

# Lived happily ever after: Combing a dynamic CMS front-end with Semantic Mashup Personalization features

Dimitrios A. Koutsomitropoulos, Aikaterini K. Kalou and Georgia D. Solomou

High Performance Information Systems Laboratory (HPCLab),  
Computer Engineering and Informatics Dpt., School of Engineering,  
University of Patras, Building B, 26500 Patras-Rio, Greece  
{kotsomit, kaloukat, solomou}@hpclab.ceid.upatras.gr

**Abstract.** Upfront Semantic Web applications are often lacking desirable features related to their ‘Web’ part, despite the novelties and acknowledged added-value introduced with them. On the other hand, characteristics like front-end intuitiveness, dynamic content rendering and streamlined user management have been dealt with, elaborated and improved over the course of many years in the world of traditional CMSs. Instead of reinventing the wheel, in this paper we propose an example of how these features can be successfully integrated within a Semantic Web application, by taking advantage of an existing CMS infrastructure. In particular, we present our semantic book mashup, which combines Web APIs with semantics in order to produce personalized book recommendations, and explain how this can be interweaved with Drupal. It is shown that this approach not only leaves reasoning expressiveness and effective ontology management uncompromised, but comes to their benefit.

## 1 Introduction

The proliferation of structured data on the Web, the increased need for knowledge elicitation and the requirement for accurate information management and retrieval make Semantic Web technologies a suitable match for current Web applications. However, although they can revolutionize the way today’s Web is perceived, Semantic Web applications are still a long way from unleashing their true potential within the everyday user experience with the Web.

Despite their widely accepted benefits, this kind of applications usually put too much effort in the bottoms-up construction of elaborate, knowledge intensive set-ups. They often dwell on high-end reasoning services, efficient rule processing and scalability over voluminous data, thus hardly leaving any room for traditional Web development. Even when this is achieved, it comes at a cost on semantic management of information, by e.g. relaxing expressivity requirements or introducing ‘lightweight’ reasoning. As a result the actual added-value eludes the average user, who often is found striving to meaningfully exploit such applications or is excluded in advance from their target group.

Traditional web content management systems (CMSs) are on the other hand enjoying wide popularity among web developers, possibly because they offer an up-to-

date and tailored web infrastructure and leave more room for the designer to concentrate on successful content production and delivery, rather than technical details. As they form the spearhead of Web 2.0, it might then feel natural to employ them as a basis for Semantic Web applications, but this presents a series of challenges that it is not always straightforward to overcome.

Semantic mashups are a paradigm of the so-called Web 3.0 that aggregates data coming from various online third-party resources, like Web APIs or Linked Data, and employs semantic web technologies and ideas in any part of their design, architecture, functionality or presentation levels. It might be that, of all semantic web applications, semantic mashups can benefit the most from this integration, exactly because of their kinship and origins in Web 2.0.

In this paper we therefore propose how such applications and CMSs can be integrated, by presenting Books@HPCLab, a semantic mashup application, which we purposely establish on top of the Drupal CMS. Books@HPCLab [6, 11] is a data mashup, initially developed from scratch, which offers personalization features to users searching for books from various data sources. The key concept of this mashup is that it gathers information from Amazon and Half Ebay Web APIs, enriches them with semantics and then employs OWL 2 reasoning to infer matching preferences.

The following text is organised as follows: in Section 2, we start by discussing the desirable properties of CMSs that make them suitable as a basis for developing Semantic Web applications. In Section 3, we explain how we proceeded with the actual integration and discuss how we addressed the problems arising in this process. Next, in Section 4, we illustrate the features and the functionality of our application, now completely re-engineered over Drupal, by outlining an indicative application scenario. Finally, Section 5 summarises our conclusions and future work.

## 2 CMS as a Semantic Web Infrastructure

A typical CMS generally comes with the ability to help and facilitate the user, even the non-technical one, in various ways. It always ensures a set of core features [10] such as:

- *Front-end Interface*: The developer community of all available CMSs invests significantly in the layout, appearance and structure of the content that is created and delivered by a CMS. Therefore, the content remains completely separate from the appearance. To this end, users of CMSs can select from a great variety of well-designed templates, free or not.
- *User management*: CMSs offer also considerable advantages in regard to user administration and access issues. It can be easily controlled whether users are allowed to register on a web application as well as what kind of privileges they can have, by providing access layers and defining sections of the web application as public or private. Moreover, CMSs allow for assigning roles to users so as to involve them in the workflow of web content production.
- *Dynamic content management*: Usually a CMS relies on an RDBMS to efficiently store and manage data and settings, which are then used to display page content.

So, the installation of a CMS always involves setting-up a database schema in the corresponding SQL server. The database schema actually used, varies depending on the CMS.

- *Modular design*: CMSs follow architecture styles such as *Model-View-Controller* (MVC) or *Presentation-Abstraction-Control* (PAC) that permit the organization of code in such a way that business logic and data presentation remain separate. This enables the integration of small, standalone applications, called modules, which accomplish a wide variety of tasks. These artifacts can be easily and simply installed / uninstalled and enabled / disabled in the core of CMSs. Modularity is one of the most powerful features and the one that saves the most development effort.
- *Caching*: It is also important that most CMSs offer cache capabilities to users/developers. Thus, CMS-based web applications can have fast response times by caching frequently requested content and reducing their overhead.

Features such as these, that contemporary CMSs unsparingly offer, are exactly the ones sometimes neglected by Semantic Web applications. In the case of our work, we chose to integrate Books@HPCLab within the core of Drupal CMS [12]. Regardless of Drupal's semantic character, other significant advantages such as flexibility and scalability make it stand out from the large pool of CMSs. Besides, Drupal has been used before as a basis for offering Linked Data services [5]. Finally, Drupal can be viewed not only as a CMS, but also as a content management *framework*, by accommodating development of any type of web application.

### 3 Design and Integration

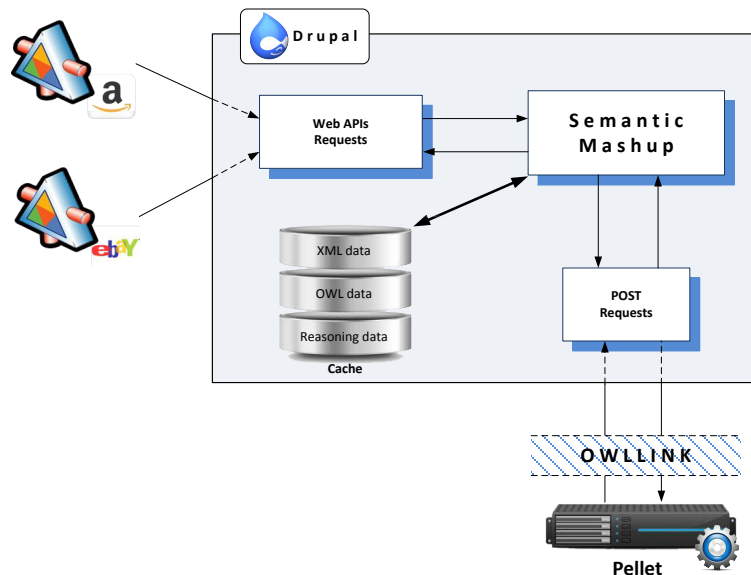
In order to re-engineer our semantic mashup on top of Drupal, we encountered a series of challenges, originating from the fact that CMSs are usually not semantics-aware. Although latest versions of Drupal offer some inherent semantic features [4], in our implementation we needed to put a strong focus on reasoning as well as arbitrary ontology management, which is beyond Drupal's state-of-the-art (or any other CMS's for that matter).

On the other hand, the modular philosophy of a CMS allows us to extend its capabilities with ready-made modules and to reuse them for our purposes. To this end, we utilize the *AmazonStore* module<sup>1</sup> that offers an attractive wrapper and front-end for the Amazon Web API. We have extended this module so as to include support for eBay as well. We also make use of the *WebForm* module<sup>2</sup>, which supports form-based data collection and that we use as the initiating point for constructing user profiles.

Next, we describe some issues we had to put up with and how we addressed each one of them. Figure 1 presents the overall design of the application, its interaction with the reasoner and its relative placement within the CMS framework.

<sup>1</sup> [http://drupal.org/project/amazon\\_store](http://drupal.org/project/amazon_store)

<sup>2</sup> <http://drupal.org/project/webform>



**Fig. 1.** Architecture and communication flow for integrating Semantic Mashup with Drupal.

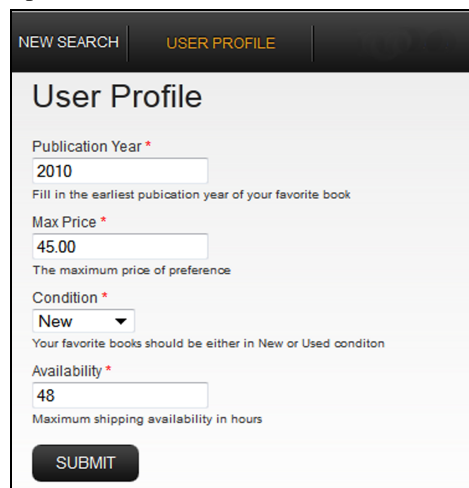
- *User profile construction and maintenance:* User preferences are collected using web forms and are stored in a relational database. In order to perform reasoning however, these preferences have to be translated into semantically rich expressions, which form the ontological profile of each user. In addition, these profiles need to be updated, every time the user changes his preferences. In our case, we maintain user preferences in the database, but we construct the profile on-the-fly, by mapping preferences to a set of OWL 2 expressions, which we then feed to the reasoner. A unique *user id* is used to logically separate knowledge bases under the same reasoner.
- *Synchronizing relational and ontology back-ends:* As is the case with user profiles, XML data returned from bookstore Web APIs need to be transformed into the OWL word in order to enable inferences. Therefore, it is possible to end up with a constant roundtrip between XML and OWL data which cannot be efficient. More to the point, it may be possible to create profiles on the fly, since they amount to only a limited number of triples, but this does not hold for book data which can span several hundreds of assertions. To resolve this, XML and OWL data are cached separately, and the transformation happens only once, when the query is originally submitted. In this way CMSs modules can remain oblivious to the ontology data and continue to operate on their own data cache, while OWL is cached in a separate database table. This caching idea is also carried over to reasoning results, which actually improves the effective reasoning throughput. An algorithm is responsible for synchronizing between the three caches, which, apart from checking for repeating queries, additionally expunges reasoning cache whenever a user updates his profile.
- *Data linking:* Transformation of book data into OWL can be naïve, by simply rep-

licating data into another syntax. However, in a mashup situation such as this, data flow into the application from many disparate sources. Following Linked Data principles [1], we therefore maintain the context of book data and their various offers, by assigning resolvable identifiers to them. These data can then be contributed to the LOD cloud and made available to other applications in RDF format.

- *Reasoner integration:* Most OWL 2 reasoners (like, Pellet, FaCT++ and Hermit) are traditionally deployed directly in-memory and interaction is performed by means of a java-based API. Although a PHP-to-Java bridge is available<sup>3</sup>, there are many reasons why one may want to keep reasoning services logically and/or physically separated [7]. Among them, the need for interoperability and independence from the actual programming language are of particular importance for integration with a CMS. In our implementation, we use OWLlink [8] as the reasoner communication protocol of choice and its implementation, the OWLlink API [9] that helps us deploy a true 3-tier architecture. OWLlink offers a consistent way of transmitting data to and receiving responses from the most popular Semantic Web reasoners, in a REST-like manner and over HTTP. Potential communication overhead that may be introduced with this approach can be alleviated by freeing up resources as a consequence of delegating computationally hard reasoning tasks to another tier [7]. Moreover, Drupal offers us generic function implementations that can be used to wrap and construct HTTP requests.

## 4 A Semantic Mashup over Drupal

When a user visits our app for the first time, he has to register by filling a form with his username and e-mail. An administrator then enables the account and a password is sent to the user at the specified mail address.



The image shows a web form titled 'User Profile' with a dark header containing 'NEW SEARCH' and 'USER PROFILE'. The form fields are: 'Publication Year \*' with the value '2010' and a note 'Fill in the earliest publication year of your favorite book'; 'Max Price \*' with the value '45.00' and a note 'The maximum price of preference'; 'Condition \*' with a dropdown menu set to 'New' and a note 'Your favorite books should be either in New or Used condition'; and 'Availability \*' with the value '48' and a note 'Maximum shipping availability in hours'. A 'SUBMIT' button is at the bottom.

**Fig. 2.** Collecting user preferences.

<sup>3</sup> <http://php-java-bridge.sourceforge.net/pjb/>

After successful authorization, logged users can set their profile using the Web-Form module. The form fields correspond to user preferences and include: book condition (“new” or “used”), maximum book price, earliest publication year and maximum availability (Fig. 2). A user can update his profile at any time. Note also that if a user does not define preferences, the application behaves as a standard book mashup and the reasoner is never engaged.

**Fig. 3.** Results list and preference ranking (stars).

The search form and results layout are based on the AmazonStore module. This time, the ‘available offers’ pull-down list also includes eBay offers that have been mashed up and linked with Amazon book data. Original results ranking is not affected (i.e. they are listed in the order they are returned by Amazon). However, the reasoner checks how many preferences a book does match and this is expressed by a star-rating system: The more the stars that appear next to each title, the higher the book ranks in user preferences (Fig.3). Finally, clicking on a title shows the full description of the particular book.

## 5 Conclusions and Future Work

Integration of Semantic Web applications with a CMS is not always straightforward. In order to achieve a seamless alignment, a series of issues has first to be resolved, and in this paper we have indicated exactly how this can be achieved in the case of our semantic mashup. Primarily, the semantic-oblivious nature of most CMSs calls for the explicit manipulation of semantically enriched data, which can be far from trivial, especially when their robust relational back-end is to be taken advantage of. Additionally, incorporating a reasoning infrastructure needs to be carefully designed as there may be substantive trade-offs involved.

Nevertheless, by combing the best of both worlds, the developer can genuinely focus on the internals of the Semantic Web implementation and assign web content management and delivery on tried and true existing frameworks, instead of wasting time and effort. It turns out that, by investing in this integration, even the semantic aspects can benefit e.g. from data caching or reasoner delegation, thus making a virtue of necessity.

As a next step, we intend to pay a closer look at the deeper integration with relational data in a means to avoid data replication and to save storage space in the database. Although our caching approach appears to work well in practice, it is not clear whether the separate cache maintenance really compensates for on-the-fly transformations or how does it compare with virtualized graph access as in D2RQ [3]. The RESTful style of reasoner communication also allows for investigating potential alternatives with a view on scalability. For example, there is evidence that rule-based triple stores, such as OWLIM [2], can improve overall performance in our mashup scenario [11]. Therefore it is worth examining this approach, this time however from the CMS perspective. Finally, we plan to package our prototype as a totally independent CMS module, thus allowing its smooth installation and reuse by other developers.

## 6 References

1. Berrueta, D., Phipps, J. (eds.). Best Practice Recipes for Publishing RDF Vocabularies, W3C Working Group Note (2008)
2. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., & Velkov, R. OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1), 33-42 (2011)
3. Bizer, C., & Seaborne, A. D2RQ-treating non-RDF databases as virtual RDF graphs. In *Proc. of the 3rd Int. Semantic Web Conference (ISWC2004)*, p.26 (2004)
4. Bratsas, C., Bamidis, P., Dimou, A., Antoniou, I., & Ioannidis, L. Semantic CMS and Wikis as Platforms for Linked Learning. *2nd Int. Workshop on Learning and Education with the Web of Data (LiLe2012) – 24<sup>th</sup> Int. World Wide Web Conference (2012)*
5. Corlosquet, S., Delbru, R., Clark, T., Polleres, A., & Decker, S. Produce and Consume Linked Data with Drupal!. In *Proc. of the 8<sup>th</sup> Int. Semantic Web Conference (ISWC 2009)*, pp. 763-778. Springer (2009)
6. Kalou, K., Pomonis, T., Koutsomitropoulos, D., & Papatheodorou, T.S. Intelligent Book Mashup: Using Semantic Web Ontologies and Rules for User Personalisation. In *Proc. of the 4th IEEE Int. Conference on Semantic Computing (ICSC 2010) - Int. Workshop on*

- Semantic Web and Reasoning for Cultural Heritage and Digital Libraries (SWARCH-DL 2010), pp. 536-541. IEEE (2010)
7. Koutsomitropoulos, D., Solomou, G., Pomonis, T., Aggelopoulos, P., & Papatheodorou, T. S. Developing distributed reasoning-based applications for the Semantic Web. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pp. 593-598. IEEE (2010)
  8. Liebig, T., Luther, M., Noppens, O., & Wessel, M. OWLlink. *Semantic Web*, 2 (1), 23-32 (2011)
  9. Noppens, O., Luther, M., & Liebig, T. The OWLlink API-Teaching OWL Components a Common Protocol. In *Proc. of the 7th Workshop on OWL: Experiences and Directions*. Vol. 614 (2010)
  10. Patel, S.K., Rathod, V.R., Prajapati, J.B. Performance Analysis of Content Management Systems-Joomla, Drupal and WordPress. In *International Journal of Computer Applications*, 21 (4), 39-43 (2011)
  11. Solomou, G., Kalou, K., Koutsomitropoulos, D., and Papatheodorou, T.S. A Mashup Personalization Service based on Semantic Web Rules and Linked Data. In *Proc. of the 7th Int. Conference on Signal Image Technology and Internet Information Systems (SITIS 2011)*, pp. 89-96. IEEE (2011)
  12. Tomlinson, T. *Beginning Drupal 7*. Apress (2010)