

Cooking HTTP content negotiation with Vapour

Diego Berrueta¹, Sergio Fernández¹ and Iván Frade²

¹ Fundación CTIC

Gijón, Asturias, Spain

{diego.berrueta,sergio.fernandez}@fundacionctic.org

<http://www.fundacionctic.org/>

² Universidad de Oviedo

Oviedo, Asturias, Spain

ivan.frade@gmail.com

<http://www.uniovi.es/>

Abstract. The Semantic Web is built upon distributed knowledge published on the Web. But this vision cannot be implemented without some basic publishing rules to make the data readable for machines. Publication of RDF vocabularies must receive special attention due to their important role in the Semantic Web architecture. In this paper we describe a scripting web-based application that validates the compliance of a vocabulary against these publication rules. Practical experimentation allows to illustrate and to discuss some common problems in the implementation of these rules.

1 Introduction

The Semantic Web is a big container, a universal medium for data, information and knowledge exchange. However the Semantic Web is not only about putting data on the Web, there are some publishing rules. Tim Berners-Lee outlined four basic principles [2] to publish Linked Data on the Web [5]. These rules describe how URIs must be used as names for things, and how to provide useful information on these things and other related ones. Although there are guidelines to coin adequate URIs for things [11], there is still the need to provide the best representation of the information for each request depending on each kind of client agent, human or software.

Web documents are retrieved using mainly the HTTP [8] protocol. This protocol provides a mechanism known as *content negotiation*. By means of content negotiation, it is possible to serve Web content in the format or language preferred by the requester (if it is available, obviously). Using transparent content negotiation in HTTP [9] has many benefits [12], and it can be implemented using different techniques in the Apache web server, as we describe in more detail in Section 2 of this paper. Section 3 introduces a scripting application that provides help and guidance to implement correctly and to debug HTTP content negotiation. In Section 4 the compliance of some of the most used vocabularies in the Semantic Web is evaluated with respect to the publishing rules. Finally, Section 5 presents some conclusions and future work.

2 Content negotiation with Apache: Recipes

Nowadays, the Apache HTTP Server is the most used Web server³, and it provides three different approaches to implement content negotiation⁴:

Type Map: Explicit handlers are described in a file (`.var`) for each resource. The necessary configuration is quite complicated and tedious, therefore this method is hardly used.

MultiViews: Based in the MIME-type and names of the files in a directory, MultiViews serves the most appropriate file in the current directory when the requested resource does not exist. It returns an additional header (`Content-Location`) to indicate the actual location of the file. This method can be extended using the Apache module `mod_mime` to associate handlers to new file extensions. However, this solution has a quite important problem: it only works if the files exist in the same directory.

Rewrite request: Probably because the two alternatives above do not provide an easy solution, the most widely used method is one which was not specifically designed to implement content negotiation. This mechanism uses the module `mod_rewrite` in order to rewrite the request according to some ad-hoc rules. As a result, requests (for objects that are not known to be information resources) are redirected using the HTTP 303 status code, to the URI of the appropriate content depending on the format requested. Obviously, some time is lost with the extra HTTP round-trip, but it is negligible for many applications, as well as mandatory according the `httpRange-14` resolution from the TAG⁵.

There is some ongoing work by W3C on *Best Practice Recipes for Publishing RDF Vocabularies* [3], a document which contains several recipes that advice on how to publish RDF/OWL Vocabularies using `mod_rewrite`. This “cookbook” provides step-by-step instructions to publish vocabularies on the Web, and gives example configurations designed to address the most common scenarios.

However, the *Recipes* are not perfect, and there is at least one important issue to be solved⁶. Tim Berners-Lee reported that “the recipe for responding to an accept header only responds to a header which EXACTLY matches [the rule antecedent]”. For those requests which contain values for the Accept header such as `text/*` or `application/rdf+xml;q=0.01`, where wildcards or *q*-values are used, the actual representation served by the rules proposed in the *Recipes* might differ from the expected one. This is a serious problem of the *Recipes*, but it can be easily solved using a script at server-side.

³ <http://www.netcraft.com/survey/> (retrieved 13/Mar/2008)

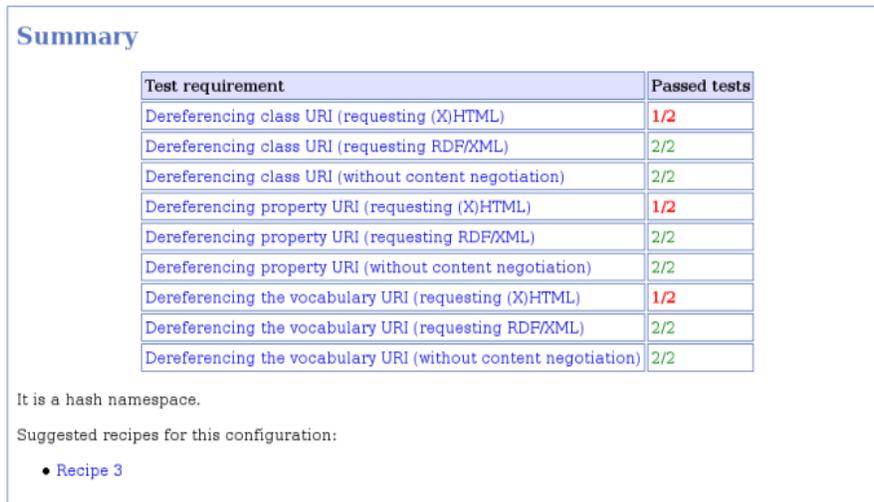
⁴ <http://httpd.apache.org/docs/2.0/content-negotiation.html>

⁵ <http://www.w3.org/2001/tag/issues.html#httpRange-14>

⁶ <http://www.w3.org/2006/07/SWD/track/issues/58> (retrieved 13/Mar/2008)

3 Vapour: a scripting approach to debug content negotiation

The previous section has shown that a correct implementation of content negotiation is not an easy task. Furthermore, manually testing an implementation is not complex, but it is long and cumbersome. Although it can be done with tools such as cURL⁷, this process is not handy, specially for intensive or repetitive tests against a vocabulary.



Summary

Test requirement	Passed tests
Dereferencing class URI (requesting (X)HTML)	1/2
Dereferencing class URI (requesting RDF/XML)	2/2
Dereferencing class URI (without content negotiation)	2/2
Dereferencing property URI (requesting (X)HTML)	1/2
Dereferencing property URI (requesting RDF/XML)	2/2
Dereferencing property URI (without content negotiation)	2/2
Dereferencing the vocabulary URI (requesting (X)HTML)	1/2
Dereferencing the vocabulary URI (requesting RDF/XML)	2/2
Dereferencing the vocabulary URI (without content negotiation)	2/2

It is a hash namespace.

Suggested recipes for this configuration:

- [Recipe 3](#)

Fig. 1. Example of a report summary made by Vapour.

In order to facilitate the task of testing the results of content negotiation on a vocabulary, we developed a web-based application called Vapour⁸. This application provides a service that makes multiple requests to a set of URIs and runs a test suite specifically designed to check the responses of the server against the best practices defined in the *Recipes*. Tests are executed against the vocabulary URI, a class URI and a property URI (the latter two can be automatically detected). Based on the input parameters, the application provides a pointer to the specific recipe, in case the user wants to learn more on how to configure the web server. Vapour stores all assertions into an in-memory RDF store, using a combination of EARL [1], HTTP Vocabulary [10] and an RDF representation of the best practices of the *Recipes*. Thus Vapour can provide the

⁷ Richard Cyganiak's explanation of how to use cURL to debug content negotiation, blog post available at: <http://dowhatimean.net/2007/02/debugging-semantic-web-sites-with-curl>

⁸ <http://vapour.sourceforge.net/>

reports both in HTML and in RDF, using content negotiation. The HTML view displays a clean and concise pass/fail report of each set of tests (Figure 1), as well as a detailed explanation of its findings that includes a graphical representation of the HTTP dialog. Needless to say, the examples included in the *Recipes* are successfully validated by Vapour.

The application is written in Python, and it uses common Python libraries such as urllib, httplib, web.py and RDFLib. Scripting languages such as Python allow an agile development of applications in a short time with little resources. Source code of Vapour is available on SourceForge⁹, and an online demo of the service is also available¹⁰.

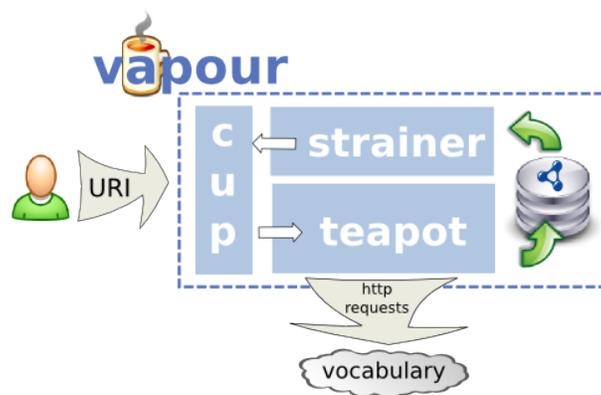


Fig. 2. High level architecture of Vapour.

As depicted in Figure 2, Vapour has a simple and functional design that fulfils the objectives of the project. There are three components:

cup is the web front-end. It uses the web.py framework and the Cheetah template engine, and it provides a web interface that allows the user to interact with other components of the application in a simple way. The architecture has been designed to allow other kind of interfaces. For instance, a command line interface is also provided.

teapot is the core of the application. It launches HTTP dialogs (with and without content negotiation) to evaluate the response status code and content-type. Teapot requests the URI of the vocabulary, and also the URIs of a class and a property from the vocabulary. All the resulting assertions are inserted into the RDF store.

strainer is the module in charge of generating the reports for each test performed by the application. It queries the RDF model using SPARQL to get

⁹ <http://sourceforge.net/projects/vapour/>

¹⁰ <http://idi.fundacionctic.org/vapour>

the result and trace of each test, and it produces a report in XHTML or RDF/XML. For the XHTML reports, we also use Cheetah templates.

The service can be deployed as a normal CGI in Apache or using a Python web framework. We reviewed the security of the application avoiding some common problems in this kind of applications, such as limiting requests per client.

4 Experimental results

Practical experimentation illustrates some common problems of how content negotiation is implemented, and enables the discussion on these problems. We checked some RDFS and OWL vocabularies published on the web. We chose the most frequently used vocabularies, in terms of number of instances, according to the last scientific study [7]. However, this ranking is aging (2004), so we also included some newer vocabularies, such as SKOS, DOAP and SIOC, which are also popular according to more up-to-date sources¹¹.

Table 1. Ratio of passed tests / total tests for a list of widely used vocabularies in the semantic web.

Namespace	Accept RDF	Accept HTML	Default response
http://www.w3.org/1999/02/22-rdf-syntax-ns#	3/3	N/A	RDF/XML
http://www.w3.org/2000/01/rdf-schema#	3/3	N/A	RDF/XML
http://xmlns.com/foaf/0.1/	3/3	3/3	HTML
http://purl.org/dc/elements/1.1/	2/2	0/2	RDF/XML
http://www.w3.org/2003/01/geo/wgs84_pos#	3/3	0/3	RDF/XML
http://rdfs.org/sioc/ns#	3/3	0/3	RDF/XML
http://www.w3.org/2004/02/skos/core#	3/3	3/3	RDF/XML
http://usefulinc.com/ns/doap#	3/3	0/3	RDF/XML
http://purl.org/rss/1.0/	1/3	0/3	HTML
http://semantic-mediawiki.org/swivt/1.0#	0/3	0/3	text/plain

Table 1 summarizes the results of running Vapour against a list of ten popular vocabularies of the semantic web. These results provide an approximation to the quality of the publication of vocabularies on the web. All the vocabularies were retrieved on 12/Mar/2008. For each vocabulary, the vocabulary URI, a class URI and a property URI were tested (except for Dublin Core, which does not have any class). The results show that most vocabularies are correctly published as RDF. However, it is significant that most vocabularies do not correctly provide HTML representations of the resources, even if they are available. Additionally, some vocabularies return an incorrect MIME type, such as `text/plain` or `application/xml`.

¹¹ See the ranking at <http://pingthesemanticweb.com/stats/namespaces.php> (retrieved 12/Mar/2008) by PingTheSemanticWeb.com [6]

5 Conclusions

Content negotiation is a powerful technique. Although the basic mechanism is simple, it is often badly implemented. Vapour is useful to debug and to provide advice on how to solve common problems, as well as to provide quality assurance in the best possible way.

The application presented in this paper is fairly simple, but it actually helps to debug the implementation of content negotiation in web servers. It is particularly interesting that Vapour provides the results also in RDF . Using this machine-readable format, it should be easy to build another service on top of Vapour, and to use these data for other tasks, such as a service to check the compliance of a specific collection of vocabularies published on the Web.

Current best practices (and consequently, Vapour) should probably be updated to cover new methods to publish RDF data, such as RDFa [4] embedded in XHTML pages. In the future, we would like to extend Vapour to cover more generic validations in Linked Open Data scenarios, and to help webmasters to better understand some common implementation issues.

References

1. S. Abou-Zahra. Evaluation and Report Language (EARL). Working draft, W3C, 2007.
2. T. Berners-Lee. Linked Data Design Issues. Available at <http://www.w3.org/DesignIssues/LinkedData.html>, 2006.
3. D. Berrueta and J. Phipps. Best Practice Recipes for Publishing RDF Vocabularies. Working draft, W3C, 2008.
4. M. Birbeck, S. Pemberton, and B. Adida. RDFa Syntax, a collection of attributes for layering RDF on XML languages. Technical report, W3C, 2006.
5. C. Bizer, R. Cyganiak, and T. Heath. How to Publish Linked Data on the Web. Available at <http://www4.wiwiw.fu-berlin.de/bizer/pub/LinkedDataTutorial/>, 2007.
6. U. Bojars, A. Passant, F. Giasson, and J. Breslin. An Architecture to Discover and Query Decentralized RDF Data. In *3rd Workshop on Scripting for the Semantic Web*, 2007.
7. L. Ding, L. Zhou, T. Finin, and A. Joshi. How the Semantic Web is Being Used: An Analysis of FOAF Documents. In *38th International Conference on System Sciences*, January 2005.
8. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol - HTTP/1.1. RFC, IETF, 1999.
9. K. Holtman and A. Mutz. Transparent Content negotiation in HTTP. RFC, IETF, 1998.
10. J. Koch, C. A. Velasco, and S. Abou-Zahra. HTTP Vocabulary in RDF. Technical report, W3C, 2007.
11. L. Sauermann and R. Cyganiak. Cool URIs for the Semantic Web. Working draft, W3C, 2007.
12. S. Seshan, M. Stemm, and R. Katz. Benefits of Transparent Content Negotiation in HTTP. In *Proceedings of the IEEE Globcom 98 Internet Mini-Conference*, 1998.