Reasoning Driven Configuration of Linked Data Content Management Systems

Stuart Taylor, Nophadol Jekjantuk, Chris Mellish, and Jeff Z. Pan

dot.rural Digital Economy Hub,
University of Aberdeen,
Aberdeen AB24 5UA, UK
WWW home page: http://www.dotrural.ac.uk

Abstract The web of data has continued to expand thanks to the principles of Linked Data, increasing its impact on the web both in its depth and range of data sources. However tools allowing ordinary web users to contribute to this web of data are still lacking. In this paper we propose Linked Data CMS, an approach allowing existing web content management system (CMS) software to be configured to display a web site based on a group of ontology classes, by making use of a *configuration* to map ontological entities to CMS entities. We have implemented a prototype of our Linked Data CMS approach using the popular Drupal CMS. This approach provides the tools for semantic web application developers to rapidly develop an entire website based on linked data, while allowing ordinary web users to contribute directly to the web of data using familiar CMS tools.

Keywords: linked data, ontologies, content management systems

1 Introduction

There are many mature development platforms that are well suited to traditional data-based applications and which could benefit from integration with Linked Data [1]. In particular, nowadays web content is frequently authored by a user using some form of Content Management System (CMS), where the content may be structured or relatively unstructured (e.g. wikis and blogs). Integrating Linked Data into the CMS paradigm could not only allow users unfamiliar with the complexities of semantic web technologies to consume Linked Data [2], but also to produce it in a familiar way based on structured data already held in a CMS. This would greatly benefit the web in terms of reuse and aggregation of data held within CMSs.

In this paper, we introduce the notion of a *Linked Data Content Management System* (Linked Data CMS). A Linked Data CMS performs similar operations to a traditional CMS but whereas a traditional CMS uses a data model of content types stored in some relational database back end, a Linked Data CMS deals with performing CRUD operations on linked data held in a triple store, in the

context of a content management system. High quality Linked Data is provided in ways compatible with W3C standards such as RDF [3] and SPARQL. We make the additional assumption that the Linked Data to be managed is described by a (possibly very lightweight, but possibly also arbitrarily complex) ontology described in OWL, which is underpinned by description logics [4]. To facilitate the Linked Data CMS we then define a mapping between certain structures of the OWL classes and entities into a traditional CMS. We call this mapping *Class Based Browsing* because the ontology classes are the central entities in our mapping. Because complete and correct retrieval of information from an ontology is more complex than simple database lookup, we also introduce a process of *Reasoning Driven Configuration*, which configures the class based browsing and Linked Data CMS based on the source ontological schema. In reasoning driven configuration, an ontology reasoner, such as the highly efficient reasoner TrOWL [5], is used to automate the process of configuring the CMS.

In the paper, Section 2 briefly surveys Linked Data, ontologies and CMS, as well as the previous work that has attempted to integrate them. In Section 3 we present a formalisation of the entities that are maintained in a traditional CMS. Section 4 describes the mapping from ontology entities to CMS entities and the abstract configuration process that creates a Linked Data CMS from an ontology. Section 5 describes how the information used in the mapping can itself be represented in an ontology and the configuration achieved by standard (meta-) ontology reasoning. Finally in Section 6 we discuss the implementation of a prototype of our class based browsing and reasoning driven configuration approach using the Drupal CMS, and in Section 7 we present a proof of concept with a case study for the CURIOS project, which deals with managing cultural heritage linked data using the Drupal Linked Data CMS.

2 Background

2.1 Linked Data

Linked Data was coined by Tim Berners-Lee [6], describing a set of conventions for publishing RDF [7] datasets on the web. The main ideas are to use HTTP URIs to refer to things, allowing these URIs to be dereferenced in order to discover further related information, in W3C standard formats such as RDF and OWL [8]. These datasets should also refer to further datasets where necessary so that other related resources can be found. This has led to a vast increase in the number of RDF *Linked Data* datasets being published on the web [9]. In 2011, the Linked Data cloud totalled approximately 31.6 billion RDF triples [10].

2.2 Ontologies

High quality Linked Open Data (data which would be given four "stars" by Berners-Lee [6]) is made available using W3C standards such as RDF. These formats use vocabularies to describe the entities in the world. Descriptions of the

vocabularies are themselves in RDF, ideally should themselves be open and can be expressed in various ways, the most general ways being the different versions of the OWL Web Ontology Language [11]. Vocabulary descriptions expressed in OWL are called *ontologies*. Since an OWL description can include not only a description of a vocabulary but also the uses of that vocabulary to state facts about individual things, in general one can think of any high quality Linked Data as being specified simply by an OWL ontology.

In this paper, we assume the use of OWL DL ontologies, which are based on ideas from the Description Logic (DL) [12] family of knowledge representation languages. An OWL DL ontology $\mathcal{O}$ consists of a set of classes $\mathbf{C}$, object properties $\mathbf{OP}$, datatype properties $\mathbf{DP}$, individuals $\mathbf{I}$, and the axioms describing the relationships between them.

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

$$\mathsf{Person} \sqcap \mathsf{PostgraduateDegree} \sqsubseteq \bot$$

("nothing can be both a Person and a PostgraduateDegree"). On the other hand, ABox axioms make statements about class and property membership, e.g.,

$$\mathsf{Person}(\mathtt{Jek}),\ \mathit{knows}(\mathtt{Stuart}, \mathtt{Jek})$$

("Jek is a Person and Stuart knows Jek"). In the following we consider an OWL DL ontology as a tuple $\mathcal{O} = \langle \mathbf{C}, \mathbf{OP}, \mathbf{DP}, \mathbf{I}, \mathcal{A}, \mathcal{T} \rangle$; we use these names to refer to the sets of ontology entities throughout the remainder of the paper.

2.3 Content Management Systems

In this paper we focus solely on Web Content Management Systems. A Web Content Management System is a system that allows website maintainers and contributors to manage the content of a website via a central graphical user interface, without relying on any significant knowledge of the underlying technologies involved in running the website.

In addition to managing page content, a CMS can typically model various *content types*. These content types define specialisations of a web page, e.g., a content type for a blog post will require some additional attributes such as author and a topic or category. Most modern CMSs allow the user to define custom content types, which can consist of the page structure, page attributes, layout and style.

2.4 Semantic Web Content Management

There has been much interest in integrating CMS and Semantic Web technologies. Semantic MediaWiki (SMW) [13] is a semantic extension of MediaWiki[1],

[1] MediaWiki is a popular wiki-engine used throughout the web, most notably it is used by `http://www.wikipedia.org/`

which allows users to semantically annotate wiki mark-up in the content of wiki articles, using OWL classes and properties. This approach allows the content of the wiki to be queried using SPARQL and reused by other semantic web tools, however it does not support presenting existing Linked Data within the wiki, and therefore is limited to managing the Linked Data (OWL assertions) generated only from its own wiki articles.

The BBC's Dynamic Semantic Publishing (DSP) architecture [14] was designed to allow BBC journalists to annotate news articles[2] with metadata from a domain ontology. DSP both consumes and produces Linked Data, however the approach is geared towards producing articles with annotations to a fixed ontology, or one that is managed outside of the CMS.

Semantic Drupal[3] [15] is of particular interest, since it provides a platform to combine Drupal with the Semantic Web. The Drupal Semantic Web Group[4] focus on the integration between Drupal and Semantic Web technologies such as RDF, RDFa and SPARQL, and have successfully produced a number of Drupal extensions addressing these areas. Semantic Drupal allows site administrators to export data from *nodes* as RDF. In this approach, *content types* and *fields* are given mappings to specified classes and properties. In general however this approach does not allow management of existing Linked Data with the Drupal CMS, since it is intended only to export Drupal data to Linked Data and import existing Linked Data to Drupal's database.

Some of the limitations of consuming Linked Data in Content Management Systems are overcome by Clark [16] with the SPARQL Views Drupal module. This approach essentially adds a SPARQL query builder to the powerful Views module[5]. This module allows site administrators to specify a view of a set of fields via a GUI. This specification is then used to generate the appropriate SPARQL query to instantiate the view and therefore not relying on any SPARQL knowledge by the user. The strength of SPARQL Views is in its utilisation of the Views module's powerful GUI building capabilities, allowing users to building complex GUIs without requiring any web programming knowledge. However, this approach is intended to be read only, i.e., views only display the results from SPARQL queries, but do not allow the corresponding SPARQL endpoint to be updated. Another limitation of this approach, is that when the number of required SPARQL Views resource types increases, e.g., when building views for a large set of OWL classes (where each class would typically be specified as an individual resource type) the task of initially building and maintaining the set resource types and associated views becomes a serious burden for the user. For example, an ontology with 10 classes, each with 10 properties would correspond

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

[3] A series of guides for configuring Drupal 7 as "Semantic Drupal" can be found at `http://semantic-drupal.com/`

[4] The Drupal Semantic Web Group can be found at `http://groups.drupal.org/semantic-web`

[5] Drupal Views module: `http://drupal.org/project/views`

to 10 resource types, 100 fields with 100 RDF mappings and at least one single view for each resource type.

The approaches presented in this section go a long way to integrating the advantages of modern Content Management Systems with Linked Data. However so far no approach for general Linked Data content management exists. In this paper we propose an extension to existing content management systems which allows the full management of ontology instances data; i.e., create, retrieve, update, delete. We propose to automate the process of configuring the CMS based on the target ontology schema.

3 Formalisation of CMS Entities

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

Definition 1. *A **CMS** is a tuple $\mathcal{S} = \langle T, F, R, P \rangle$ where T is set of page templates, which define the structure of particular types of page in the CMS (i.e., sets of fields and relationships); F is a set of fields, which are used to display literal values; R is a set of relationships, which define links to another pages in the CMS; P is a set of page instances, which use a page template to display some specific information.*

Definition 2. *A field is a single $f = \langle f_{lab} \rangle \in F$, where f_{lab} is a human readable label for the field.*

Definition 3. *A relationship is a pair $r = \langle r_{lab}, r_T \rangle \in R$, where r_{lab} is the relationship label; $r_T \in T$ is a template for the pages for the objects of the relationship.*

Definition 4. *A page template for a CMS $\langle T, F, R, P \rangle$ is a tuple*

$$t = \langle t_{lab}, t_F, t_R \rangle \in T$$

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

A page template consists of a set of fields, and a set of relationships. It is used as a template for creating page instances in the CMS (i.e., a page instance has the same fields and relationships as its page template). Page templates also contain the information on how to render page instances, however we don't include the display concerns this paper. A page instance is an instantiated page template, for a particular set of field and relationship values:

Definition 5. *A page instance for a CMS $\langle T, F, R, P \rangle$ is a tuple*

$$p = \langle p_{lab}, p_T, p_F, p_R \rangle \in P, \quad where$$

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

3.1 CMS Entity Example

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

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

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

The page template $person$ is defined using the fields and relationships in $\mathcal{S}$, and the page instance may would be populated with the data about May:

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

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

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

4 Linked Data Content Management

4.1 Class Based Browsing

Our approach focuses on browsing ontological entities in a CMS, based on ontology classes. The idea is that a particular set of classes from an ontology's class hierarchy is selected to provide *views* of their instances in the CMS. The Linked Data CMS configuration (which we refer to as *the configuration*) is a set of these classes, along with some metadata to allow the Linked Data CMS to configure the CMS entities in $\mathcal{S}$ described in the previous section. We call the combination of these classes and associated metadata *browsing classes*.

Definition 6. *A Linked Data CMS configuration for an ontology $\mathcal{O}$ with components $\langle C, OP, DP, I, \mathcal{A}, \mathcal{T}\rangle$ is a pair $\mathcal{C} = \langle B, lab\rangle$, where B is a set of browsing classes and $lab : (C \cup OP \cup DP \cup I) \rightarrow String$ is a function that maps ontology entities to human-readable labels.*

Definition 7. *A browsing class in a configuration $\langle B, lab\rangle$ for the ontology $\mathcal{O} = \langle C, OP, DP, I, \mathcal{A}, \mathcal{T}\rangle$ is a tuple*

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

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

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

4.2 Linked Data CMS mapping

Given an ontology $\mathcal{O}$ and a configuration $\mathcal{C} = \langle B, lab\rangle$ for $\mathcal{O}$, the algorithm in Figure 1 can be used to construct a CMS. Because the definition of a Linked Data CMS mapping makes use of the notion of entailment ("$\models$") from $\mathcal{O}$, ontology reasoning is required to validate a potential configuration (test that all the conditions are satisfied). We address the issue of validation in section 5.2 below.

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

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

Inputs: An ontology $\mathcal{O}$ and a configuration $\mathcal{C} = \langle B, lab \rangle$ for $\mathcal{O}$
Output: A CMS $\langle T, F, R, P \rangle$

$Temp[]$: A hash table from concepts to templates, initially empty
$Page[]$: A hash table from individuals to pages, initially empty
r: a relationship
T, F, R, P :- sets of templates, fields, relationships and pages

$F := \{\langle lab(p) \rangle : b \in B, p \in b_{DP}\}$;
$R := \phi$;
$T := \phi$;
// Initialise template structures
For each $b \in B$
    $Temp[b_C] := \langle lab(b_C), \{\langle lab(p) \rangle : p \in b_{DP}\}, \phi \rangle$;
// Finalise templates and relationships
For each $b \in B$
    For each $p \in b_{OP}$
        For each $b' \in B$
            If $b'_C = b_\omega(p)$ then
                $r := \langle lab(p), Temp[b'_C] \rangle$;
                $R := R \cup \{r\}$;
                $Temp[b_C]_R := Temp[b_C]_R \cup \{r\}$;
        $T := T \cup \{Temp[b_C]\}$;
// Initialise page strructures
$P := \phi$;
For each $b \in B$
    For each a such that $\mathcal{O} \models b_C(a)$
        $Page[a] := \langle lab(a), Temp[b_C],$
            $\{\langle lab(p), v \rangle : p \in b_{DP}, \mathcal{O} \models p(a, v)\}, \phi \rangle$;
// Connect pages
For each $b \in B$
    For each a such that $\mathcal{O} \models b_C(a)$
        For each $p \in b_{OP}$
            For each v such that $\mathcal{O} \models p(a, v)$
                $r := \langle lab(p), Temp[b_\omega(p)] \rangle$;
                $Page[a]_R := Page[a]_R \cup \{\langle r, Page[v] \rangle\}$;
        $P := P \cup \{Page[a]\}$;
Return $\langle T, F, R, P \rangle$;

Figure 1. Linked Data CMS Mapping Algorithm

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

Definition 8. *A Linked Data CMS is the combination of an ontology $\mathcal{O}$, a configuration $\mathcal{C} = \langle B, lab \rangle$ and the resulting CMS $\mathcal{S} = \langle T, F, R, P \rangle$ once the Linked Data CMS mapping has been applied.*

4.3 Class Based Browsing Example

We now revisit the CMS entity example presented in Section 3.1 and show how this CMS structure can be created from an example ontology $\mathcal{O}$ using our Linked Data CMS approach.

The ontology $\mathcal{O}$ consists of:

$$\mathsf{Person}(\mathsf{may});\ \mathsf{name}(\mathsf{may}, \text{“May”});\ \mathsf{dob}(\mathsf{may}, \text{“1972-01-05”});$$
$$\mathsf{Person}(\mathsf{pam});\ \mathsf{name}(\mathsf{pam}, \text{“Pam”});\ \mathsf{parentOf}(\mathsf{may}, \mathsf{pam});$$
$$\mathsf{Person}(\mathsf{chris});\ \mathsf{name}(\mathsf{chris}, \text{“Chris”});\ \mathsf{parentOf}(\mathsf{may}, \mathsf{chris});$$
$$\text{rdfs:label}(\mathsf{may}, \text{may});\ \text{rdfs:label}(\mathsf{pam}, \text{pam});\ \text{rdfs:label}(\mathsf{chris}, \text{chris}).$$

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

We can define a configuration $\mathcal{C} = \langle B, lab \rangle$ as follows.

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

Although these labels are not the same as those used in Section 3.1, $lab(x)$ could easily be defined to produce labels based on the type of x, e.g., if x is a class, then $lab(x) = concat(l, \text{“ Page”}) : \mathcal{O} \models \text{rdfs:label}(x, l)$; where *concat* is the string concatenation function. However, for ease of presentation we use the simpler definition of $lab(x)$ in this example.

Now the Linked Data CMS mapping is applied to $\mathcal{C}$ to produce a CMS $\mathcal{S}$ as follows. First the basic CMS structure and fields are created:

$$\mathcal{S} = \langle T, \{name, dob\}, R, P \rangle;$$
$$name = \langle \text{name}, \text{“name”} \rangle; \quad dob = \langle \text{dob}, \text{“dob”} \rangle;$$

A page template is then created for each browsing class in B and the relationships are created to connect these together:

$$person = \langle \text{“Person”}, \{name, dob\}, \{parentOf\} \rangle;$$
$$T = \{person\};$$
$$parentOf = \langle \text{“parentOf”}, person \rangle;$$
$$R = \{parentOf\}$$

Then for each new page template, a page instance is created for each individual in the ontology that has a corresponding browsing class:

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

In this example we have created the CMS entities in $\mathcal{S}$ to reproduce the CMS structure described in Section 3.1, based on the ontology $\mathcal{O}$ and configuration $\mathcal{C}$. This example illustrates that given an ontology and a fairly simple configuration, a significant CMS can be created in a straight forward manner. The ontology in this example was small (since we only used three individuals) however using our approach we can increase the number of page instances at no extra cost to the user, since the complexity of defining a configuration is only related to the number of classes chosen for display.

5 Representing the Browsing Classes

In the previous section, we described the configuration, the central part of the Linked Data CMS, as a formal entity. This entity is referred to by the mapping algorithm which, using ontology reasoning as needed, constructs the CMS from the information in the ontology. In this section, we propose the use of a meta ontology to represent the configuration. This has the advantage that validation of the configuration can be done by standard meta ontology reasoning.

OWL 2 supports a very basic, but decidable, approach to metamodeling called “punning”, which means that the sets of names for classes, properties and individuals do not have to be disjoint. Punning allows one to, for instance, treat `:Eagle` as separately denoting both a class of Birds and an individual of the class `:Species`. This can be quite useful for some sorts of modelling. However, there is no logical relation between classes, individuals or properties with the same name. The semantics of punning is based on contextual semantics [17]. There are some restrictions on punning - the same identifier (IRI) cannot be use to denote both a class and a datatype property, also datatype properties and object properties cannot have the same name.

5.1 The Linked Data CMS Using a Meta Ontology

A Linked Data CMS consists of a configuration meta-ontology[6] O_{config} which is applied to a domain ontology O_{domain} in order to map entities in the domain

[6] The meta-ontology schema can be found at: `http://www.abdn.ac.uk/~csc363/ldcms/ldcms.owl`

ontology to entities in a CMS, such as Drupal. O_{config} has as individuals a set of classes **C**, object properties **OP** and datatype properties **DP** from O_{domain}; and a set of browsing classes **B**. Each browsing class $b \in B$ is an ontology individual in the meta-ontology describing how a particular base class **bc** (OWL class in the domain ontology) should be rendered by the CMS.

The following example shows part of O_{config} for a configuration with a single browsing class in Turtle syntax:

```
:PersonPage rdf:type cms:BrowsingClass ;
  rdfs:label 'Person Page' ;
  cms:baseClass domain:Person ;
  cms:relationship [
    rdf:type cms:Relationship ;
    rdfs:label 'Child Of' ;
    cms:target domain:Person ;
    cms:property domain:childOf ] ;
  cms:relationship [
    rdf:type cms:Relationship ;
    rdfs:label 'Lived At' ;
    cms:target domain:Residence ;
    cms:property domain:livedAt ] ;
  cms:field [
    rdf:type cms:Field ;
    rdfs:label 'Full Name' ;
    cms:property domain:name ] .
```

5.2 Validating the Linked Data CMS

Given the definition of browsing class in Section 4.1, a configuration such as the above must satisfy a number of constraints. For instance, each relationship target must be a base class in the configuration and the base class should include the range of the relationship property. In the above, we need, for instance, to have:

$$O_{domain} \models range(livedAt) \sqsubseteq Residence$$

A browsing class can be validated using the following SPARQL query where **bc** is the browsing class to be validated. If the ASK query returns false, then the browsing class is invalid.

```
# Validate relationship against domain ontology.
prefix cms: <http://www.abdn.ac.uk/~csc363/ldcms/ldcms.owl#>
ASK {
  # Query configuration meta-ontology.
  SERVICE <http://sparql.example.org:3030/config/query> {
   <bc> a cms:BrowsingClass ;
      cms:relationship [
        cms:property ?P ;
```

```
        cms:target ?C ].
    [] a cms:BrowsingClass ;
      cms:baseClass ?C .
  }
  # Query domain ontology.
  SERVICE <http://sparql.example.org:3030/dataset/query> {
   ?P rdfs:range [ rdfs:subClassOf ?C ].
  }
}
```

In this query a SPARQL 1.1 Federated Query [18] is used to query O_{config} hosted at the SPARQL endpoint `http://sparql.example.org:3030/config/query` and O_{domain} at the SPARQL endpoint `http://sparql.example.org:3030/dataset/query`. The query against O_{config} makes use of the entailments of O_{domain} when validating the relationship target in specified in the browsing class `<bc>`.

6 Drupal Linked Data CMS

We have implemented our Linked Data CMS approach using the Drupal platform. Drupal 7 was chosen due to having RDF support built into the core system and many contributed modules. Our Linked Data CMS approach is implemented as a Drupal module which builds on a number of existing semantic web based modules. In this section we provide an overview on how our approach has been implemented within Drupal.

In our Drupal implementation we allow the user to specify the configuration in much the same manner as described in Section 4.1. The configuration is used to create entities in Drupal using the Views, Panels and SPARQL Views modules to represent the page templates. At runtime the page instances are generated on the fly by the SPARQL Views module (see Section 2.4). We configure Drupal with three main types of page template: (i) the *listings view*, which allows users to browse and search ontology individuals; (ii) the *details view*, which allows users to view all the fields and relationships for a particular individual (Figure 2); (iii) the *update view*, which allows users to create, update and delete individuals in the ontology.

The SPARQL Views module allows users to map Drupal fields to RDF predicates and then build a view based on those fields. The field mappings are grouped into SPARQL Views Resource Types, each intended to represent a particular type of RDF resource. In the background SPARQL Views generates a SPARQL query based on the RDF mapping and Views specification. The view can be configured by users selecting the appropriate *fields*, *filters* (usually based on URL parameters), *relationships* to other SPARQL Views resources and display configuration.

Whereas maintaining a set of SPARQL Views by hand can be very complex (section 2.4), in essence our Linked Data CMS automatically configures a set of SPARQL Views Resources and Views based on an OWL ontology and user

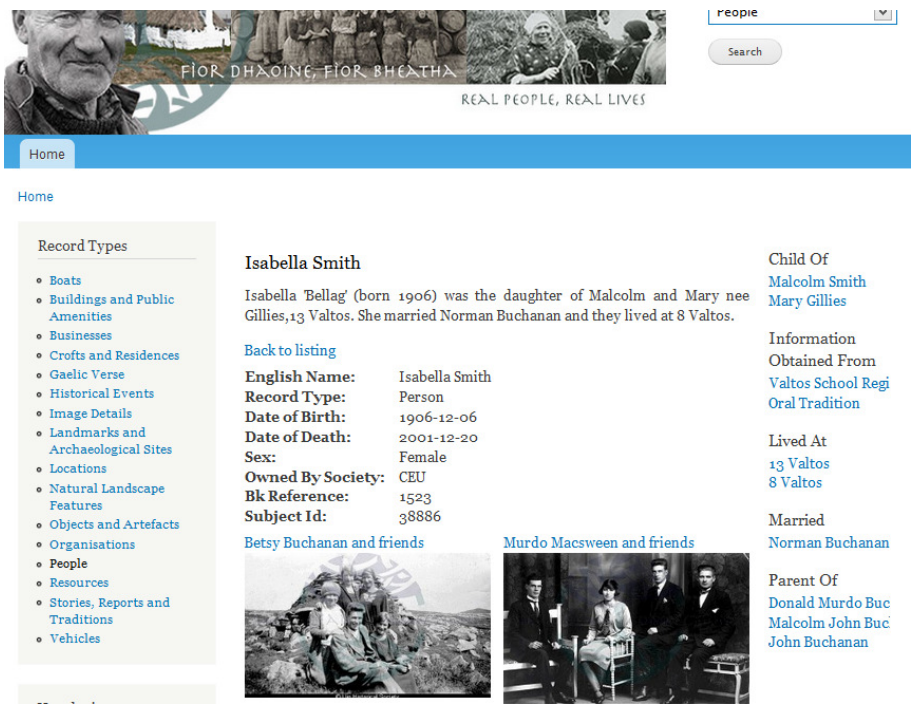

Figure 2. Drupal Linked Data CMS: Details View

specification of how that ontology should be represented in Drupal; along with an additional set of pages allowing those resources to be maintained (create / update / delete). The browsing classes identified by the user, along with their associated datatype and object properties are mapped to resource types, fields and relationships respectively. A default SPARQL Views view is then created for each browsing class (listings and details views), this view can then be modified by the user using the standard Views interface.

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

Drupal site administrators can also maintain the Drupal Linked Data CMS generated by the configuration in the same way as a regular Drupal site. Specifically once the configuration mapping has been applied, site administrators are

free to change the labels, fields, RDF mappings, page layouts and so on using the administration interface provided by the various Drupal modules.

7 Proof of Concept: Cultural Heritage Linked Data

We have used our Linked Data CMS approach to build a system that manages a repository of cultural heritage linked data. The Hebridean Connections cultural repository is a repository about people, their relationships, their occupations, the places they have lived, events they have been involved in and even the historical artefacts they have interacted with. The repository was originally a database of information collected by several historical societies in the Western Isles of Scotland. It has been recreated as an OWL ontology by the CURIOS project [19] at dot.rural in the University of Aberdeen[7].

7.1 Cultural Heritage Ontology

As an OWL ontology, the repository consists of approximately 32,000 individuals, 520 classes, 250 object properties, 55 datatype properties. The ontology schema uses an OWL 2 RL [20] level of OWL expressivity, where most of the expressive power is used to express relationships between object properties.

This ontology makes use of domain, range, functional, reflexive, symmetric and transitive property axioms, while having a relatively simple set of atomic classes. Using our reasoning driven configuration, the Linked Data CMS has been configured with 16 browsing classes, with 106 datatype and object properties from the ontology. The browsing classes contain a total of 211 field assignments (datatype properties) and 118 relationship assignments (object properties)[8]. The expressivity of the Hebridean Connections fits within the OWL 2 RL profile, allowing for efficient runtime reasoning performance.

The browsing classes used for the ontology have been selected at a fairly high level close to the top of the classification hierarchy. Some examples of the browsing classes are: Person, Occupation, Residence, Business; each class subsumes a number of more specific classes which we have also used to configure search filters in the CMS. The browsing classes and their sub-classes appear in the ranges of the object properties used as relationships in the Linked Data CMS.

In this case study we use the Fuseki SPARQL server [21], since it can be integrated with Jena API reasoners and supports SPARQL 1.1 [18]. We use SPARQL 1.1 for all communication with the endpoint (query and update services). This allows the selection of any SPARQL 1.1 compatible endpoint for the system.

8 Conclusion

In this paper, we have presented a general approach to automatically setting up a standard web content management system to manage semantic web data based

[7] CURIOS Project Homepage: `http://www.dotrural.ac.uk/curios/`

[8] An *assignment* is an occurrence of a field or relationship being used in a browsing class

on an OWL ontology and a user specification of the views of the ontology to be presented (the *configuration*). The entities generated by the mapping algorithm (Section 4.2) rely on ontology entailments [4], which are used to automatically infer additional relationships between pages in the CMS. The validation of the user specification and the generation of the CMS are both *reasoning-driven*, in that they make essential use of ontology reasoning [2].

The approach relies on using a particular method of managing ontology data in the content management system, called class based browsing. Using our approach the gap between linked data and popular the CMS tools currently in use on the web is greatly reduced, allowing ordinary web users to contribute directly to the semantic web using familiar CMS tools. We showed how our Linked Data CMS approach can be implemented in the Drupal CMS by building on top of some popular Drupal modules. An additional advantage of this implementation is that once the class based browsing has been configured in Drupal, users can still use all of the standard Drupal tools to customise the Linked Data CMS. Finally, we presented an instantiation of the Linked Data CMS with cultural heritage data. As for future work, we plan to investigate how to consider other user requirements [22] of ontology enabled software [23], when setting up content management systems.

Acknowledgement This work is partially supported by the RCUK dot.rural Digital Economic Hub and the EU K-Drive (286348) projects.

References

1. Hogan, A., Pan, J.Z., Polleres, A., Ren, Y.: Scalable OWL 2 Reasoning for Linked Data. In: Reasoning Web. Semantic Technologies for the Web of Data. Lecture Notes in Computer Science 6848 Springer, ISBN 978-3-642-23031-8. (2011)
2. Pan, J.Z., Thomas, E., Ren, Y., Taylor., S.: Tractable Fuzzy and Crisp Reasoning in Ontology Applications. In: IEEE Computational Intelligence Magazine. (2012)
3. Heino, N., Pan., J.Z.: RDFS Reasoning on Massively Parallel Hardware. In: Proc. of the 11th International Semantic Web Conference (ISWC2012). (2012)
4. Pan, J.Z.: Description Logics: Reasoning Support for the Semantic Web. PhD thesis, School of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK (2004)
5. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 Reasoning Infrastructure. In: the Proc. of the Extended Semantic Web Conference (ESWC2010). (2010)
6. Berners-Lee, T.: Linked-data design issues. W3C design issue document (June 2009) `http://www.w3.org/DesignIssue/LinkedData.html`.
7. Manola, F., Miller, E.: RDF Primer (2004) W3C Recommendation, `http://www.w3.org/TR/rdf-primer/`.
8. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S., eds.: OWL 2 Web Ontology Language: Primer. W3C Recommendation (2009) Available at `http://www.w3.org/TR/owl2-primer/`.
9. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. **5**(3) (2009) 1–22
10. Cyganiak, R.: The linking open data cloud diagram. `http://richard.cyganiak.de/2007/10/lod/` (2011) Accessed: 2012-10-16.

11. Motik, B., Patel-Schneider, P.F., Grau, B.C.: OWL 2 Web Ontology Language: Direct Semantics (2009) W3C Recommendation, http://www.w3.org/TR/owl2-direct-semantics/.
12. Baader, F., Horrocks, I., Sattler, U.: Description Logics for the Semantic Web. KI – Künstliche Intelligenz **16**(4) (2002) 57–59
13. Krötzsch, M., Vrandecic, D., Völkel, M.: Semantic MediaWiki. In Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L., eds.: International Semantic Web Conference. Volume 4273 of Lecture Notes in Computer Science., Springer (2006) 935–942
14. Rayfield, J.: Dynamic semantic publishing. In Maass, W., Kowatsch, T., eds.: Semantic Technologies in Content Management Systems. Springer (2012) 49–64
15. Corlosquet, S., Delbru, R., Clark, T., Polleres, A., Decker, S.: Produce and consume Linked Data with Drupal! In Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K., eds.: International Semantic Web Conference. Volume 5823 of Lecture Notes in Computer Science., Springer (2009) 763–778
16. Clark, L.: SPARQL Views: A Visual SPARQL Query Builder for Drupal. In Polleres, A., Chen, H., eds.: ISWC Posters&Demos. Volume 658 of CEUR Workshop Proceedings., CEUR-WS.org (2010)
17. Motik, B.: On the Properties of Metamodeling in OWL. Journal of Logic and Computation **17**(4) (2007) 617–637
18. SPARQL: SPARQL 1.1 overview (2012) W3C Working Draft, http://www.w3.org/TR/sparql11-overview/.
19. Mellish, C., Wallace, C., Tait, E., Hunter, C., Macleod, M.: Can digital technologies increase engagement with community history? In: Digital Engagement 2011. (2011)
20. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language – Profiles. Technical report, W3C (2009)
21. Seaborne, A.: Fuseki: serving RDF data over HTTP. http://jena.apache.org/documentation/serving_data/ (2011) Accessed: 2012-10-27.
22. Siegemund, K., Zhao, Y., Pan, J.Z., Assmann., U.: Measure Software Requirement Specifications by Ontology Reasoning. In: Proc. of the 8th International Workshop on Semantic Web Enabled Software Engineering (SWESE2012). (2012)
23. Pan, J.Z., Staab, S., Amann, U., Ebert, J., Zhao, Y.: Ontology-Driven Software Development. Springer (ISBN: 978-3-642-31225-0) (2013)