and their authors, editors and publishers, to represent people, their relationship to each other and their online content. All these information can be merged together and can be viewed as global database: the Giant Global Graph [10].

URIs

In the context of the global web, each resource on the web must be identified by a unique pointer called a Uniform Resource Identifier (URI), which is a string of characters following the RFC3986 [11]. A URI can identify web documents like Web pages, but also concepts or things of the real world, like a person, a country or an idea. URIs allow disambiguation of terms which can have several meanings, for example in the case of 'apple':

```
<http://dbpedia.org/resource/Apple>  
<http://dbpedia.org/resource/Apple_Corps>  
<http://dbpedia.org/resource/Apple_Inc.>  
<http://dbpedia.org/resource/Apple_Macintosh>  
<http://dbpedia.org/resource/Apple_Valley>  
<http://dbpedia.org/resource/Apple_River>
```

where the URIs respectively identify the fruit, the music band, the company, the computer model, the valley and the river.

Sometimes in order to keep the notation shorter, prefixes will be used. This technique is taken from the Turtle [8] specification. As an example, the URI

```
<http://xmlns.com/foaf/0.1/name>
```

³<http://www.stembook.org/>

can be broken down into a base URI `<http://xmlns.com/foaf/0.1/>` which can be declare as prefix `foaf` and the RDF term name. The original URI can then be shortened to `foaf:name`, as long as the prefix is specified as follows:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
```

See Appendix A for a list of prefixes and their respective namespaces used in this thesis.

RDF statement and RDF graph

RDF describes resources with properties and property values. An RDF statement is the combination of 3 elements:

- a **subject**: the RDF resource which is being described and generally identified by a URI.
- a **property** or **predicate**: a resource with a name such as `author` or `knows` taken from an RDF vocabulary.
- an **object**: the value of the property – sometimes called property value – which can be either a literal like “Stéphane Corlosquet” or another resource.

Natural language statements such as (1) “I know Axel Polleres” and (2) “My name is Stéphane Corlosquet” can be easily represented in RDF, as depicted in the figure 2.1. It is important to differentiate these two types of statements where the object can either be another resource (top) or a literal (bottom). Although both objects could be perceived as names in natural language, they in fact are of different types: in the first statement (1), `http://www.polleres.net/foaf.rdf#me` is used as an identifier for the person whose name is Axel Polleres, while in the second statement (2), “Stéphane Corlosquet” is used as a literal value for the name. A literal is a value of type string, integer, boolean, date, etc. In RDF, there exists two types of literals:

- Plain literals are composed of a lexical form and optionally a language tag as defined by RFC-3066,⁴ normalized to lowercase. Examples of plain literal are `"garçon"@fr` and `"කොල්ලෙක්"@si`.
- Typed literals are composed of a lexical form and a datatype URI being an RDF URI reference. An example is `"13.4"^^xsd:float`.

All literals have a lexical form being a Unicode[2] string, which should be in Normal Form C.⁵

An RDF graph gets naturally constructed by cumulating several RDF statements, such as on the figure 2.2. Note the `rdf:type` property which allows to specify that both resources `me` and `Axel Polleres` belong to the class `foaf:Person`.

To make it easier to differentiate resources and literals in RDF graphs, it is common practise to represent resources in ovals and literals in rectangles. Another best practise

⁴<http://www.isi.edu/in-notes/rfc3066.txt>

⁵<http://www.unicode.org/unicode/reports/tr15/>

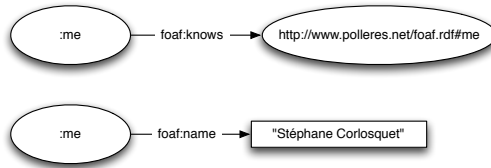


Figure 2.1: Basic RDF statements.

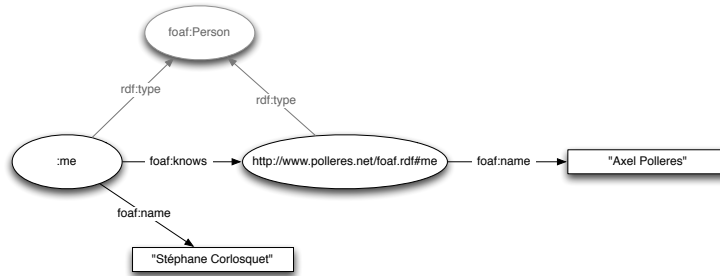


Figure 2.2: Basic RDF graph.

on the syntax level is to use capitalized terms for classes (ex. `Person`) while properties start with a lower case as in `knows` or `name`. These recommendations will be used in this thesis. The Turtle syntax (detailed in Section 2.3) will be used in most of the examples of this thesis.

In RDF semantics[33], RDF graphs are said to be directed, as they contain edges giving each relation (property) a specific direction. Any RDF graph can be decomposed in subgraphs. A subgraph is a subset of triples from the original graphs. This notion will be used later to define the concept of named graph in SPARQL queries in the section 2.2.3. It is also possible to merge RDF graphs together which produces a new graph as shown on the figure 2.3. This is possible thanks to the flexibility of RDF and unique URIs which are valid across applications and web documents, thus enabling interoperability on the Web of Data. The URI of the graph that contains each statement is sometimes added as fourth element in order to show its provenance: this element is called context or graph. Such a structure is referred to as quad or quadruple.⁶

Blank nodes and ground graphs

Sometimes called bnode or anonymous node, a blank node represents a resource that is not currently identified by a URI. There can be two reasons why the URI is absent: either (1) the value is meaningless and will therefore never exist, or (2) it could exist but it is currently missing and might be defined later. Most of the time, it is used to represent resources about which information is incomplete such as a person whose

⁶<http://sw.deri.org/2008/07/n-quads/>

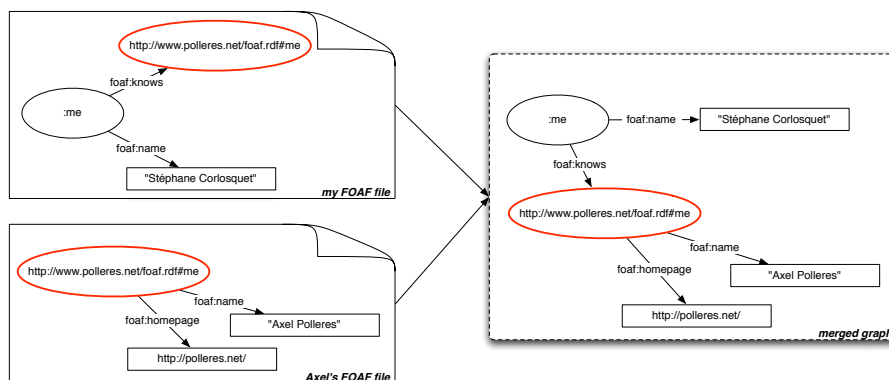


Figure 2.3: RDF Graph merge

identifier is unknown but may be defined by other properties like a name, a homepage, an email address, a phone number as shown in the figure 2.4. Essentially, blank nodes allow to talk about resources without having to know their identifier. Typically, implemented systems like the Redland API [9] will assign randomly chosen identifiers to bnodes such as `_:genid274952`. However these identifiers are not URIs and are only valid within the system implementing it. A graph containing no blank node is called a ground graph.

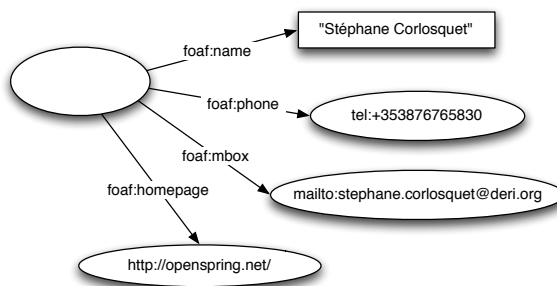


Figure 2.4: Blank node graph

2.2.2 Ontologies and Vocabularies

An ontology is formal representation of a set of concepts within a domain and how they relate to each others. Ontologies usually include the following elements:

- **Individuals:** instances constituting the basic components of an ontology, which can be concrete entities like people, animals, buildings, plants, as well as more

abstract individuals such as numbers and words. The latter group is sometimes included in the classes rather than individuals.

- **Classes:** concepts also called type, sort, category or kind, which are abstract groups, sets or collections of objects. A class can be subsumed by another class and is then called subclass of the parent class (superclass). Any member of the subclass will automatically be member of the parent classes. The consequence is that a subclass will inherit the properties of the parent (subsuming) class. In some ontologies, a class is only allowed to have one parent, while in most ontologies, multiple parent classes are allowed. One example of multiple inheritance is the a Kitten which could be a subclass of Cat and a subclass of Predator: as a result, the class Kitten would inherit the properties of the class Cat (species, color) and of the class Predator (has_prey).
- **Attributes:** aspects, properties, features, characteristics, or parameters that objects and classes can have. These attributes can be either a class or an individual. The value of an attribute can be a complex data type. In our example of the kitten, the attribute gender can only be one of the list {male, female}.
- **Relations:** ways in which objects of an ontology can be related to one another. A relation is of a particular type (class). The subsumption relation (superclass) is the most important one, which allows to built hierarchical tree-like structure where each object is the child of a parent class. Another type of relation is the meronymy relation (part of) used to relates objects which are combined to form composite objects.

Small, lightweight ontologies are sometimes called vocabularies. Ontologies typically comprise a much bigger set of terms than vocabularies – sometimes thousands of terms – and are therefore much more complex to design, create and maintain. They are also more expressive than vocabularies. In a nutshell, the term vocabulary is usually used to refer to simple, small, lightweight ontologies with limited expressivity.

Vocabularies and ontologies are commonly used in the Semantic Web to structure and describe the data available as RDF. Domain making use of ontologies are among others: healthcare, pharmaceuticals, biomedical, library science, artificial intelligence, etc.

RDF Schema

As explained previously, RDF describes resources with properties, values and classes. However it does permit to define application specific classes and properties. RDF Schema [19] is an extension to RDF which provides a framework to describe such custom classes and properties. Subclasses of classes can also be defined, forming hierarchies of classes (taxonomies) as shown in the example of figure 2.5 inspired from the Semantic Web for Research Communities ontology.⁷ These classes can then be instantiated in application specific RDF graphs. Classes in RDF can be compared to classes in object oriented programming languages but also show significant differences [48].

⁷<http://swrc.ontoware.org/ontology#>

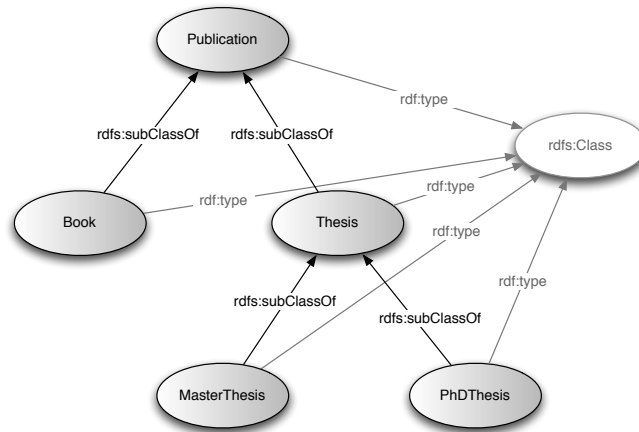


Figure 2.5: RDFS graph representing a simple hierarchy of classes.

Moreover, the `rdfs:domain` and `rdfs:range` properties provide a way to specify what type (class) of resources can be used as subject or object of an RDF statement. In the case of `foaf:knows`, both domain and range are `foaf:Person` (a `foaf:Person` can only know another `foaf:Person`). Another example is `foaf:firstName` whose domain is `foaf:Person` and range is a literal (the first name of a person must be a string). According to the RDF Semantics [33], RDFS entailment rules can be applied to infer extra triples as illustrated on Figure 2.6:

- *Transitivity of `rdfs:subClassOf`* The thesis “Technologies du Web Sémantique pour l’Entreprise 2.0” is typed as `PhDThesis`, which is a subclass of `Thesis` and `Publication` entails that the thesis is also of type `Thesis` and `Publication`.
- *Range* According to the FOAF specification [20], the range of the `foaf:maker` property is `foaf:Agent`, which entails that `<http://apassant.net/alex>` is of type `foaf:Agent`.
- *Domain* FOAF [20] specifies that the domain of the `foaf:firstName` property is `foaf:Person`, which entails that `<http://apassant.net/alex>` is of type `foaf:Person`.

Domain specific classes and properties are grouped together to form what is called RDF Vocabularies. FOAF,⁸ SKOS,⁹ SIOC¹⁰ are just a few of them. Because of the flexibility of RDF, terms defined in one vocabulary can be reused in another vocabulary, and subclasses of well know classes can be created where appropriate. Reusing existing vocabularies and specializing it to your application domain is encouraged in order to

⁸<http://foaf-project.org/>

⁹<http://www.w3.org/2004/02/skos/>

¹⁰<http://sioc-project.org/>

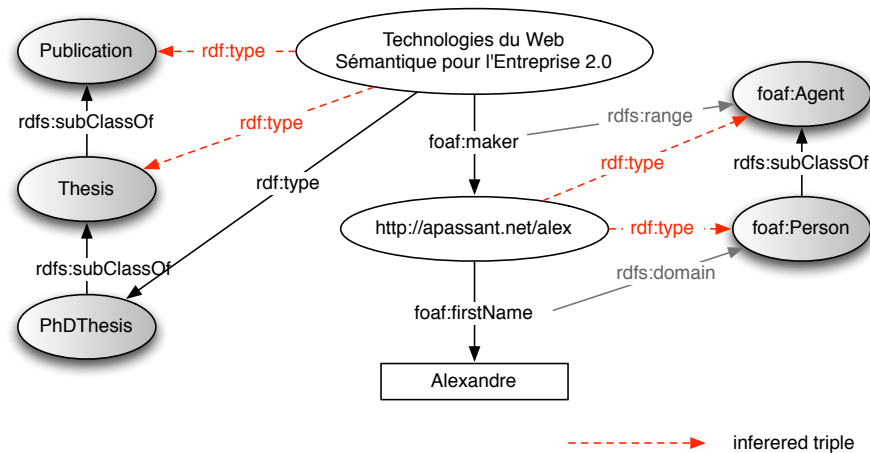


Figure 2.6: Graph representing RDFS entailment rules.

enable interoperability between applications, this is the reason why RDF vocabularies are published on the web. They are typically available in machines-readable format (RDFS) as well as human-readable format (HTML) to provide a reference to users willing to understand the vocabulary and eventually reuse it. The Neologism project [7] was created to make it easier the process of authoring and publishing vocabularies on the web using the latest standards in terms of content negotiation and dereferenceable URIs. Neologism will be described in more details in Chapter 6.

OWL

Several ontology languages exist: OWL [24], OWL 2 [29], KIF [28], CycL [43], RIF [18] and F-Logic [37]. The most well known is the Web Ontology language (OWL) which is built on top of RDF and RDFS for representing knowledge on the web. It is also based on older ontology language projects like OIL, DAML and DAML+OIL. OWL was designed to be used on the Internet and all its objects are defined as RDF and identifiable by URIs.

An OWL ontology contains a set of axioms which define some constraints on the individuals and the types of relationship between them. By providing these semantics, they allow systems to infer new knowledge from the data explicitly provided.

Some features of OWL which we will use later are as follows; this list is not meant to be exhaustive:

- **Properties.** OWL makes the distinction between two main categories of properties:
 1. *Object properties* link individuals to other individuals and are defined as instances of the `owl:ObjectProperty` class;

2. *Datatype properties* link individuals to data values and are defined as instances of the `owl:DatatypeProperty` class.

Both `owl:ObjectProperty` and `owl:DatatypeProperty` are subclasses of `rdf:Property`.

- **Multiple Domain.** A property can be defined as having multiple classes acting as domain by forming a union of all these classes with the `owl:unionOf` property.
- **Multiple Range.** In RDFS, a property can already have multiple range axioms. OWL further allows to restrict `rdfs:ranges` of properties to enumeration or complex class definitions, for instance a property can range over:
 1. a class of type `owl:DataRange` using the `owl:oneOf` construct with a enumerated list of data values;
 2. a complex class description, e.g. the `owl:unionOf`, `owl:intersectionOf`, etc. a set of classes.
- **Cardinality constraints.** In RDF, a given instance can have an arbitrary amount of values for a particular property. In OWL, the cardinality of a property can be restricted to:
 1. a maximum number of values with `owl:maxCardinality`;
 2. a minimum number of values with `owl:minCardinality`;
 3. a fixed number of distinct values with `owl:cardinality`;

By using a combination of these construct, one can specify that a property is required (at least one value), a specific number of values must exist or that a property must not occur. Note that a property constraint such as cardinality is defined within a class context by subsuming this class with an `owl:Restriction`, see Figure 3.1 for an example.

2.2.3 Querying RDF with SPARQL

In the previous sections we have discussed how to describe data in a uniform way with RDF as well as its structure with RDFS and OWL. We will now look at how to query this data.

SPARQL Protocol and RDF Query Language (SPARQL) [51] is a query language designed to allow querying over disparate RDF data sources. SPARQL has been standardized by the RDF Data Access Working Group (DAWG) of the W3C in January 2008, and is now very well spread in the RDF implementations. The features and rationale of the next version of SPARQL are currently being discussed at [38]. A SPARQL query consists of triple patterns, conjunctions, disjunctions, and optional patterns.

With SPARQL it is possible to extract values from structured and semi-structured data, perform complex joins across different databases and explore data by querying initially unknown relationships, all in a single query.

A basic SPARQL query can be written as follows:

```

SELECT ?title
FROM <http://example.org/book/book1>
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}

```

This query is composed of a `SELECT` clause identifying the variables to appear in the query results, a `FROM` clause indicating the dataset to be queried, and the `WHERE` clause providing the basic graph pattern to match against the data graph. Variables in SPARQL start with a “?” or a “\$”. The graph pattern of this example above is very simple and consists of a single triple pattern with a single variable `?title` in the object position. Only the bindings for this variable will be returned.

The SPARQL query processor is the server-side application which will do all the work of searching for the results over the databases involved in the query. During the execution of a SPARQL query, a typical processor will:

- fetch external RDF data defined in the `FROM` or `FROM NAMED` clauses, or alternatively directly evaluate the query on the default dataset stored in a local RDF store;
- search for the subgraphs matching the graph pattern in that data;
- bind the variables of the query to the corresponding parts of matching triple(s).

Some implementation of SPARQL processor are available: ARQ (Jena),¹¹ Openlink Virtuoso,¹² Sesame 2,¹³ ARC2,¹⁴ etc.

In order to make queries more readable and shorter, SPARQL allows the definition of prefixes similarly to the Turtle notation as shown earlier in the section 2.2.1. Prefixed names following the syntax `prefix:name` can then be used in the SPARQL query. Note that the prefix can be empty such as in the case:

```

PREFIX : <http://example.org/book/> .
PREFIX dc: <http://purl.org/dc/elements/1.1/title> .
SELECT ?title
WHERE
{
  :book1 dc:title ?title .
}

```

SPARQL queries examples

Assuming the store against which this query is executed contains the following triples:

```

<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> \
"The Social Semantic Web" .
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/creator> \
"John Breslin" .
<http://example.org/book/book2> <http://purl.org/dc/elements/1.1/title> \
"RESTful Web Services" .

```

¹¹<http://jena.sourceforge.net/ARQ/>

¹²<http://www.openlinksw.com/virtuoso/>

¹³<http://www.openrdf.org/>

¹⁴<http://arc.semsol.org/>

The triples matching the graph pattern of the query above is

```
http://example.org/book/book1 <http://purl.org/dc/elements/1.1/title> \
"The Social Semantic Web" .
```

and after binding the variable `?title`, the results of the query are:

title
"The Social Semantic Web"

To show the flexibility of SPARQL, let us query the graph above and retrieve all the titles of the books. This is done simply by changing the query pattern as the following:

```
PREFIX : <http://example.org/book/> .
PREFIX dc: <http://purl.org/dc/elements/1.1/title> .
SELECT ?title
WHERE
{
  ?book dc:title ?title .
}
```

In this query we have 2 variables in the query pattern (`?book` and `?title`) but only one of these appears in the SELECT clause and will be binded to the results:

title
"The Social Semantic Web"
"RESTful Web Services"

It is also possible to have more complex query patterns such as the following query which retrieve all the books and optionally their author:

```
PREFIX : <http://example.org/book/> .
PREFIX dc: <http://purl.org/dc/elements/1.1/title> .
SELECT ?title
WHERE
{
  ?book dc:title ?title .
  OPTIONAL { ?book dc:author ?author }
}
```

Note the SELECT clause now contains 2 variables. The results would be:

title	author
"The Social Semantic Web"	"John Breslin"
"RESTful Web Services"	

In addition to the SELECT queries, SPARQL supports three other query types. ASK simply returns "yes" if the query's graph pattern has any matches in the dataset and "no" otherwise. DESCRIBE returns a graph containing information related to the nodes matched in the graph pattern. Finally, CONSTRUCT is used to output a graph pattern for each query solution, which allows a new RDF graph to be created directly from the results of the query. It can be used to convert RDF data from one schema to another.

2.3 RDF serialization formats

Several RDF serialization formats have been designed over the past years to publish RDF online. They all share the same characteristic which is to represent the same RDF data but written (serialized) in different formats.

2.3.1 RDF/XML

RDF/XML is the most widespread format supported by all RDF parsers. It is an XML based serialization of RDF. It has been introduced in 1999 as part of the W3C specification of the RDF's data model and is therefore sometimes called RDF, which can sometimes lead to confusion: it is important to distinguish the XML format from the abstract RDF model itself. Its Internet Media Type is `application/rdf+xml`.

The RDF/XML format is known to be verbose and not easily readable for humans:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
    <dc:title>Tony Benn</dc:title>
    <dc:publisher>Wikipedia</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

2.3.2 RDFa

RDFa [1] is a set of extensions to XHTML which permits to embed RDF within HTML directly, hence the name RDFa: Resource Description Framework - in - attributes. RDFa became a W3C Recommendation in October 2008. Its recommended Internet Media Type is `application/xhtml+xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  version="XHTML+RDFa 1.0" xml:lang="en">
  <head>
    <title>John's Home Page</title>
    <base href="http://example.org/john-d/" />
    <meta property="dc:creator" content="Jonathan Doe" />
  </head>
  <body>
    <h1>John's Home Page</h1>
    <p>My name is <span property="foaf:nick">John D</span> and I like
      <a href="http://www.neubauten.org/" rel="foaf:interest"
        xml:lang="de">Einstrzende Neubauten</a>.
    </p>
    <p>
      My <span rel="foaf:interest" resource="urn:ISBN:0752820907">favorite
      book</span> is the inspiring <span about="urn:ISBN:0752820907"><cite
      property="dc:title">Weaving the Web</cite> by
      <span property="dc:creator">Tim Berners-Lee</span></span>
    </p>
  </body>
</html>
```

Because RDFa adds RDF terms to the HTML code directly, it has some advantages when it comes to User Interface for example, where it becomes possible to trigger specific events or layout depending on the meaning of a piece of HTML code. Some argue that RDFa adds more markup to Web pages, making them heavier to download and process,¹⁵ however this overhead can be reduced by optimizing the page load with gzip for example. Specific tools can help with this task.¹⁶

2.3.3 Notation3

Notation3[12], or N3, is a shorthand non-XML serialization of RDF. It is a class of several RDF formats whose relations are represented in Figure 2.7. Its Internet Media Type is `text/n3; charset=utf-8`.

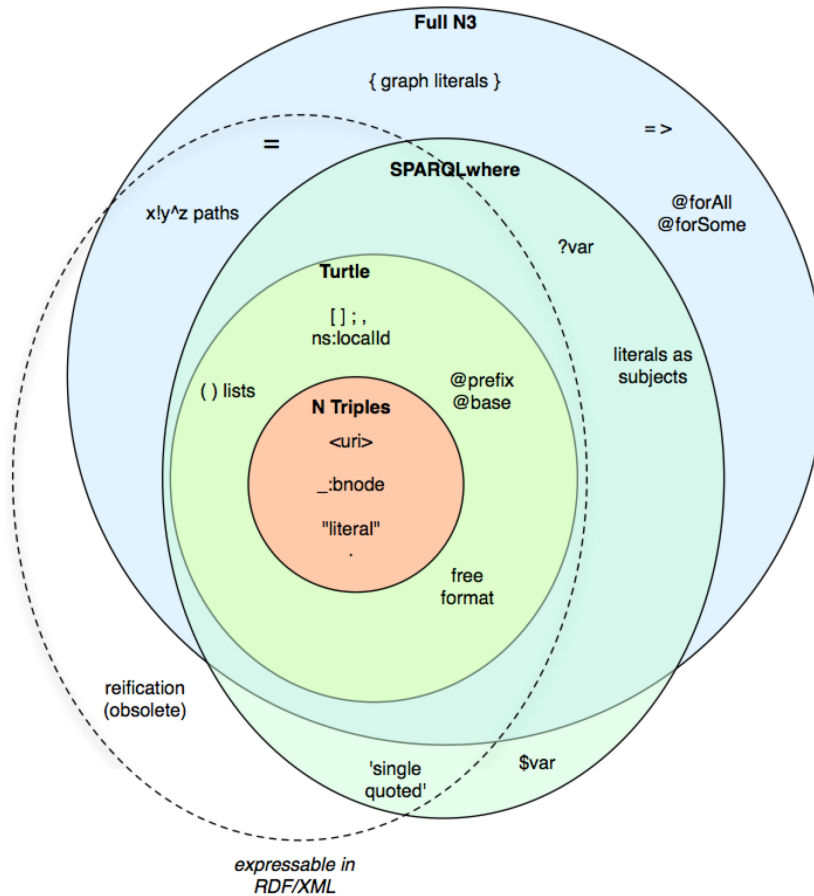


Figure 2.7: Notation 3 classification (from [12]).

¹⁵<http://buytaert.net/rdfa-and-drupal#comments>

¹⁶See for example YSlow developed by Yahoo!: <http://developer.yahoo.com/yslow/>

An example of N3 RDF data can read:

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.

<http://en.wikipedia.org/wiki/Galway>
  dc:title "Galway";
  dc:publisher "Wikipedia".
```

2.3.4 Turtle

Turtle (Terse RDF Triple Language) [8] is a subset of Notation 3 for expressing RDF data only (without the N3 rules). It is commonly used for its easy readability and is also very short. Its Internet Media Type is `text/turtle`.

2.3.5 N-Triples

N-Triples is a line-based, plain text serialisation format for RDF graphs. It is a subset of the Turtle format. Because it is line based, it makes it easy to parse and combine files or portions of files, since each line independent from each other as opposed to the RDF/XML format or Turtle which can include some nested resources. Its Internet Media Type is `text/plain`.

```
# The N-Triples statements below are equivalent to this RDF/XML:
#
# <rdf:RDF xmlns="http://xmlns.com/foaf/0.1/"
#   xmlns:dc="http://purl.org/dc/terms/"
#   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
#   <Document rdf:about="http://www.w3.org/2001/sw/RDFCore/ntriples/">
#     <dc:title>N-Triples</dc:title>
#     <maker>
#       <Person rdf:nodeID="art">
#         <name>Art Barstow</name>
#       </Person>
#     </maker>
#     <maker>
#       <Person rdf:nodeID="dave">
#         <name>Dave Beckett</name>
#       </Person>
#     </maker>
#   </Document>
# </rdf:RDF>

<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  <http://xmlns.com/foaf/0.1/Document> .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://purl.org/dc/terms/title> "N-Triples" .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/maker> _:art .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/maker> _:dave .

_:art <http://www.w3.org/1999/02/22-rdf-syntax-ns#> <http://xmlns.com/foaf/0.1/Person> .
_:art <http://xmlns.com/foaf/0.1/name> "Art Barstow".

_:dave <http://www.w3.org/1999/02/22-rdf-syntax-ns#> <http://xmlns.com/foaf/0.1/Person> .
_:dave <http://xmlns.com/foaf/0.1/name> "Dave Beckett".
```

2.4 Vocabulary publishing on the Semantic Web

Section 2.2.2 introduced vocabularies as one key concept of the Semantic Web. This section further emphasize their value and present some existing approaches to vocabu-

lary publishing.

Anyone who wants to publish information as RDF on the Semantic Web first faces the choice of which RDF Schema vocabulary or OWL ontology to use. Some areas, such as social networks (FOAF), online communities (SIOC) or general document metadata (DC) are covered by established vocabularies. Outside of these domains, registries like SchemaWeb¹⁷ and search services like Falcons Concept Search¹⁸ or Swoogle [26] assist in the task of finding vocabularies for niche topics, but what they find might be of insufficient quality, or might not cover all required terms, and at present many areas of interest are not covered by any vocabulary at all.

In summary, most efforts to publish information on the Semantic Web first require an effort to create, extend or modify an RDF Schema vocabulary or OWL ontology. But this is a complex and time-consuming task in itself. It involves:

- creating the formal specification of the vocabulary in RDFS or OWL,
- writing documentation that is clear and helpful for users of the ontology,
- keeping both documents in sync as the vocabulary evolves,
- archiving older versions of the documents,
- defining and maintaining mappings to related vocabularies,
- configuring the web server in accordance with W3C best practices [15].

2.4.1 The value of vocabularies

When we speak of vocabularies, we mean simple, “lightweight” ontologies, such as FOAF, DC, SIOC and SKOS. Expressivity is typically limited to RDF Schema plus selected OWL features, e.g. inverse functional properties and class disjointness. Their value is in providing common terminology for exchanging information between programs. The actual information is in the RDF instance data that is expressed with the vocabulary’s terms, while in more complex ontologies, the actual information lies in the definitions of the classes and properties. A vocabulary is created by publishing a description of its terms in natural using HTML or formal using RDFS/OWL language. Since classes and properties are identified by URIs, it is considered a good practice to make these URIs resolvable [14, 15]. This enables clients to look up definitions of the vocabulary terms, with the following benefits:

- Information publishers can refer to a specification. This is important to create interoperability around a vocabulary. The top ten most popular vocabularies of 2006 all have a such a specification.¹⁹
- RDF-aware tools such as data browsers (e.g. Tabulator [14]), SPARQL query builders and RDF instance editors can use the formal specification to improve

¹⁷<http://www.schemaweb.info/>

¹⁸<http://iws.seu.edu.cn/services/falcons/conceptsearch/>

¹⁹<http://ebiquity.umbc.edu/resource/html/id/196/>

the user experience, e.g. by showing friendlier labels and comments, listing available terms and providing widgets appropriate to a property's data type.

- Inference can be performed to increase recall when performing queries or lookups against RDF data, which is especially useful when terms are mapped to other vocabularies. Systems that use such techniques are the SWSE search engine [30, 35] and the Sindice semantic lookup index [47, 25].

2.4.2 Current approaches to vocabulary publishing

There exist various ways to author vocabularies which are described below.

Vocabulary maintenance with text editors and custom scripts. Many popular vocabularies such as FOAF and SIOC are maintained by a process involving hand-authoring of RDF and HTML files and custom scripts, e.g. SpecGen.²⁰ Often, complex custom web server configurations, e.g. in Apache, which is the most commonly used nowadays, are employed to follow best practices regarding content negotiation, MIME types and resolvable URIs [15].

Offline ontology editors. OWL ontology editors such as Protégé [40],²¹ TopBraid Composer²² and SWOOP²³ can be used to create the formal specification of a vocabulary. While being great tools for knowledge engineering professionals, these applications have a steep learning curve and they intimidate casual users. They use a file-based, offline model, where ontology files are stored on the local user's computer. Remote publishing, if supported at all, is an after-thought.

Web-based systems. Powl [3] is a Semantic Web development platform for PHP which handles the parsing, storing and serialization of RDFS and OWL knowledge bases for advanced ontology management. OntoWiki [5] provides basic ontology editing, but its main focus is the display and editing of RDF instance data. IkeWiki [54] allows to build an ontology from the annotations added to pages and links. MyOntology [56] focuses on collaborative editing in a larger community, in the hope of creating rich knowledge bases, while creation of simple vocabularies typically does not involve many collaborating users. Knoodl²⁴ is a hosted service with strong community features and an easy-to-use vocabulary editor, but it does not publish created vocabularies with resolvable URIs or according to best-practice guidelines.

We identify four points where we can simplify the process: (i) Instant web-based publishing instead of file-based offline editing. (ii) Focus on a limited subset of RDFS and OWL. (iii) No instance editing or browsing. (iv) Handling of HTTP details like URI management, content negotiation and redirects within the web-based application.

²⁰<http://sioc-project.org/specgen>

²¹<http://protege.stanford.edu/>

²²<http://www.topbraidcomposer.com/>

²³<http://www.mindswap.org/2004/SWOOP/>

²⁴<http://knoodl.com/>

2.5 Linked Data

The term *Linked Data* describes a method for exposing, sharing and connecting data on the Web via dereferenceable URIs. Tim Berners-Lee is the author of the four principles governing Linked Data, which are published in the seminal Linked Data design notes [13]. They read as follows:

1. Use URIs to identify things on the Web (resources),
2. Use dereferenceable HTTP URIs so that people and machine can look up these URIs,
3. When looking up an URI, i.e. resolving an RDF resource URI as an HTTP URL, provide useful information about the resource using standards such as RDF and SPARQL,
4. Include links to other resources in order to enable the discovery of more data.

The recent release of a series of long awaited W3C standards like SPARQL, GRDDL and RDFa, as well as the Linking Open Data community project²⁵ has lead to a great amount of datasets now available as Linked Data: in May 2009, the data sets contained 4.7 million RDF triples interlinked by around 142 million RDF links (cf. Figure 2.8). Among these data sets, DBpedia²⁶ – an RDF export of the large part of the structured data available in Wikipedia – is one of the most well connected due to its generic nature, the RDF export of the popular online citation index DBLP²⁷ is also of interest as we will use it later in our use case.

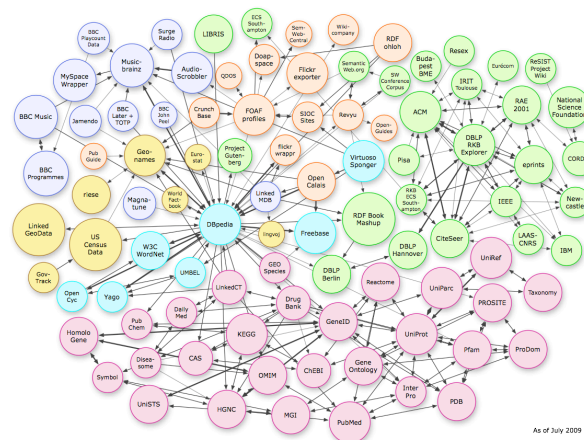


Figure 2.8: The Linking Open Data cloud (2009).

²⁵<http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/>

²⁶<http://dbpedia.org/>

²⁷<http://dblp.l3s.de/d2r>

2.6 Content Management Systems and RDF support

In this section, we will present the Content Management Systems in general. Later we will focus on the CMS which already have partially implemented some Semantic Web technologies.

2.6.1 Content Management Systems

Content Management System [61] is a broad term defining an application used to create, edit, delete and publish content in a consistent and structured way.

- Content is any type or unit of digital information or experiences, including text, images, graphics, video, sound, documents, records. Content is delivered via a medium such as the Internet, television, audio CDs, or DVDs. Any data stored electronically could potentially be considered as content.
- Content Management is the effective management of the content listed above, which combines rules, process and workflows. This management leads to a consistent structure and organization of the content in a CMS.
- System is the tool or combination of tools that make such content management possible in an efficient and effective way.

In this thesis we will specifically focus on the Web Content Management Systems which produce content available on the Internet in the form of HTML but also RSS, XML and other formats. They provide authoring functions designed to allow users with very little knowledge to create, manage and publish content online in a very easy manner. Compared to website builder applications like Adobe Dreamweaver or Mozilla SeaMonkey, Content Management Systems allow non-technical users to make changes directly on existing sites without requiring much training or specific software other than a regular Internet browser.

The features most Content Management Systems include are the following.

- identification of all key users and their content management roles;
- the ability to assign roles and responsibilities to different content categories or types;
- definition of workflow tasks for collaborative creation, often coupled with event messaging so that content managers are alerted to changes in content (For example, a content creator submits a story, which is published only after the copy editor revises it and the editor-in-chief approves it.);
- the ability to track and manage multiple versions of a single instance of content;
- the ability to capture content (e.g. scanning);
- the ability to publish the content to a repository to support access to the content (Increasingly, the repository is an inherent part of the system, and incorporates enterprise search and retrieval.);

- separation of content's semantic layer from its layout (For example, the CMS may automatically set the color, fonts, or emphasis of text.).

Furthermore, CMSs are able to handle a lot of tasks in the background which would be otherwise very time consuming and prone to errors like generating valid HTML pages following predefined templates, exporting data in RSS format for aggregation in other sites, etc.

Given the great variety of domains requiring a Content Management System in today's world, there is no single implementation able to deal efficiently with ever use case. While some applications such as Wordpress focus on a very specific type of content (blog entries), some other system are more general and flexible such as Drupal or Joomla!, and can be tuned to meet the use case requirements.

Various licences are used to release the Content Management Systems: some of them are proprietary (e.g. Sharepoint) but some others are Free and Open Source, allowing anybody to access the code and contribute to the project. Sometimes the dual licensing is used to cater to different audiences and benefits from several licences conditions.

Wordpress

Wordpress²⁸ is the most popular blogging platform currently in use. Because it is targeted to a very specific usage and does not require much skill to be installed, it has seen a excellent adoption among the bloggers.

Joomla!/Mambo

Joomla!²⁹ is a powerful Open Source Content Management System for building professional web sites easily. It is often the system of choice for small business or home users who want a professional looking site that is simple to deploy and use.

Drupal

Drupal³⁰ is a popular open-source content management system (CMS). It is among the top three open-source CMS products in terms of market share [55]. Drupal facilitates the creation of web sites by handling many aspects of site maintenance, such as data workflow, access control, user accounts, and the encoding and storage of data in the database.

Choosing a CMS

Choosing a CMS depends and the application needs and must be careful thought through. The IBM Internet Technology Group released a series of tutorials [60] on the CMS topic and included the following comparison chart.

²⁸<http://wordpress.com/>

²⁹<http://www.joomla.org/>

³⁰<http://drupal.org/>

	Drupal	Mambo	Typo3	Movable type	Word press	Text pattern
Ease of install	●	●	○	●	●	●
Learning curve	●	●	○	●	○	○
Session control	●	●	●	○	○	○
User control	●	●	●	●	●	●
Extensability	●	●	●	●	●	○
Scalability	●	●	●	●	○	○
Themability	●	●	●	●	●	●
xHTML/CSS	●	●	○	●	●	●

Figure 2.9: Web Content Management Systems comparison (from [60]).

2.6.2 CMSs and RDF support

The main RDF support we found for Wordpress is the *RDF tools* extension³¹ built by Benjamin Nowack. It is based on Dan Brickley's idea of SparqlPress³² and adds an ARC-based RDF store and SPARQL endpoint to the Wordpress blogging system.

The main Semantic Web project we found for Joomla! is the light-weight RDF syndication Google Summer of Code project³³ by Dan Le Phuoc. It relies on an architecture based on DERI Pipes [42] to process RDF data from Joomla! sites with the syndication component installed.

Drupal has multiple Semantic Web modules which have been developed by various members of the community over the past years. They are listed in Table 2.1. The main RDF API module is the base module on top of which we have built our solutions which are detailed further in Chapter 3.

2.6.3 Details on Drupal

For the reasons emphasized in the previous sections, we found Drupal to be the most appropriate CMS to use in our work. Therefore we delve into more details on Drupal 6 below which will be required to further understand this thesis.

Since the first release in January 2001, Drupal has seen a great increase in its popularity in the last years due to its ever growing community. While originally mostly used by charities and non commercial organizations, Drupal has since then been adopted by industry, ranging from companies like Sun, Yahoo! Research or Sony.³⁴ Today Drupal is used for personal sites, non profit as well as corporate and business sites. The amount

³¹<http://bnode.org/blog/2008/01/15/rdf-tools-an-rdf-store-for-wordpress>

³²<http://bzt.mfd-consult.dk/sparqlpress/>

³³<http://developer.joomla.org/gsoc2008/semantic-web/180-show-cases-of-light-weight-rdf-syndication-in-joomla.html>

³⁴For a list of well known Drupal sites, see <http://buytaert.net/tag/drupal-sites>

Name	Status	Description
RDF API	alpha	Enables the use of Resource Description Framework (RDF) metadata on Drupal site.
Calais	stable	Integration with the Calais Web service.
File Framework	alpha	File management framework storing information about the files in RDF.
Machine tags	unstable	Extends the taxonomy module to allows machine tags which are exported in RDF.
MOAT	unstable	Integration with the MOAT service[49].
Relations	alpha	Provides an API for arbitrary node relationships based on RDF.
Semantic Search	unstable	Faceted search front end for Drupal using an RDF-store on the back end as search index.
SIOC	stable	Exports Drupal forum and blog posts as RDF using the SIOC vocabulary.
SPARQL	alpha	Allows to query SPARQL endpoints via Drupal.
Views Datasource	alpha	Allows exports of Drupal Views in various formats like microformat and RDF.

Table 2.1: RDF related modules available for Drupal

of live Drupal sites on the Internet is estimated at more than 175.000 sites.³⁵

The most well known features of Drupal are the following:

- Friendly URLs using Apache's `mod_rewrite` capability
- Easily extensible using Drupal's module framework (The community has developed many useful modules that provide functions such as taxonomy display, jabber authentication, private messages, bookmarks, and so on.)
- A personalization environment for individualized content and presentation based on user preferences
- Role-based permission system to define access to the viewing and editing of content
- Content is fully indexed to support search
- Drupal is written on top of a database abstraction layer, so the framework can be easily extended to other database back ends
- Support for other content forms such as polls, threaded comments, and discussions and content syndication

³⁵According to the usage statistics for Drupal at <http://drupal.org/project/usage/drupal>. These statistics are gathered from the Drupal sites which have the *Update Status* module enabled. This module was introduced in Drupal 6 core. Therefore this number does not include many of the sites running an older version of Drupal and represents an underestimate of the actual amount of Drupal sites populating the Web,.

- Separation of content from styling in a templating system that uses HTML, CSS, and PHP
- Administrative support for logging, analysis, and Web-based administration
- Online help and support

Drupal is easily extensible via modules which can alter the default behavior and add extra functionalities. The Drupal eco-system is composed of the following.

- A core distribution *Drupal core* which is kept under permanent review by hundreds of developers and has a restricted set of committers: 2 per branch. The core provides basic APIs for the contributed modules to hook their code into. The latest Drupal core version is 6.
- A plethora of more than four thousands contributed modules (plugins) which are authored by various members of the Drupal community. A module is a set PHP files to be downloaded on the server of a Drupal site. These files are automatically detected by Drupal which will let them add or change some of its features or behavior. These modules are available for download for free on Drupal.org.³⁶

A typical Drupal site involves several types of contributors as shown on Figure 2.10.

Module developers write the PHP code which adds specific functionalities to a Drupal site. The type of functionality a module can provide is very broad: improved user interface, extra fields, enhanced search, different content display, various export formats, extended statistics and much more.

Site administrators initially sets up a site by installing the core Drupal web application and choosing from a large collection of *modules* that add specific functionality to the site, such as improved user interface, enhanced search, various export formats, extended statistics and so on. Site administrators need a fair bit of technical knowledge to choose and configure modules, but usually do not write code. Also sometimes called *Drupal architects*, they will maintain the site healthy, make sure its code base is kept up to date against new security threats, and eventually add new functionalities depending on the need of the users of the site.

Content authors produce the main content of the site. In some cases there might be a need for moderators who will take care of reviewing the content and publishing it. Some more complex workflow can be setup where several stages of reviews are necessary for example.

Visitors sometimes can leave comments or rate the content of a site, although most of the time, they simply read the content.

³⁶See <http://drupal.org/project/modules> for an exhaustive list of modules available for Drupal.

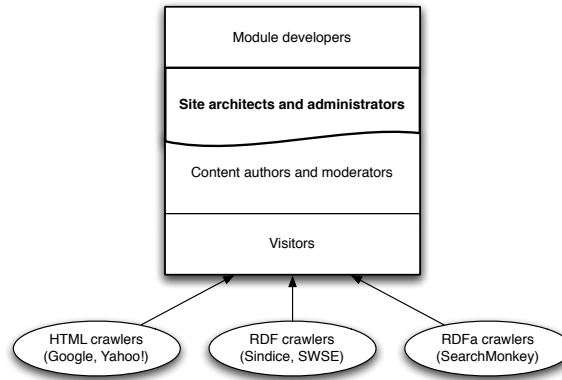


Figure 2.10: Hierarchy of contributors on a typical Drupal site

Among these categories, the site administrators are key: they are the ones who have a handle on both the data and its structure. Our approach targets the site administrators at the module level where they have full control over the site, so that the rest of the chain down to the simple visitors of the site can benefit from it. It enables the site administrators to put their sites onto the Web of Data and integrate RDF data in their sites. Nothing is changed for content authors making our approach as transparent for most people. Normal visitors will not see any change but we enable a new kind of visitor: those using RDF-enabled browsers, or machine clients that understand RDF.

The Content Construction Kit

Each item of content in Drupal is called a *node*. Each node has a title and a unique id used internally to refer to the node. Each node gets a page typically published at the URL `http://site.com/node/{node_id}`. Nodes can be created, edited and deleted by content authors. Some modules extend the nodes, for example a taxonomy module allows assignment of nodes to categories, and a comment module adds blog-style comment boxes to each node.

The *Content Construction Kit (CCK)* is one of the most popular and powerful modules used on Drupal sites. The reason for such a popularity is that it caters for one of the most painful tasks of module developers and site administrators: content structure and storage. It allows, via a User Interface, the site administrator to define types of nodes, called *content types*, and to define *fields* for each of these content type. Fields can be of various kinds such as plain text fields, dates, email addresses, file uploads, or references to other nodes. Additional kinds of fields can be created programmatically via modules.

The actual storage of the content type and its fields data is handle by the CCK API which will optimize the storage of this data in the database depending on for instance if the field can have multiple values or is shared by more than one content type.

When defining content types and fields, the site administrator has to provide the

following information: *ID*,³⁷ *label*, and *description* for content types and fields. Additionally, CCK allows to specify the following constraints on fields:

- *Cardinality*: fields can be optional or required, and may have a maximum cardinality.
- *Domain*: fields can be shared among one or more content types.
- *Range*: fields can be of type text, integer, decimal, float, date, file attachment, or node reference; fields of type node reference can be restricted to nodes of specific content types; fields of type text can be restricted to a fixed list of text values.

Thus, site administrators use CCK to define a site-specific content model, which is then used by content authors to populate the site. The focus of the work we are presenting here is to expose the site content as RDF and the CCK site content model as OWL ontologies.

Motivating example: the project blogs site

Throughout this site thesis we will use a running example to illustrate our approach: a project blogs website³⁸ contains various information about the researchers at DERI and their collaborators, including their publications, blog posts and projects they work for. Our goal is to expose the site data and structure in a machine-readable form as well as pull in data available from the Linked Data cloud or other Drupal sites in order to enrich the information displayed on the site.

Figures 2.11, 2.12, 2.13 and 2.14 show the typical look and feel of a Drupal page and administrative interface for the *Person* content type, without our extensions installed. This content type offers fields such as *name*, *homepage*, *email*, *picture*, *colleagues*, *blog url*, *current project*, *past project*, *publications*, *contributions*.

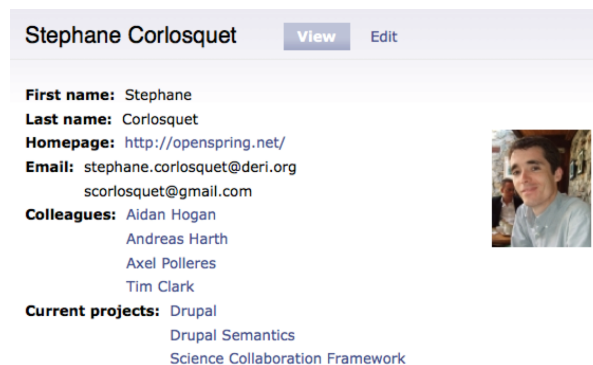


Figure 2.11: User profile page built with Drupal’s CCK.

³⁷a string containing lower case alphanumeric characters and underscores.

³⁸Demo-site running on Drupal 6 is available at <http://drupal.deri.ie/projectblogs/>

An example of node (page) of the type *Person* is depicted on Figure 2.11 where all the fields are listed with their respective values. These values are formatted depending on the type of field they belong to, e.g. a value of type link such as for the field *homepage* is a link to `http://openspring.net/`, a value of type Node reference such as for the field *colleagues* will be a link to the page of Aidan Hogan for instance, which is hosted on the same site. Note also the *View* and *Edit* links at the top which are available to logged in users who have the permissions to edit the page.

Figure 2.12: Administration page of the Person content type in Drupal’s CCK.

Figure 2.12 presents the basic form of the *Person* content type. It does not contain any information about the field but more general settings like the name of the content type, whether it should allow comments, have revisions, etc.

Label	Name	Type	Operation
+ Name	Node module form.		
+ First name	field_fn	Text	Configure
+ Last name	field_ln	Text	Configure
+ Picture	field_picture	Image	Configure
+ URI	field_myuri	Link	Configure
+ Homepage	field_homepage	Link	Configure

Figure 2.13: List of fields of the Person content type in Drupal’s CCK.

The fields form for the *Person* content type is displayed on Figure 2.13. This form allows to easily reorder the fields by a “drag and drop” technique, add new fields, remove existing fields or access the configuration form for a field.

The configuration form for the field *gender* appears on Figure 2.14, where it can be set as required. Its cardinality can be specified as well as, if appropriate, a list of allowed values – in which case the form will present a drop list of choices for this field.

Global settings

These settings apply to the *Gender* field in every content type in which it appears.

Required

Number of values:

1

Maximum number of values users can enter for this field.

Warning! Changing this setting after data has been created could result in the loss of data!

Allowed values

Create a list of options as a list in **Allowed values list** or as an array in PHP code. These values will be the same for *Gender* in all content types.

Allowed values list:

Male
Female

Figure 2.14: Defining constraints on the gender field in Drupal's CCK.

Particularly, we will illustrate in the further sections how to extend the *publications* field to automatically display a list of publications pulled from various data endpoints.

Chapter 3

Solutions to connect Drupal to the Web of Data

Given a Drupal CCK content model consisting of content types, fields, and nodes that instantiate the content types, let us discuss the best ways of representing it in RDF. We consider the following features desirable for the RDF output which are in line with the Linked Data principles and best practices [16]:

- (i) **Resolvable HTTP URIs** for all resources, to take advantage of existing tools that can consume Linked Data style RDF content. That is, when resolving URIs, one should find machine-readable information describing the URI. On the one hand in Drupal, typically URIs of the running site are simply URLs pointing to Web pages, but on the other hand, each of these pages also represents a node of a certain content type in the CCK content model. Thus, in our model, each node becomes an RDF resource, and the HTML Web page describing the node is enriched with RDFa [1] that reflect the links in the content model. That is, for each node URI
 - we add an `rdf:type` triple asserting class membership in a class representing the content type of the node to the page.
 - we add a triple for each *field* displayed on the page where the predicate is a property representing the field itself and the field value is either a datatyped literal (for text, integer, decimal, float, or date fields) or the URI of the respective node reference.
- (ii) **Expressing Drupal CCK constraints in RDFS + OWL.** Constraints that are defined on the types and fields (domains, ranges, cardinalities, disjointness) should be automatically published as RDF Schema [19] or OWL [24] expressions. We will enable this by an auto-generated *site vocabulary* that is linked from the site and which describes all content type and field URIs as classes and properties in an ontology that reflects exactly the constraints expressible in CCK. We will explain this mapping in detail in Section 3.1 below.

- (iii) **Re-use of published ontology terms.** To support sites talking about arbitrary domains, pre-defined/auto-generated RDF classes and properties are most likely insufficient. In fact, the default site vocabulary only comprises an isolated ontology not related to the rest of the Semantic Web. In order to link content to existing ontologies, we have to provide a handle for the site administrator to select terms from existing ontologies when setting up the content model. This requires that sites may reuse/import vocabulary terms from common existing ontologies. We will explain this in more detail in Section 3.1.
- (iv) **Safe vocabulary re-use.** Mixing the content model constraints with constraints of a published ontology might have unintended semantic effects, especially since most site administrators will not be familiar with the details of the OWL semantics. The system must prevent such effects as far as possible. Practical examples are included in Section 3.1.
- (v) **Exposing a query interface.** We rely on the SPARQL protocol [51] here, i.e. the site should expose its data in a SPARQL endpoint associated with the site. This should be easy to set up and should not be a burden for the site administrator.
- (vi) **Reuse of Linked Data.** Where possible, linkage to other instances of the Linked Data cloud should be defined.
- (v) **Vocabulary authoring** The expressivity of the site vocabulary which is just restricted to the content model of a site might not be enough for those who would like to define her own vocabulary. To this purpose, we created a lightweight vocabulary authoring tool implemented in Drupal: Neologism. As this tool is not directly connected with the other modules, it will be described separately in Chapter 6.

These features strike a balance between preserving as much information as possible from the original content model, keeping the barrier to entry low, and enabling interoperability between multiple data publishers and consumers.

Integrating the above features in a such a system as Drupal is challenging for several reasons:

No dedicated development staff. Site operators of smaller we sites are not expected to “program” RDF export themselves, it has to be supported by adequate software or plugins that fit in the typical Drupal site administration tools and modules, and that enable a one-click, low entry barrier solution to expose site content in RDF.

Per-site schemas. The domain schema differs from site to site. The mapping from the site’s schema to a corresponding RDF Schema or OWL ontology cannot be pre-defined by a software developer; it must be defined by the site operator, and we have to provide adequate support to reuse existing vocabularies and ontologies.

No ontologists. Site administrators will have little interest in learning the details of RDF and description logics. The process of configuring RDF support has to be simple and straightforward, or else it won’t be used.

We will now present our extensions for Drupal, which shall fulfill the goals outlined above.

3.1 From Content Models to Site Vocabularies

Administrators use CCK to define a site-specific content model, which is then used by content authors to populate the site. The focus of our work is to expose (i) such CCK site content models as an OWL ontologies that reflect the site structure which the designer had in mind and (ii) the site content as RDF data using this ontology.

We have implemented a Drupal module that enhances Drupal's CCK with the ability to auto-generate RDF classes and properties for all content types and fields and displaying RDFa triples on the site as outlined in item (i) above. Thereby we provide zero-effort RDFa output for any Drupal site, as long as no mappings to well-known public vocabularies are required. Further – besides the data – we can map CCK data models to a fragment of OWL outlined, addressing (ii) as follows.

3.1.1 Building a Site Vocabulary

We build a so-called *site vocabulary*, i.e., an RDFS/OWL ontology which describes the content types and fields used in the data model as classes and properties. It is automatically generated and published on the site under the default namespace

```
http://siteurl/ns#
```

which we subsequently denote by the namespace prefix `site:`.

Firstly, the field and type names are extracted from field and type *IDs* from CCK, such that – following common conventions – fields are assigned a property name starting with a lower case character, and content types are assigned a class name starting with an upper case character. Field and content type labels and descriptions are likewise exported as `rdfs:labels` and `rdfs:comments`. Here goes a typical content type and field definition extracted from CCK into RDFS:

```
site:Person a rdfs:Class;
  rdfs:label "Person";
  rdfs:comment "Researchers in DERI and their collaborators";
site:fn a rdf:Property;
  rdfs:label "First name";
  rdfs:comment "First name of a Person";
```

Likewise, field constraints from CCK are reflected in the site vocabulary: *Cardinality* is mapped to cardinality restrictions in OWL, i.e. *required* fields are restricted to `owl:cardinality 1`. whereas fields with a maximum cardinality *n* are restricted to `owl:maxCardinality n`. For instance, if we assume that each *Person* is required to have a *name* field, works in at most 5 *projects*, these constraints in CCK would be exported to OWL as follows.

```

site:Person a rdfs:Class; rdfs:subClassOf
[ a owl:Restriction;
  owl:onProperty site:name;
  owl:cardinality 1],
[ a owl:Restriction;
  owl:onProperty site:project;
  owl:maxCardinality 5].

```

Figure 3.1 gives an illustrated view of this set of restrictions.

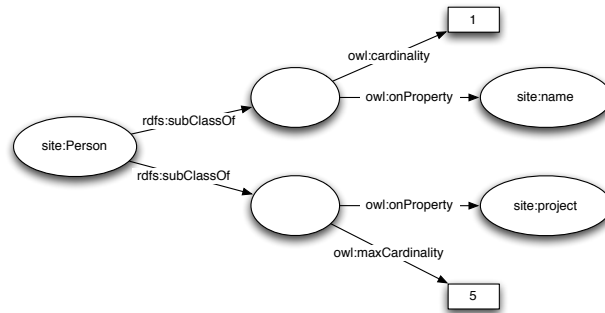


Figure 3.1: Graph of a set of restrictions on a class.

Domains are reflected by `rdfs:domain` constraints. Here, fields used by a single type can be modeled by a simple `rdfs:domain` triple. For instance, assuming that the *colleagues* field for *Persons* is not shared with any other content type in the current content model, we can simply write:

```

site:colleagues rdfs:domain site:Person.

```

CCK fields shared among several types have the union of all types sharing the field as their domain. E.g., since *Publication* and *Blog post* share the *title* field, the site vocabulary contains

```

site:title rdfs:domain
[owl:unionOf (site:Publication site:Blog_post)].

```

Ranges of fields are analogously encoded by `rdfs:range` triples. Additionally, we distinguish here between fields of range text, integer, decimal, float, or date, and those referring to file attachments, or node references. For the former, firstly, we assign the datatypes supported in Drupal with their respective XML Schema datatypes, i.e. *text* → `xs:string`, *integer* → `xs:integer`, *decimal* → `xs:decimal`, *float* → `xs:float`, or *date* → `xs:date`. Secondly, as we know that Drupal will only export literal values of specific datatypes for these, we can constrain the site vocabulary further, by declaring the respective fields of type `owl:DatatypeProperty`. For instance, the text field *name* is reflected in the site vocabulary as:

```

site:name rdfs:range xs:string;
  a owl:DatatypeProperty .

```

Fields that range over texts restricted to a fixed list of text values will be assigned a respective enumerated class of values using `owl:DataRanges`, e.g. *gender* is modeled as

```
site:gender a owl:DatatypeProperty;
  rdfs:range
[ a owl:DataRange; owl:oneOf ( "male" "female" ) ].
```

Finally, fields that range over file attachments (which get a URI in Drupal) or node reference, are declared of type `owl:ObjectProperty`. Fields ranging over more than one content type are reflected in the site vocabulary by `owl:unionOf`. E.g., *contributions* may be *Publications* or *Blog posts*, respectively.

```
site:origin a owl:ObjectProperty; rdfs:range
[ owl:unionOf ( site:Publication site:Blog_post ) ].
```

3.1.2 Adhering to Linked Data principles

Following the conventions mentioned in the previous section, the site vocabulary is generated and published automatically at the site URL under the default namespace `http://siteurl/ns#`, which we denoted by the namespace prefix `site:` in the examples before. Likewise, any Drupal page on a site will be annotated with RDFa triples that dereference terms of this site vocabulary as classes and properties linking Drupal content nodes as subjects and objects. We are inline with the Linked Data principles and best practices [15] as we provide resolvable HTTP URIs for all resources: each of the pages also represents a node of a certain content type in the CCK content model. That is, in our model, each node becomes an RDF resource, and the HTML Web page describing the node is enriched with RDFa [1] that reflect the links in the content model. We distinguish between the document – typically of type `foaf:Document` – and the resource it describes which are linked to each other via a `foaf:page/foaf:topic` relationship. By this design, any Drupal site using our module is off-the-shelf amenable to any existing tool that can consume Linked Data style RDF content.

3.2 Mapping Content Models to Existing Ontologies

While the functionality we have described previously fits Drupal sites well into the Linked Data world, so far, we have created nothing more than an isolated ontology based on the existing site content model. However, the benefits of this exercise remain limited, unless we additionally allow linking the site vocabulary to existing vocabularies and ontologies populating the Semantic Web. For instance, instead of just exposing the *Person* type as a class in the site vocabulary, we might want to reuse a class in an existing ontology, such as `foaf:Person` from the FOAF¹ ontology which some other publishers on the web already use. Likewise, we may wish to state that a *Publication* is actually a `foaf:Document`, or that the *Persons* are linked to their *Publications* by the `dc:creator` property from Dublin Core,² etc.

¹<http://xmlns.com/foaf/0.1/>

²<http://purl.org/dc/elements/1.1/>

To this end, our module adds a new tab “Manage RDF mappings” to the content type administration panel of CCK for managing such mappings to existing ontologies, cf. Figure 3.2. An autocomplete list of suggested terms is shown, based on the keyword entered by the user. The terms are coming from two different sources, which are detailed below.

3.2.1 External vocabulary importer module

The module RDF external vocabulary importer (evoc)³ has been created to allow the import of vocabularies available on the web and make the imported terms available in the mapping interface. The site administrator simply needs to fill in a form with the vocabulary URI and the prefix to be used in the system to refer to the vocabulary term when using the CURIE format. We assume that the external vocabularies have been created using a tool such as Protégé,⁴ OpenVocab,⁵ or Neologism,⁶ and published somewhere on the Web in RDF Schema or OWL format. A set of SPARQL queries are sent against the vocabulary to extract its classes and properties and some information about them like label, comment, superclass, domain, range. These are then cached locally to provide a smoother user experience. Given their popularity, the Dublin Core, FOAF and SIOC vocabularies are imported automatically upon installation of the evoc module.

3.2.2 External ontology search service

We have also developed an ontology search service to help users to find ontologies published on the Web of Data. The search engine is entity-centric, i.e. instead of returning a list of relevant ontologies, it returns a list of relevant ontology entities to the user request. The service is currently covering cleaned up Web crawls of DERI’s SWSE.org [30] and Sindice.com [47] search engines comprising Web data documents that define properties and classes.

Data Pre-Processing Before being indexed, a sequence of pre-processing tasks is performed on the ontology data. Among them, the most important ones are reasoning and splitting. Reasoning is applied on each ontology in order to infer useful information for a search engine, such as the hierarchy of classes and properties, the domains and ranges of properties, etc. Reasoning over semantically structured documents enable to make explicit what would otherwise be implicit knowledge: it adds value to the information and enables an entity-centric search engine to ultimately be much more competitive in terms of precision and recall [45]. Ontologies are then split into smaller pieces on a per-entity basis. For each authoritative URI⁷ found in the ontology, we

³<http://drupal.org/project/evoc>

⁴<http://protege.stanford.edu/>

⁵<http://open.vocab.org/>

⁶<http://neologism.deri.ie/>

⁷The Linked Data principles suggest that URIs for named entities should be dereferenceable and should link directly to the data describing the entity itself. Following this recommendation, we define as an *authoritative URI*, a URI that is dereferenceable and is linked to the ontology.

simply extract all its outgoing links.

Indexing Model The simplest semi-structured indexing method is to represent an ontology entity as a set of attribute-value pairs using a field-based approach [44]. For example, the ontology class `foaf:Person` will have fields *label*, *comment* and *subClassOf*; index terms are constructed by concatenating the field names with values of this field, for example as *subClassOf:Agent*.

Objects of type *literals* and *URI* are normalised (tokenised) before being concatenated with their field name. It is thus possible to use full-text search not only on literals, but also on URIs identifying ontology terms. For example one could search for "Agent" to match `foaf:Agent`, ignoring the namespace.

We allow search for plain keywords, combinations of keywords, or structured queries (e.g. `student AND subClassOf:Person` or `name AND domain:Person`). Search examples are shown in Figure 3.2. Details on improving the ranking of our search algorithm can be found in [59].

3.2.3 Mapping process

The terms suggested by both of the import service and the ontology search service can be mapped to each content type and their fields. For mapping content types, one can choose among the classes of the imported ontologies and for fields, one can choose among the properties. The local terms will be linked with `rdfs:subClassOf` and `rdfs:subPropertyOf` statements to the mapped terms in the site vocabulary, e.g.:

```
site:Person rdfs:subClassOf foaf:Person
```

Wherever a mapping is defined, extra triples using the mapped terms are exposed in the RDFa output of the page.

Additionally, we allow inverse reuse of existing properties. E.g., let us assume the site administrator imports a vocabulary `ex:` that defines a relation between projects and their members via the property `ex:has_member`. Our user interface also allows to relate fields to the inverse of imported properties. For instance, the *current project* field which relate people to projects could be related to `ex:has_member` in such an inverse manner, resulting in

```
site:current_project rdfs:subPropertyOf
  [ owl:inverseOf ex:has_member ] .
```

being added to the site vocabulary.

The use of subclassing and subproperties for mapping to existing ontologies – instead of reusing the imported terms directly in the definitions of the site vocabulary – is a simple way of minimizing unintended conflicts between the semantics of local vocabulary and public terms. Per OWL semantics, constraints imposed on the local term by the content model/site vocabulary such as the cardinality restrictions which we derive from CCK (see Section 3.1) will thus not propagate to the public term. This ensures *safe vocabulary re-use*, i.e. avoids what is sometimes referred to as “Ontology Hijacking” [34].

Intuitively, safe reuse means that a vocabulary importing another one does not modify the meaning of the imported vocabulary or “hijack” instance data of the imported vocabulary.

Let us assume that we would, on the contrary, directly use the imported properties and classes in the site vocabulary. That would cause problems. For instance, the export of the content model as described in the previous section contains the triple `site:colleagues a owl:ObjectProperty`. Would we have used the property `rel:worksWith` directly here, we would have changed the SWRC vocabulary,⁸ which, in itself doesn’t declare `rel:worksWith` a `owl:ObjectProperty`, explicitly. Direct reuse of existing classes in the site vocabulary would raise similar issues.

We want to emphasize, however, that the reuse of `rdfs:subclass`, `rdfs:subproperty` as well as inverse `rdfs:subproperty` relations alone is not sufficient to guarantee consistency of the site vocabulary. While the following proposition intuitively holds, this is no longer true as soon as external vocabularies are imported.

Proposition 1 *A site vocabulary that does not import any external ontologies is always consistent.*

Nevertheless, in case of importing external properties and classes, consistency can no longer be guaranteed without further restrictions, even if both the site vocabularies and the imported ontologies were consistent. This is easily illustrated by some examples.

Example 1 Assume that one would inversely assign the `foaf:homepage` property to `site:colleagues`. Given that `foaf:homepage` is an `inverseFunctional` property, this would imply functionality of `site:colleagues`, i.e. that each person would have at most one colleague. This would, possibly result in strange side effects on the site instance data such as yielding colleagues of a person with two or more colleagues equal. ◇

In other cases, contradicting cardinalities could even make site instance data inconsistent. Likewise the inverse reuse of Datatype Properties is problematic. To address this problem, when displaying external properties assignable to fields in the content model (see Figure 3.2) our tool extending CCK could make several restrictions such as not displaying properties with cardinality restrictions attached that do not comply with those specified in CCK for the respective field.

We emphasize that we do not aim to deploy a full OWL reasoner to detect all inconsistencies possibly introduced by ontology reuse in our system. The identification of syntactic fragments of OWL, which are safely usable in our tool without the need to deploy a fully-fledged OWL (DL) reasoner (i.e., which features used in the imported ontologies require which level of expressiveness for detecting possible inconsistencies by reuse) is on our agenda.

⁸<http://purl.org/vocab/relationship/>

Our ultimate goal is a tool which allows the site administrator to import classes and properties from existing ontologies in a fail-safe way such that no possible inconsistencies may be possibly introduced by the extended CCK editor and later filling of the site with content.

The reason why we are reluctant of deploying full reasoning services is that we want our tool to fit in the Drupal ecosystem as a normal module that is installable on a site without the need to install separate external software components such as a DL reasoner. Wherefore we restrict ourselves to lightweight reasoning by applying heuristics such as the detection of cardinality violations mentioned above. Our current approach is a best-effort approach to avoid “misuse” of imported ontologies as far as possible while deliberately keeping the interface simple. An alternative path would be the invocation of a reasoning service deployed as a web-accessible service.

It must be stressed that the whole mapping step is optional, and the main benefit of the Web of Data – exposing site data for re-use by third parties – is realized by the default mapping.

3.2.4 User experience

This section briefly introduces our CCK extension from a user point of view .

Content type and field mapping

The first example illustrated on the left of Figure 3.2 is the mapping of the Person content type to an RDF class. In order to ease the mapping process and prevent confusion between classes and properties, the module will only display RDF classes or RDF properties where appropriate. Moreover an AJAX autocomplete search through the imported terms allows the user to quickly identify the most relevant terms for each mapping. These measures help to make the mapping process a fairly straightforward task that does not require deep understanding of the Semantic Web principles. When typing `person` in the text field, a list of suggestions is pulled from both the local set of terms and from the external search service. The local terms appear first: `foaf:Person` is a CURIE and reuses the prefix definition of the site which can be defined during the import step. Then the list is completed with the terms from the external service, which includes a much greater variety of terms. This might help the user in discovering new terms or ontologies which she may not previously have encountered. Note that the external terms are displayed as full URI as we want to avoid imposing any prefix on them. We are currently evaluating the best way to display these.

The second example on the right of Figure 3.2 illustrates the case where the user wants to reuse the Relationship Ontology⁹ to express relationships between colleagues who work with each other. Despite the fact that the Relationship Ontology was not imported locally, the external ontology search web service (3.2.2) was able to suggest the right term URI.

⁹<http://purl.org/vocab/relationship/>

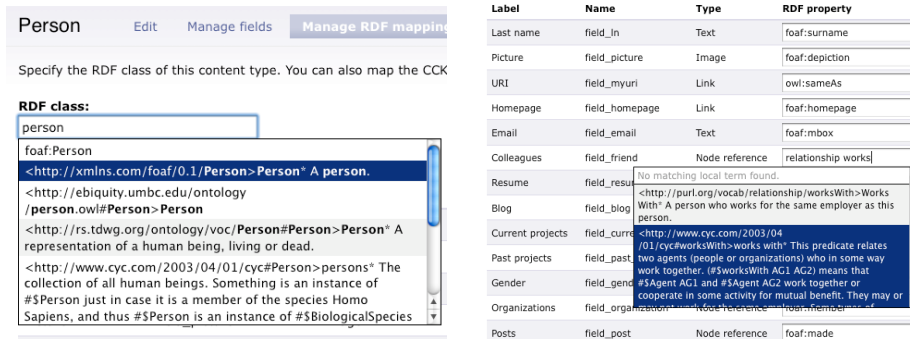


Figure 3.2: RDF mappings management through the Drupal interface: RDF class mapping (left) and RDF property mapping (right).

Vocabulary Import

Upon installation of the evoc module, the system will automatically import the commonly used vocabularies Dublin Core, FOAF and SIOC. As mentioned earlier, the user can also import vocabularies published elsewhere on the web via a form containing a field for the URI of the vocabulary to be imported and a field for the prefix which will be used across the application (see Figure 3.3). The system will then cache all the external RDF terms and propose them for mapping.

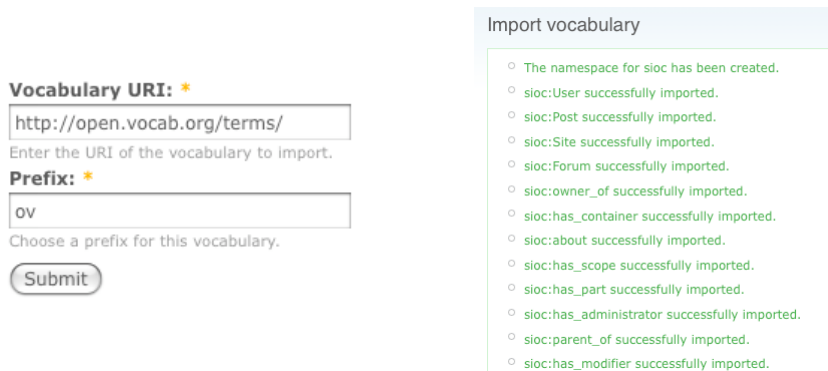


Figure 3.3: Form for importing an external vocabulary (left) and confirmation message after a vocabulary import (right).

3.3 Exposing and Consuming Linked Data with SPARQL

In analogy with the idea of Linked Data, cf. Section 2.5, we extend our use case to enable an environment of “Linked CMS sites”, as illustrated in Figure 3.4. Our goal is to use our project blogs website as a hub containing information federated from various

remote locations:

- DBLP is a public SPARQL endpoint containing metadata on scientific publications. It is part of the Linking Open Data cloud and runs on a D2R server.¹⁰
- The Science Collaboration Framework website which contains information about the SCF team and their scientific publications. It runs Drupal and the modules described in this thesis.

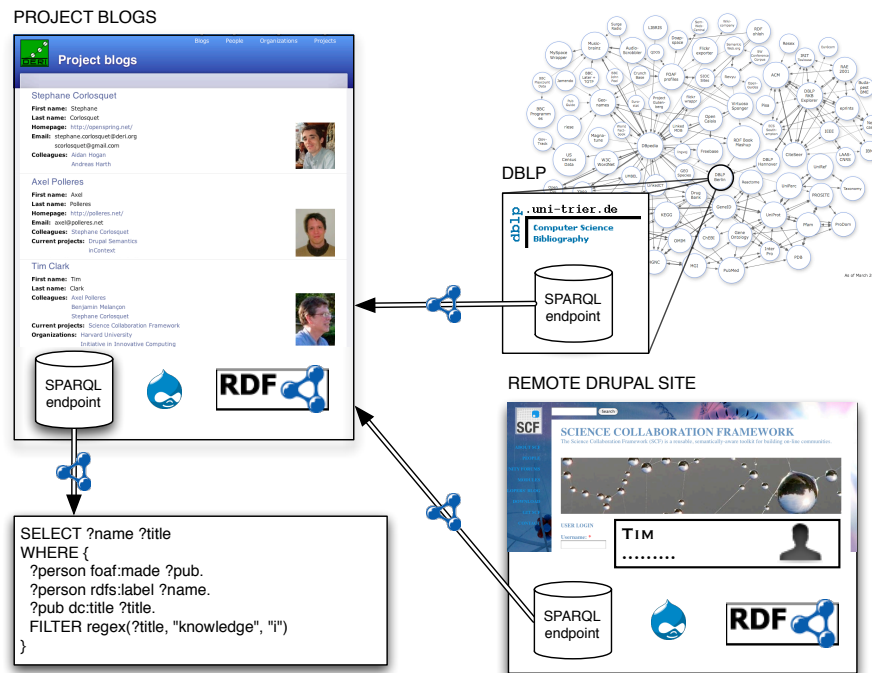


Figure 3.4: Extended example in a typical Linked Data eco-system.

3.3.1 Exposing RDF data with a SPARQL endpoint

The first step to ensure interoperability on the Web of Data is to provide an endpoint which exposes RDF data. The *RDF SPARQL endpoint* module uses the PHP ARC2 library.¹¹ Upon installation, the module will create a local RDF repository which will host all the RDF data generated by the RDF CCK module (see Section 3.1). The site can then be indexed with a simple click. The RDF data of each node is stored in a graph which can be kept up to date easily when the node is updated or deleted. Figure 3.5 (left) depicts a list of publications whose title contains the keyword “knowledge”.

¹⁰<http://www4.wiwiw.fu-berlin.de/bizer/d2r-server/>

¹¹<http://arc.semsol.org/>

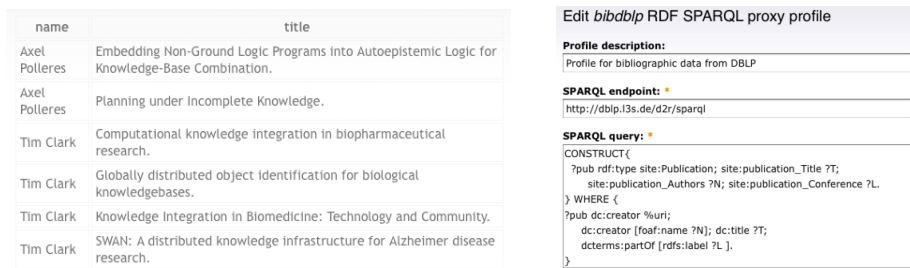


Figure 3.5: A list of SPARQL results (left) and an RDF SPARQL Proxy profile form (right).

3.3.2 Consuming Linked Data by lazy loading of remote RDF resources

With an ever growing amount of data available on the Semantic Web, one site cannot technically afford to host all the data available on the Web, even if the scope was restricted to a specific domain. Instead, each piece of data can be retrieved only when needed. This design pattern is known as lazy loading [27]. Information on the Web of Data is made available in endpoints which can be queried and from where information can be retrieved according to the specific needs of an application. The SPARQL query language [51] allows complex WHERE patterns able to extract the pieces of information desired, but another feature of SPARQL is the ability to specify a schema in which the RDF data should be returned. These CONSTRUCT queries are useful in the case where the information retrieved should retain a particular structure which would not fit in flat array of results such as a SELECT SPARQL query would provide.

Building atop the existing RDF schema provided by the RDF CCK module presented in Section 3.1, we developed *RDF SPARQL Proxy*, a module which allows to import RDF instances on demand, via a CONSTRUCT query in which the WHERE clause corresponds to the schema of the distant data on the SPARQL endpoint, and the CONSTRUCT clause corresponds to the local site schema defined by RDF CCK. As depicted on Figure 3.5(right), site administrator can define profiles, which specify the rules for creating or updating the local RDF instances based on the schema of the distant RDF data. In order to keep these profiles generic, we allow input parameters such as URIs. In this example, we map the publications by an author represented by her URI `%uri` along with the information about each publication (title, name of authors, conference) to our local schema. The value of the `%uri` parameter will be replaced for the value given as input, either in the address bar of the browser or by the API calling the RDF SPARQL Proxy module. For our use case, we have setup two such profiles: one for bridging the DBLP SPARQL endpoint to the project blogs website, and a second for bridging the Science Collaboration Framework website. When visiting Tim’s profile page, the relevant publication information will be fetched from both DBLP and SCF websites, and either new nodes will be created on the site or older ones will be updated if necessary.

Chapter 4

User Evaluation and Adoption of the Implemented Solutions

Our hypothesis and general rationale is that ease-of-use and a one-click solution to export Linked Data from CMSs will boost the Semantic Web. We exposed this hypothesis to a small user evaluation. What we aimed to prove is that linking a site to existing vocabularies by use of our Drupal module does not impose a significant burden to site administrators and the benefits of exposing Semantic Web data such as searchability may outweigh this negligible extra effort. In this section we evaluate the usage of the implementations we created, the extra effort required by our approach and some of its particular use cases and applications.

4.1 Usability

We did a limited-scale user evaluation aimed at showing that linking a site to existing vocabularies with our Drupal module does not impose a significant burden on site administrators. We argue that the benefits of exposing Semantic Web data such as greatly improved searchability, will typically outweigh this extra effort.

Our evaluation was carried out on a group of 10 users, moderately familiar with Drupal and more or less familiar with the Semantic Web. They were asked to set up a content model composed of 2 content types *Article* and *Editor* which included 5 and 4 fields, respectively. Each major step of the process was timed in order to be exploited in a statistical analysis. The documents which were given to the users during the evaluation are in Appendix B

Step 1. We asked our users of various degrees of familiarity with Drupal to set up a predefined content model composed of 2 content types which included 4 and 5 fields, respectively, using the CCK User Interface, an approach typical for site administrators on many typical Drupal sites.

Step 2. In a second step we evaluated the extra effort required to create RDF mappings to the fields defined in CCK. The pool of users was divided into two five-person groups. Group A (user 1 to 5) had to judge which mappings were the most

appropriate and group B (users 6 to 10) was given a list of predefined mappings.

In Figure 4.1, the extra step to map the content model to existing ontologies represented about half the time of the initial setup. On average, the extra time spent on specifying the mappings took 58% of the initial setup time for group A and significantly less (39%) for the group B. While linking to external vocabularies was subjectively experienced as easy by all users, this difference between test groups A and B, indicates that a significant component of effort and time consumed was actually spent deciding to which properties and classes to link with the CCK fields, i.e., which ontologies should be reused.

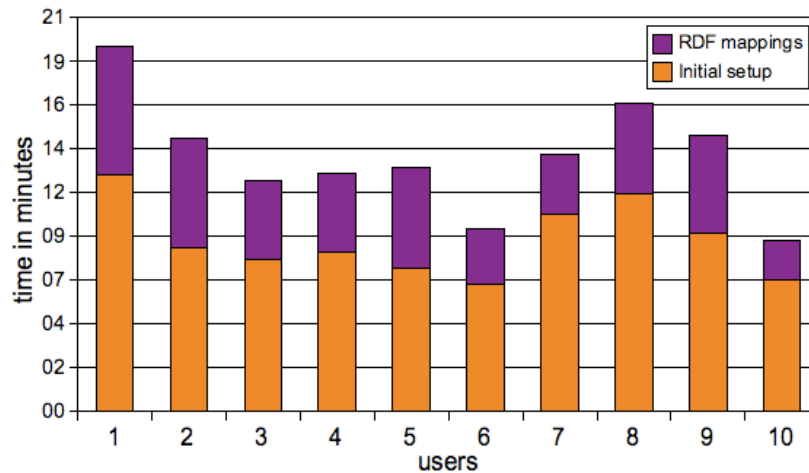


Figure 4.1: Comparison between initial setup and RDF mappings.

Summarizing, our findings show that whereas linking to external vocabularies was experienced easy by our test group, the main effort and time consumed was actually in deciding to which properties and classes to link, that is which ontologies should be reused how. Inspired by this finding we put on our agenda more investigation on how we can support non-Semantic-Web-savvy users in finding the “right” classes and properties for their needs.

4.2 Adoption

In order to make this implementation available to as many developers and site administrators as possible, we have released the RDF CCK module on [Drupal.org](http://drupal.org).¹ Since its release Nov. 2008, the RDF CCK module has – steadily increasing – reached a number of 75 deployed installations² at the time of this writing as shown in Figure 4.2. This is encouraging. Our module is currently being tested and will be deployed in the next

¹<http://drupal.org/project/rdfcck>

²according to <http://drupal.org/project/usage/rdfcck>

version of the Science Collaboration Framework (SCF) platform, a special Drupal distribution developed at the Massachusetts General Hospital and Harvard University in collaboration with DERI and other participating institutions [23].

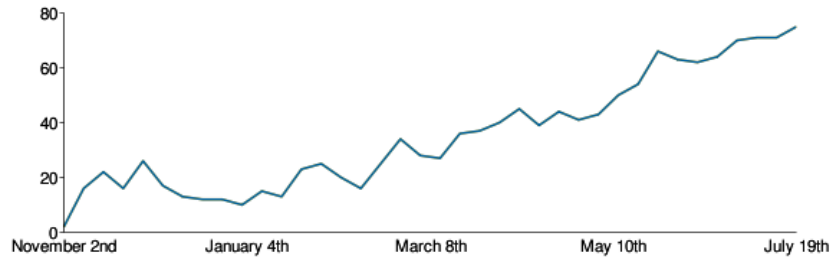


Figure 4.2: Evolution of the number of installations of RDF CCK since its release.

Chapter 5

Minimal RDF Support for Drupal Core

The solutions described in the previous chapters have been developed for Drupal 6. In order to facilitate the dissemination of our approach on the Web, and in particular within the Drupal community, we have established and implemented a set of minimal features for the Core component of the next version of Drupal (Drupal 7), see Table 5.1. They are explained in this chapter.

Feature	Minimal RDF support	Full RDF support
Site vocabulary	-	X
RDFa output	X	X
Mapping to external vocabularies	X	X
Vocabulary import	-	X
SPARQL endpoint	-	X
RDF SPARQL Proxy	-	X
Vocabulary publishing	-	X

Table 5.1: RDF related modules available for Drupal

5.1 RDF Schema proposal

The first aspect to take into consideration is the RDF classes and properties to use for mapping the core constituents of Drupal to RDF. Figure 5.1 is the RDF schema which was proposed and discussed with the Drupal community at <http://groups.drupal.org/node/9311>.

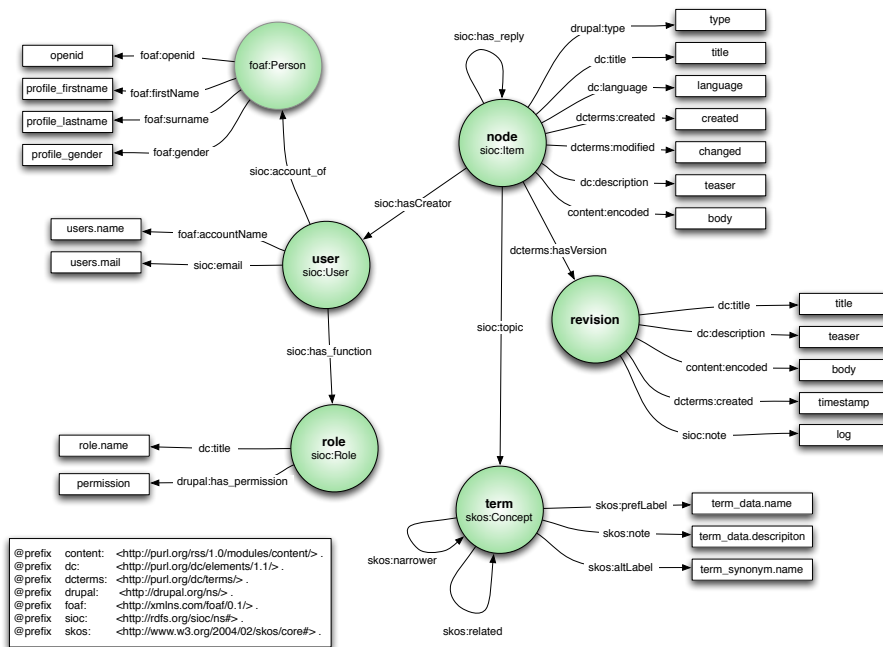


Figure 5.1: Drupal RDF Schema proposal.

The schema includes the main entity types of Drupal Core:

- **Node:** main content holder in Drupal. Each has a specific Content Type, e.g. *article*, *blog_post*, *forum_topic*, etc.
- **Term:** implemented by the taxonomy module, used to categorize content or tag nodes with the topic they cover.
- **User:** holds data about the authors or members of a site.
- **Revision:** version of a node at a given point in time. This feature is used on wikis for example.
- **Role:** each user can have a set of roles, for instance *forum moderator*, *editor*, *site administrator*. Typically, each role will be associated with a set of permissions. Examples of permissions are edit a certain type of page, revert revisions, delete content, etc.
- **Comment:** authorised users can leave comments on nodes. This entity type is not represented directly on the schema, as it is included in the concept of node (comment as node).

This Drupal core use case is one of the reasons which motivated the creation of the SIOC access module.¹

5.2 Implementation for Drupal Core 7

Based on the schema of the previous section and the work on the code sprint organised at DERI in June 2009,² several patches for Drupal core have been created.³ They are detailed below.

5.2.1 RDF Mapping Definition

While the scope of RDF in core is limited to generating RDFa for now, the design is not RDFa-only and was produced in such a way that it can be extended by other contributed modules, such as the RDF API⁴ module which could export this data in other RDF formats such as RDF/XML, N-Triples, JSON, etc. For that reason, the RDF is not encoded directly in RDFa in the HTML templates, but rather embedded in the core data structure of Drupal. While the external vocabulary import functionality is not present in core, external RDF terms are still used and asserted programmatically using an array structure. Each module defining its own data model is responsible for assigning the appropriate mappings to this model. CURIEs are used to keep the code shorter and more readable, and some system-wide common prefixes are defined in the system module. The system module is one of the main underlying modules present on all Drupal sites. A module can extend Drupal by implementing hooks.⁵ When Drupal wants to allow modules to interfere with its behavior, it will determine which modules are implementing a given hook and call these functions. Code Listing 5.1 shows an implementation of `hook_rdf_namespaces` by the system module. Note that any other module could declare more prefixes by implementing this hook, would they require new prefixes to express their RDF mappings.

Code Listing 5.1: Prefixes declaration for the system module.

```
/**
 * Implement hook_rdf_namespaces().
 */
function system_rdf_namespaces() {
  return array(
    'dc'      => 'http://purl.org/dc/terms/',
    'foaf'    => 'http://xmlns.com/foaf/0.1/',
    'rdf'     => 'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
    'sioc'    => 'http://rdfs.org/sioc/ns#',
    'xsd'     => 'http://www.w3.org/2001/XMLSchema',
  );
}
```

¹<http://rdfs.org/sioc/access>

²<http://groups.drupal.org/node/21469>

³http://drupal.org/project/issues/search/drupal?issue_tags=RDF

⁴<http://drupal.org/project/rdf>

⁵See a list of hooks Drupal 7 offers at <http://api.drupal.org/api/group/hooks/7>

While the Content Construction Kit (CCK) was a module separate from core in Drupal 6, some of its features have been integrated in the core of Drupal 7 under the code name “Field API”,⁶ which allows custom fields to be attached to Drupal objects. It also takes care of the storage, loading, editing and rendering of the fields. CCK was restricted to add fields to nodes, the Field API can attach a field to any entity type: node, user, taxonomy term, etc. Drupal 7 introduces the concept of ‘bundle’ – an extension of the concept of content types – which is a group of fields forming an entity. An example of bundle is ‘blog’ which is composed of a title, content of the post, creation date, author and her name, respectively `title`, `body`, `created`, `uid` and `name`. Code listing 5.2 shows how the blog module can define the RDF mappings of the ‘blog’ bundle by implementing the hook `rdf_mapping`.

Code Listing 5.2: Example of RDF mapping definition for the blog module.

```
/**
 * Implementation of hook_rdf_mapping().
 */
function blog_rdf_mapping() {
  return array(
    'blog' => array(
      'rdftype' => 'sioc:Post',
      'title'   => 'dc:title',
      'body'    => 'content:encoded',
      'created' => array(
        'property' => array('dc:date', 'dc:created'),
        'datatype' => 'xsd:dateTime',
        'callback' => 'date_iso8601',
      ),
      'name'    => array('dc:creator', 'foaf:name'),
      'uid'     => 'sioc:has_creator',
    )
  );
}
```

During the loading of an entity (node, user, etc.), the RDF mappings are automatically attached to the entity, so that the RDF model of this entity is carried along with its data for further processing.

5.2.2 RDFa output

At this point the RDF data has only been made available in memory, which would not serve much purpose unless it is serialized in any way. With RDFa, it becomes possible to embed RDF data in XHTML documents: as each piece of data is being rendered by Drupal’s theme layer, extra XHTML attributes are added to express the underlying RDF mappings. Based on the roadmap for RDFa in Drupal 7 which was discussed with the Drupal community,⁷ Drupal’s theme layer was adapted to allow this type of output.

First, some changes have been made to the header of the default XHTML template generated by Drupal. These are illustrated in Code Listing 5.3.

⁶<http://api.drupal.org/api/group/field/7>

⁷<http://groups.drupal.org/node/16597>

- The DOCTYPE has been adapted to match the W3C RDFa specifications [1].
- In order to be able to reuse the same CURIEs as the ones defined in the RDF mappings, the RDF namespace prefixes are serialized in the <html> tag of the XHTML output generated.
- A profile attribute is added to the <head> tag to specify the GRDDL transformation to use when extracting RDF from the RDFa document.

Code Listing 5.3: XHTML output of an RDFa enabled Drupal page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sioc="http://rdfs.org/sioc/ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<head profile="http://ns.inria.fr/grddl/rdfa/">
...

```

Some theme functions are already compatible with RDFa attributes, such as the `item_list` theme function which renders a set of items in a XHTML list. Code Listing 5.4 shows an example in PHP and its rendered RDFa XHTML code.

Code Listing 5.4: Example of a theme function for rendering a list of items.

```
$items = array();
$items[] = array('The Social Semantic Web', 'property' => 'dc:title');
$items[] = array('John Breslin', 'property' => 'dc:creator');
$items[] = array('2009-09-01', 'property' => 'dc:date');
$options[] = array('about' => 'http://example.org/book/book1');
theme('item_list', $items, 'the book', 'ul', $options);

```

```
<div class="item-list">
  <h3>the book</h3>
  <ul about="http://example.org/book/book1">
    <li property="dc:title">The Social Semantic Web</li>
    <li property="dc:creator">John Breslin</li>
    <li property="dc:date">2009-09-01</li>
  </ul>
</div>

```

Chapter 6

Neologism: Easy RDFS vocabulary publishing

While the implementation of our approach described in Chapter 3 provide a site vocabulary automatically out of the box, some administrators might want to go beyond the expressivity and the set of classes and properties generated in the site vocabulary in order to cater for their specific needs. In this chapter we introduce a tool which allows to create RDF vocabularies independently from the internal content model of a Drupal site.

*Neologism*¹ is a web-based vocabulary editor and publishing platform designed to address these issues related to Vocabulary authoring and publishing on the Web. It is currently available as an open-source project.²

6.1 Architecture

Public interface. To non-authenticated users on the Web, Neologism presents a very simple interface: a homepage that lists one or more vocabularies, and for each of them a *vocabulary page* containing some general information about the vocabulary, followed by the descriptions of all its classes and properties.

Editor. After a vocabulary maintainer logs in, additional links become visible on the vocabulary page and allow adding new terms, as well as editing of existing terms. Terms are created and edited through a web form (Figure 6.2). The form allows entry of an ID (to become part of the term's URI), label, comment, subclasses, subproperties, domain, range, disjoint classes, inverse properties, and marking a property as functional or inverse functional. Authenticated users can also create new vocabularies and modify the vocabulary metadata.

¹<http://neologism.deri.ie/>

²<http://neologism.googlecode.com/>

RDFS output, URIs and content negotiation. The URIs identifying classes and properties are always generated by appending the hash character and the term's ID to the URI of the vocabulary page. This makes sure that the vocabulary page is returned when these URIs are resolved. HTTP requests to the vocabulary page are subject to content negotiation. Web browsers will see the HTML variant shown in Figure 6.1. RDF-aware clients will receive the RDFS/OWL specification, either in RDF/XML or N3 syntax. In a nutshell, Neologism publishes standards-compliant vocabularies on the Web without requiring any additional effort on the part of vocabulary maintainers.

Implementation. Neologism is implemented in PHP as a Drupal module. Drupal reduces development time by providing many features for free, such as account management, database abstraction layer and content management. It also makes integration with a larger Drupal-based site very easy, for example to provide a news blog and discussion forum for each vocabulary built with Neologism. All data is stored in a MySQL database. RAP³ is used to serialize RDF/XML and N3. The PHP Content Negotiation library⁴ is used instead of the usual Apache rules to implement content negotiation, and Vapour⁵ was used to validate its correctness. The overview diagram is implemented using Adobe Flex and coded in ActionScript; the ObjectHandles and Tween libraries are used for animation and object handling.

6.2 User experience

The page describing a vocabulary is very similar to most online vocabularies. Figure 6.1 presents a vocabulary page when logged on the site and ready for authoring.



Figure 6.1: A vocabulary page in Neologism, as it appears to an authenticated user.

In editing mode, the user can specify the information of the class or property she wishes to create or edit as shown on Figure 6.2.

The vocabulary page provides access to a diagram that shows the vocabulary's classes and their relationships (Figure 6.3). The vocabulary maintainer can arrange

³<http://www4.wiwiw.fu-berlin.de/bizer/rdfapi/>

⁴<http://ptlis.net/source/php-content-negotiation/>

⁵<http://vapour.sourceforge.net/>

LeagueGame View Vocabulary Edit Track

Vocabulary

You can change the vocabulary this class belongs to. Only change this if you know what you are doing.

vocabulary: *

bowling

Specify the vocabulary this class belongs to.

Id: *

LeagueGame

Label: *

League game

This will be used for rdfs:label

Comment:

This will be used for rdfs:comment

Superclass:

Game

Select the superclass of this class. Your new class will have a statement expressing that it is a rdfs:subClassOf the selected class.

Disjoint with:

Bowler

Game

Container

LeagueGame

Figure 6.2: A form for editing a class.

the diagram into a sensible layout and then save its current state which will henceforth be shown to all users. Users can also move classes on the graph should they wish to view the vocabulary in a different manner, or explore it in a specific way. Zooming functions are also provided.

6.3 Adoption

As for Neologism, some groups have already chosen to use Neologism on their production site to publish their RDF vocabulary. It is the case for the <http://rdfs.org/ns/> site which hosts:

Vocabulary of Interlinked Datasets (void) (<http://rdfs.org/ns/void>) The Vocabulary of Interlinked Datasets (void) is a vocabulary and a set of instructions that enables the discovery and usage of linked datasets. A linked dataset is a collection of data, published and maintained by a single provider, available as RDF on the Web, where at least some of the resources in the dataset are identified by dereferenceable URIs.

HTTP Semantics (<http://rdfs.org/ns/http-sem>) A vocabulary for capturing the semantics of HTTP regarding resources and their representations and their interactions.

RDFForms (<http://rdfs.org/ns/rdfforms>) A vocabulary that defines how to represent HTML forms and fields (input, select, etc.) and CRUD operations on forms.

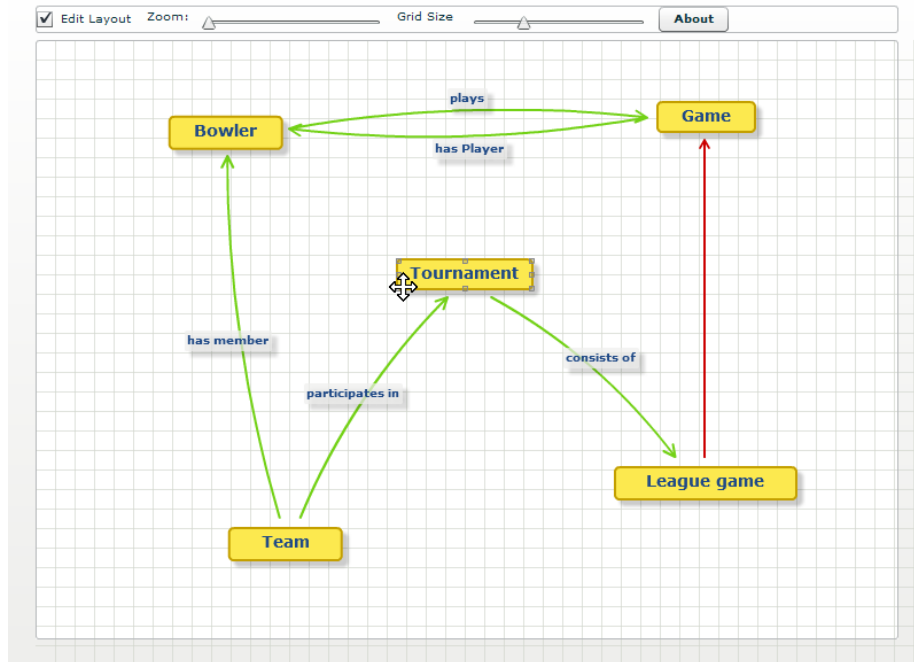


Figure 6.3: The vocabulary overview diagram.

Abstract Resource Description Vocabulary (aardv) (<http://rdfs.org/ns/aardv>)

Abstract Resource Description Vocabulary (aardv) is the least common denominator for the resource descriptor vocabularies XRD, POWDER, and void.

Statistical Core Vocabulary (SCOVO) (<http://rdfs.org/ns/scovo-plus>)

A vocabulary for representing statistical information on the Web of Data.

Chapter 7

Conclusions and Outlook

In this thesis, we have presented a number of extensions to Drupal that enable the exposure of common site content as Linked Data and likewise allow to aggregate and reuse existing RDF data and vocabularies from the Web in Drupal sites. Finally we described a lightweight vocabulary editor which allows site administrators to create new RDF vocabularies or extend their site vocabularies without the need to deploy complex tools such as Protégé [40].

Our most widely deployed module RDF CCK – available at the official Drupal site <http://drupal.org/project/rdfcck> – allows to link existing and newly deployed Drupal sites to the Web of Data with a few clicks. It auto-exports the content model of a Drupal site to an ontology that is published following common best practices for ontology publication and enables the exposure of Drupal site content as RDFa. We link to existing properties and classes from other Semantic Web vocabularies by subclass/subproperty relations following best practises and safe vocabulary reuse. However, a broader issue remains for the larger RDF community to solve: a complex part of the process of generating high-quality RDF is the task of finding and choosing good vocabularies and ontologies. There are some services that support this task, such as the search facility for commonly used vocabulary terms on the Web of Data we also presented, but they are not sufficient and this task remains difficult for non-experts. This is a major problem that the community needs to address in order for the Web of Data to succeed.

The site vocabulary uses a fragment of the available OWL constructs which is sufficient to reflect the site content model's structure in a machine-readable manner and is guaranteed to be consistent. Further planned improvements include the extended consistency checks on imported ontologies that should be done while editing and prevent the site administrator from introducing inconsistencies. For example, during the mapping process, suggested properties for a field with a CCK cardinality should only include compatible external properties, i.e. properties with a maximum cardinality greater or equal to the CCK field.

We are also aware that since Drupal local site vocabularies are likely to change as the site structure evolves, therefore they should be reused sparingly: more general, maintained, and possibly community driven ontologies are to be preferred for mapping

a site content model to RDF.

A second module – RDF SPARQL Endpoint – exposes upon installation a SPARQL endpoint on the site data without any additional configuration steps for the site administrator who wishes this feature.

Furthermore, for experienced users, we also offer a third module – RDF SPARQL Proxy – that allows to dynamically load data into the CCK content types, and displays this data using a lazy loading strategy for minimizing delays in the user experience.

In combination, all three modules above offer new possibilities to create networked Web applications and pushing further population of the Web of Data by significantly lowering entry barriers for a large user community – CMS administrators.

Next steps include the extension of RDF/OWL export for Drupal to other modules such as for instance the Taxonomy module which allows to define tag hierarchies usable in Drupal, which we plan to expose as RDF using SKOS[46]. In our radar is also the Inherit module¹ which abstracts the content types and allows to create “sub content types” inheriting the fields of the parent content type, thus reflecting the concepts of subclasses and subproperties in RDF. We will try to also lower the burden for users of the RDF SPARQL Proxy – currently only accessible to users knowledgeable in SPARQL – to really reveal the full potential of our approach to a wider audience.

In the recent years, website administrators have been encouraged to publish sitemaps.² A sitemap is an XML file which lists the URLs of a site along with some metadata. Sitemaps help search engine to crawl sites more efficiently. The Semantic Sitemap format[22] extends the existing and widely used sitemap XML format³ and allows webmasters to describe Semantic Web datasets and how to best consume it. The XML sitemap module for Drupal⁴ could be extended to embrace the Semantic Sitemap format and expose more information about the RDF data available via the modules detailed in this thesis.

The system we have implemented is a working prototype at <http://drupal.derl.ie/projectblogs/>. All the modules developed in the course of this thesis have been released under the GPL 2 licence⁵ and are available for download on Drupal.org.

For more advanced users who wish to build their own RDF vocabulary, whether it is to extend the auto generated site vocabulary by RDF CCK, or simply to design a vocabulary to answer specific needs, we have developed Neologism, a web-based vocabulary publishing system that simplifies the process of creating, publishing and maintaining RDF vocabularies by (i) instant web-based publishing, (ii) focus on a limited subset of RDFS and OWL, (iii) avoiding instance editing or browsing, and (iv) handling URI management and HTTP content negotiation. We hope that the presented system will encourage the creation of new vocabularies and thereby contribute to a generally more interesting, relevant and standards-compliant Semantic Web.

We have identified some areas in which Neologism can be improved.

¹<http://drupal.org/project/inherit>

²<http://sitemaps.org/>

³<http://www.sitemaps.org/protocol.php>

⁴<http://drupal.org/project/xmlsitemap>

⁵<http://www.gnu.org/licenses/gpl-2.0.html>

Hosted Neologism service. Currently, vocabulary maintainers must install Neologism on their own webspace. A central hosted service, which could be easily built on the Drupal platform, would remove this barrier.

Branching and revision tracking. Neologism does not yet offer revision control. Some desirable features for vocabulary revision control are: archival of all prior versions; grouping of several small edits into a single version to avoid putting the vocabulary into an inconsistent intermediate state; publishing changes as a draft before accepting them as a new version.

Plugin system. We intentionally kept the set of supported class and property annotations small to simplify the user experience, and don't support many possible further annotations, such as OWL cardinality constraints, plural and inverse labels,⁶ multilingual labels or associating Fresnel lenses [17] with classes and properties. Such additional annotations could be supported through plugins that are installed by vocabulary maintainers.

Consistency checking. Neologism doesn't check the created vocabulary for consistency. This can become an issue when a vocabulary is integrated with several external vocabularies. A solution could be the integration of an external reasoning service that performs consistency checks and is invoked through an API over the Web.

Plans for practical deployment

Harvard Medical School⁷ and DERI are collaborating on a larger use case in the course of which some of the technologies mentioned in this thesis were developed. The Science Collaboration Framework (SCF) [23] is a distributed Drupal installation launched in Beta version at various institutions working in the Biomedical domain.

Biomedical informatics provide one particularly cogent and well-researched set of use cases for the facility we have built and there are big expectations for the use of Linked Data in this domain, especially in the SCF Project. SCF is building Drupal based tools that will enable scientific collaboration and Semantic search in this area.

Mapping of graph-based metadata embodying controlled terminologies and relationships (ontologies) to CMS-managed content promises to be exceptionally useful in biomedical informatics, and more broadly in scientific communications on the web. Biomedicine, a highly descriptive, inductive and experimentally based discipline, is rife with complex terminologies. Synonyms, subsumption, and other semantic relationships in such terminologies are natural and necessary. But currently we are still limited in the power of text searching across documents and sites if the relationships and properties in the text are not computable across the elements of these terminologies (or ontologies). This requires that certain elements in the text be assigned a semantic context which is computable in the CMS. This is a use case for semantic tagging of documents, which can leverage the well-defined ontologies in this domain.

⁶<http://www.wasab.dk/morten/2004/03/label>

⁷<http://hms.harvard.edu/>

For example, from scientific papers in this domain we may extract text strings such as “nf-κB”, “nuclear factor kappa B”, or “nf-kappa-B”. By adequate thesauri or user tagging using CommonTag,⁸ all of these could actually be matched to the URI of “NFKB1”, which could be coming from the HUGO official gene names.⁹ More generally, synonyms could all resolve to a common URI represented in the Neurocommons [52] triple store, and the synonymy relationship would be represented in RDF and available at the Neurocommons SPARQL endpoint. Such extended search facilities are next on our agenda, once the simple annotation of publications authors like presented in a simplified form in this thesis is realised. Here, mapping RDF to an associated CCK generated type in Drupal will import the synonymy relationships and enable term expansion to increase search power.

Existing biomedical ontologies and database records which represent information about genes and other biomedical terms represent structured relationships, all of which can be found in RDF and drawn into our site.

This use case becomes particularly compelling when one considers that biomedical research consists of myriad sub-specialities ranging across from basic research to clinical practice, as well as incorporating divisions by biological process, organ, species, cell type, molecule, protein family, technological approach, clinical orientation, disorder, and so forth. Each of these areas can and often does have its own slightly different semantic universe and forms of discourse. The ability to intersect documents and information from and about researchers across these domains of discourse, at scale, with assistance from computers, is dependant upon our ability to leverage formal terminologies and ontologies by linking them to text in scientific communications. That is precisely the purpose of the modules described in this thesis. The experts in these domains are hardly IT or Semantic Web experts, though they are able to use easy-configurable tools for aggregating and setting up CMSs like Drupal, setting up the required modules via SCF on their site, and enter relevant data.

At the moment, RDF CCK is being deployed in the SCF Beta version, the other modules mentioned in this thesis are shortly before deployment and several of them have been co-developed or inspired by existing SCF modules such as SCF Node Proxy module, which we mentioned in the introduction.

The “infection” of emerging power-user communities such as the rapidly growing Drupal site administrator and developer groups is in our opinion a key in boosting Semantic Web technologies. We shall provide easy-to-use, unobtrusive RDF exposure in a way general enough for a variety of sites, thus potentially contributing significantly to populating the Web with high-quality RDF data. As a matter of fact, our work was listed in the recently publish technical report on Linked Data Applications [32].

In the bigger picture, the work carried out specifically for Drupal could constitute a base example for other Content Management Systems such as Typo3 or Joomla!. Such work is currently being carried out by the IKS EU project¹⁰ which manifested their interest in our approach and invited us to present this work at their first workshop.¹¹

⁸<http://commontag.org>

⁹HUGO Gene Nomenclature Committee <http://www.genenames.org/>

¹⁰<http://www.iks-project.eu/>

¹¹<http://www.iks-project.eu/requirements-workshop>

Bibliography

- [1] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton (eds.). RDFa in XHTML: Syntax and Processing, October 2008. W3C Recommendation, available at <http://www.w3.org/TR/rdfa-syntax/>.
- [2] Joan Aliprand, Julie Allen, Joe Becker, Mark Davis, Michael Everson, Asmus Freytag, John H. Jenkins, Mike Ksar, Rick McGowan, Lisa Moore, Michel Suignard, and Ken Whistler. *The Unicode standard version 3.0*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2000.
- [3] Sören Auer. Powl - a web based platform for collaborative semantic web development. In *Proc. of 1st Workshop Workshop Scripting for the Semantic Web (SFSW05), Hersonissos, Greece, May 30, 2005*, MAY 2005.
- [4] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumüller. Triplify - lightweight linked data publication from relational databases. In *Proceedings of WWW 2009*, 2009.
- [5] Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki, a tool for social, semantic collaboration. *The Semantic Web - ISWC 2006*, 4273/2006:736–749, 2006.
- [6] Reto Bachmann-Gmür. Knobot. In *2006 Jena User Conference*. 2006 Jena User Conference, 2006.
- [7] Cosmin Basca, Stéphane Corlosquet, Richard Cyganiak, Sergio Fernández, and Thomas Schandl. Neologism: Easy vocabulary publishing. In *SFSW2008 workshop*, 2008.
- [8] Dave Beckett and Tim Berners-Lee. Turtle - Terse RDF Triple Language, January 2008. W3C Team Submission. Available at <http://www.w3.org/TeamSubmission/turtle/>.
- [9] David Beckett. The Design and Implementation of the Redland RDF Application Framework. In *Proceedings of Semantic Web Workshop of the 10th International World Wide Web Conference*, Hong-Kong, China, May 2001.
- [10] T. Berners-Lee. The Giant Global Graph, November 2007. Available at <http://dig.csail.mit.edu/breadcrumbs/node/215>.

- [11] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax - RFC 3986, January 2005.
- [12] Tim Berners-Lee. Notation3 (N3) A readable RDF syntax, March 2006. Available at <http://www.w3.org/DesignIssues/Notation3>.
- [13] Tim Berners-Lee. Linked Data, Design Issues, June 2009. Available at <http://www.w3.org/DesignIssues/LinkedData.html>.
- [14] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, , and David Sheets. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In *The 3rd International Semantic Web User Interaction Workshop (SWUI06)*, 2006.
- [15] Diego Berrueta and Jon Phipps. Best Practice Recipes for Publishing RDF Vocabularies. Working Draft, W3C, 2008. W3C Working Group Note, available at <http://www.w3.org/TR/swbp-vocab-pub/>.
- [16] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee, editors. *Linked Data on the Web (LDOW2008)*, April 2008.
- [17] Christian Bizer, Ryan Lee, and Emmanuel Pietriga. Fresnel, a Browser-Independent Presentation Vocabulary for RDF. In *International Semantic Web Conference 2006*, 2006.
- [18] Harold Boley, Michael Kifer, Paula-Lavinia Pătrânjan, and Axel Polleres. Rule interchange on the web. In *Reasoning Web 2007*, volume 4636 of *Lecture Notes in Computer Science*, pages 269–309. Springer, September 2007.
- [19] Dan Brickley and R.V. Guha (eds.). RDF vocabulary description language 1.0: RDF Schema, February 2004. W3C Recommendation, available at <http://www.w3.org/TR/rdf-schema/>.
- [20] Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Technical report, 2005.
- [21] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 137–157, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.
- [22] Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 690–704. Springer, 2008.

- [23] Sudeshna Das, Lisa Girard, Tom Green, Louis Weitzman, Alister Lewis-Bowen, and Tim Clark. Building biomedical web communities using a semantically aware content management system. *Briefings in Bioinformatics*, pages 10(2):129–38, March 2009.
- [24] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference, February 2004. W3C Recommendation.
- [25] Renaud Delbru, Axel Polleres, Giovanni Tummarello, and Stefan Decker. Context dependent reasoning for semantic documents in sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*, Karlsruhe, Germany, October 2008.
- [26] Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *CIKM*, pages 652–659. ACM, 2004.
- [27] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- [28] Michael R. Genesereth. Knowledge interchange format. In *KR*, pages 599–600, 1991.
- [29] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview, June 2009. Available at <http://www.w3.org/TR/owl2-overview/>.
- [30] Andreas Harth, Jürgen Umbrich, and Stefan Decker. Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In *5th Int.l Semantic Web Conference*, Athens, GA, USA, November 2006.
- [31] Michael Hausenblas. Exploiting linked data to build web applications. *IEEE Internet Computing*, 13(4):68–73, 2009.
- [32] Michael Hausenblas. Linked data applications. Tech. Report DERI-TR-2009-07-26, DERI, July 2009.
- [33] Patrick Hayes. RDF semantics. Technical report, W3C, February 2004. W3C Recommendation.
- [34] Aidan Hogan, Andreas Harth, and Axel Polleres. SAOR: Authoritative Reasoning for the Web. In *ASWC 2008*, pages 76–90, 2008.
- [35] Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable authoritative owl reasoning for the web. *International Journal on Semantic Web and Information Systems*, 5(2), 2009.

- [36] Yuhui Jin, Stefan Decker, and Gio Wiederhold. Ontowebber: Model-driven ontology-based web site management. In Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness, editors, *SWWS*, pages 529–547, 2001.
- [37] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
- [38] Kjetil Kjernsmo and Alexandre Passant. SPARQL New Features and Rationale, July 2009. Available at <http://www.w3.org/TR/2009/WD-sparql-features-20090702/>.
- [39] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C, February 2004. W3C Recommendation.
- [40] Holger Knublauch, Ray W. Ferguson, Natalya F. Noy, and Mark A. Musen. The Protégé OWL Plugin: An open development environment for semantic web applications. *The Semantic Web - ISWC 2004*, 3298/2004:229–243, 2004.
- [41] Ora Lassila and Ralph Swick (eds.). Resource Description Framework (RDF) Model and Syntax Specification, February 1999. W3C Recommendation, available at <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>.
- [42] Danh Le-Phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *18th International World Wide Web Conference (WWW2009)*, April 2009.
- [43] Douglas B. Lenat and Ramanathan V. Guha. The evolution of cycl, the cyc representation language. *SIGART Bulletin*, 2(3):84–87, 1991.
- [44] Robert W.P. Luk, H. V. Leong, Tharam S. Dillon, Alvin T.S. Chan, W. Bruce Croft, and James Allan. A survey in indexing and searching xml documents. *Journal of the American Society for Information Science and Technology*, 53(6):415–437, 2002.
- [45] James Mayfield and Tim Finin. Information retrieval on the Semantic Web: Integrating inference and retrieval. In *SIGIR Workshop on the Semantic Web*, August 2003.
- [46] Alistair Miles and Sean Bechhofer (eds.). SKOS Simple Knowledge Organization System Reference, March 2009. W3C Candidate Recommendation, available at <http://www.w3.org/TR/2009/CR-skos-reference-20090317/>.
- [47] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: A document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1), 2008.

- [48] Eyal Oren, Renaud Delbru, Sebastian Gerke, Armin Haller, and Stefan Decker. ActiveRDF: object-oriented semantic web programming. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 817–824. ACM, 2007.
- [49] Alexandre Passant and Philippe Laublet. Meaning of a tag: A collaborative approach to bridge the gap between tagging and linked data. In *Proceedings of the Linked Data on the Web Workshop (LDOW2008) at the 17th International World Wide Web Conference (WWW2008)*, Beijing, China, April 2008.
- [50] Alexandre Passant and Philippe Laublet. Towards an interlinked semantic wiki farm. In *3rd Semantic Wiki Workshop (SemWiki2008) co-located with ESWC2008*, May 2008.
- [51] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF, January 2008. W3C Recommendation, available at <http://www.w3.org/TR/rdf-sparql-query/>.
- [52] Alan Ruttenberg, Jonathan Rees, Matthias Samwald, and M Scott Marshall. Life sciences on the semantic web: the neurocommons and beyond. *Briefings in Bioinformatics*, Epub 2009 Mar 12, pages 10(2):193–204, March 2009.
- [53] Satya Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF. Available at <http://esw.w3.org/topic/Rdb2RdfXG/StateOfTheArt>.
- [54] Sebastian Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *1st International Workshop on Semantic Technologies in Collaborative Applications (STICA'06)*, Manchester, UK, June 2006.
- [55] Ric Shreves. Open Source CMS Market Share. White paper, Water & Stone. <http://waterandstone.com/downloads/2008OpenSourceCMSMarketSurvey.pdf>.
- [56] Katharina Siorpaes and Martin Hepp. myOntology: The marriage of ontology engineering and collective intelligence. In *ESWC 2007 Workshop Bridging the Gap between Semantic Web and Web 2.0*, 2007.
- [57] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H. P. Schnurr, R. Studer, and Y. Sure. Semantic community web portals. *Computer Networks*, 33(1-6):473 – 491, 2000.
- [58] Ljiljana Stojanovic, Nenad Stojanovic, and Raphael Volz. Migrating data-intensive web sites into the semantic web. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1100–1107, New York, NY, USA, 2002. ACM.

- [59] Nikolai Toupikov, Juergen Umbrich, Renaud Delbru, Michael Hausenblas, and Giovanni Tummarello. DING! Dataset Ranking using Formal Descriptions. In *Linked Data on the Web Workshop (LDOW09)*, WWW09, Madrid, Spain, 2009.
- [60] Louis Weitzman, Alister Lewis-Bowen, and Stephen Evanchik. Using open source software to design, develop, and deploy a collaborative web site, part 1: Introduction and overview, July 2006.
- [61] Steve Williams. What is a content management system, or cms? <http://www.contentmanager.eu.com/history.htm>.

Appendix A

Namespaces

The following namespaces are used throughout this thesis.

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix rel: <http://purl.org/vocab/relationship/> .
@prefix content: <http://purl.org/rss/1.0/modules/content/> .
@prefix drupal: <http://drupal.org/ns/> .
```

Appendix B

Drupal RDF Content Construction Kit evaluation

This appendix presents the questionnaires and documentation our test users have been given to carry out the evaluation presented in Chapter 4.

B.1 Introduction

In this evaluation you will be asked to set up a simple content model on a Drupal site. In the end you will map this content model to RDF classes and properties. This guide will give you the information you need to know and the basic steps to follow in order to achieve this. We will measure the time you need to do each section of this set up. Thanks in advance for your patience. The use case is a site containing information about Editors and the Articles they wrote.

B.2 Content Construction Kit

Each item of content in Drupal is called a node. Nodes usually correspond to the pages of a site. Nodes can be created, edited and deleted by content authors.

The Content Construction Kit (CCK) is one of the most popular modules used on Drupal sites. It allows the site administrator to define types of nodes, called content types, and to define fields for each content type. Fields can be of different kinds such as plain text fields, dates, email addresses, file uploads, or references to other nodes. When defining content types and fields, the site administrator provides the following information:

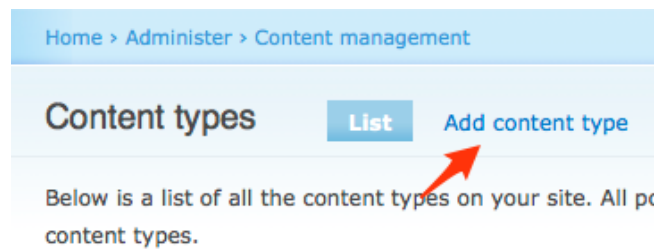
- label, ID, and description for content types and fields,
- fields can be optional or required,
- fields can have a maximum cardinality

- fields that reference other nodes can be restricted to nodes of a certain type.

B.3 Setting up the content model

B.3.1 Content types

1. In the administer section, click on the "content types" link. This will display a list of existing content types on the site.
2. Click on the tab "Add content type".



3. Enter the name of the first content type: Article.

The screenshot shows the "Add content type" form in Drupal. At the top, there are tabs: "List", "Add content type" (which is active), "Fields", and "Export". Below the tabs, there is a paragraph of text: "To create a new content type, enter the human-readable name, the machine-readable name, and the fields that are on this page. Once created, users of your site will be able to create content of this content type." Below this text is a form section titled "Identification". It contains two fields: "Name: *" with the value "Article" and "Type: *" with the value "article". Below each field is a small text box providing instructions: "The human-readable name of this content type. This text will be displayed as part of the content type. It is recommended that this name begin with a capital letter and contain only letters, numbers, and spaces." and "The machine-readable name of this content type. This text will be used for constructing the content type. This name must contain only lowercase letters, numbers, and underscores."

4. Add a description such as "An article is written by several editors".
5. Leave the rest of the form as it is, and submit it by clicking on "Save content type".
6. Create a second content type for Editor, and this time, open the fieldset "submission form settings". In the Title field label field, enter "Name" instead of "Title". Name will be the name of the editor, which will also be the name of the resource (page). Make the Body field label empty.

Submission form settings

Title field label: *

Name

Body field label:

To omit the body field for this content type, remove any text and leave this field blank.

7. Submit it by clicking on "Save content type".

B.3.2 Fields

We will now add fields to the newly created content types.

1. Click on the "manage fields" for the Article content type. This will display a list of fields for the Article content type. The greyed fields are the ones built in the system.

Add

New field

Abstract field_ abstract Text

Label Field name (a-z, 0-9, _) Type of data to store.

Existing field

- Select an existing field -

Field to share Form element to edit the data.

Select list
 Select list
 Check boxes/radio buttons
 Single on/off checkbox
 Text field
 Text area (multiple rows)
 Static images

2. Add a first field named "Abstract" (the machine name has to be lower case). It will be of type "text" and the form element will be "text area".

Article article An article is written by several editors edit manage fields delete

3. Click on "Save" at the bottom of the page. The second form allows to fine tune the field. We will leave it as it is for now, click on "Save field settings" at the bottom of the form.
4. The second field to create for the Article content type is "Editors":
 - type: Node reference
 - form element: select list
5. on the next form for advanced settings, specify that this field is required and choose "unlimited" in the Number of values drop down list.
6. For the Content types that can be referenced field, choose "Editor".
7. Set up a new field "Date of publication":

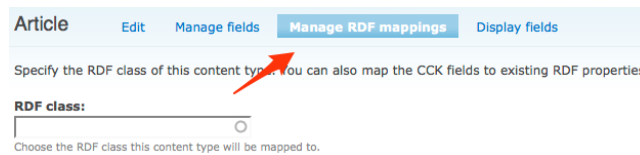
- type: Date
- form element: select list

and use the default settings.

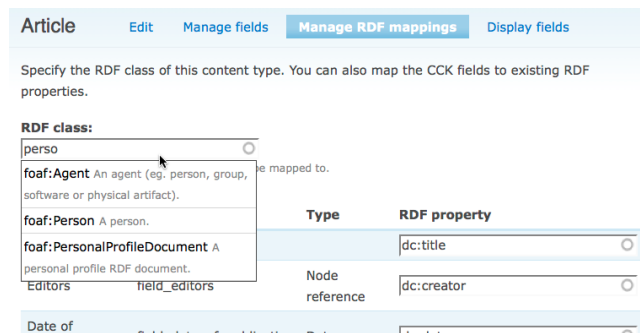
8. Change the order of the fields in the Article content type in the order that makes sense to you (put the menu settings in the end).
9. Switch to the "Editor" content type by clicking on "content types" in the administer section like in step 1.
10. Add a field "Picture" of type Image.
11. Add a field "Homepage" of type Link.
12. Add a field "Friends" of type Node Reference. For the Content types that can be referenced field, choose "Editor".

B.4 Map the Content Model to RDF terms

1. Go to the fields of the Article content type (see step 3.2.1) and click on the tab "Manage RDF mappings".



2. This page includes autocomplete fields i.e. a list of matching values will be displayed as you type. Example: typing the characters "pers" will list "foaf:Person"...The system already has several vocabularies imported: DC, SIOC, FOAF so you are encouraged to reuse these.



3. Choose an RDF class mapping for the Article content type.

4. Do the same for each fields where possible.
5. Similarly, set up the mappings for the Editor content type.

End of the evaluation. Please fill in the questionnaire. Thanks for your patience!

B.5 Drupal RDF Content Construction Kit evaluation - Questionnaire

1. Did you understand what you did and every single step of the process?

2. The goal of the evaluation is the measure how much extra effort is required to setup the mapping compared to the initial effort of setting up the content model. How did you find the mapping process compared to the initial content model setup?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Easy					
Fast					
Straight forward					

3. Would you have something to say in order to improve the user interface or the workflow to set up a content model and its mappings to RDF?

4. Would you have preferred to do the mappings during the initial set up process? (while creating each content type and field)?

5. How did you find choosing the right mapping? Was it easy or did it require some knowledge about RDF vocabularies?

6. Did you find the descriptions of the RDF terms useful for deciding what term to choose?

7. in the case of the picture field, how did you choose the right property?

- (a) you knew it by heart?
- (b) you started typing and read the description
- (c) started typing and chose the first choice
- (d) other? please precise

B.6 Drupal RDF Content Construction Kit evaluation - RDF Mappings

Use the following mappings to configure RDF CCK.

Article RDF class mapping: `sioc:Post`.

Field	Mapping
<i>Title</i>	<code>dc:title</code>
<i>Editors</i>	<code>dc:contributor</code>
<i>Date of Publication</i>	<code>dc:date</code>
<i>Abstract</i>	<code>dcterms:abstract</code>
<i>Body</i>	<code>sioc:content</code>

Editor RDF class mapping: `sioc:User`.

Field	Mapping
<i>Name</i>	<code>foaf:name</code>
<i>Picture</i>	<code>contributor</code>
<i>Homepage</i>	<code>foaf:homepage</code>
<i>Friends</i>	<code>foaf:knows</code>