Linking Enterprise Data

François-Paul Servant
RENAULT
FR EQV NOV 3 31 - 13 av Paul Langevin
92359 Le Plessis Robinson Cedex FRANCE
+33 (1) 76 84 38 30
francois-paul.servant@renault.com

ABSTRACT

The "Linking Open Data" community initiative contributed a great deal to the concretization of the web of data, describing best practice, publishing large sets of RDF data on the web, and consequently giving birth to a new area of possibilities for innovative mashups using these data. Enterprises' information systems too can be envisioned as a space of linked data. We describe herein how we used Linked Data principles in a work intended to foster adoption of semantic web technologies in our company.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous; D.2 [**Software**]: Software Engineering

General Terms

Experimentation

Keywords

RDF in the enterprise, RDF based web services, Linking Enterprise Data, Linked Data

1. INTRODUCTION

Integrating the disparate applications and data sources in a large corporation is expensive, and using semantic web technologies could dramatically cut down these costs: this is the "Business Model for the Semantic Web" [1]. The lowest layer of semantic web specifications, RDF - an open and mature standard – has some very appealing properties in a corporate context [4]. RDF is indeed a format built upon a simple, powerful, and well known data model that makes exchanging, aggregating and querying information easier.

However, despite their potential, adoption of semantic web technologies seems to remain rather slow in the enterprise's world - at least this is our feeling about the situation at Renault. Not being advertised by solution providers, they are very often simply overlooked, or at best considered as "promising", but not ready to be used right now. They may even suffer from a negative prejudice, being perceived as yet another technological hype, whose promises, such as the easy exchange of information, have already been heard many times before. Also, the current focus is about "Web Services", and people do not know what semantic web technologies add to the picture - a data oriented viewpoint that complements application oriented one provided by web services. To put it shortly, people need to be explained that RDF can make it easier to exchange and use the results computed by services.

But things change! After a slow start, the Semantic Web finally took off, and it is one merit of the "Linking Open Data" community project to have proven that it was right now possible to build on the Semantic Web stack, to publish large RDF data sets, and to create applications using that data. In this respect, the description of Linked Data principles and the accompanying how-to undoubtedly were of big help [3].

Why not use the same strategy in the enterprise? A company's Information Systems can be envisioned as a space of Linked Data. Convinced that Linked Data principles were good, also in a corporate context, we decided to try to put them in practice and foster adoption of semantic web technologies at Renault.

These principles provide effective solutions for two questions that Renault regards as priorities for its IS architecture: data repositories and services. Their implementation yields indeed an architecture of REST services, easy to set up and to get connected to. Our work should make that clear.

The cornerstone of Linked Data principles and of semantic web architecture - the identification of data by the mean of URIs - is, by itself, of great importance in the definition of information systems. Not only because a sharable way of naming things is needed to support exchange of information about those things: it is indeed our observation that a frequent cause for problems is the absence of proper identification of *real world things*. Those problems tend to surface when legacy systems need to be interconnected. In [6], we described such a situation, where concepts central to the domain were not formally identified, thus hindering efficient use of existing information resources, and increasing the costs of data reconciliation.

In what will follow we describe what has been done in the work we undertook:

- the publishing of a data repository as Linked Data,
- the implementation of a simple RDF browser,
- and the prototyping of access to the published data from outer application.

We then discuss some noticeable points concerning Linked Data.

LDOW2008, April 22, 2008, Beijing, China.

2. PUBLISHING A DATA REPOSITORY AS LINKED DATA

2.1 Previous experiences

We had some previous experience with the publishing of RDF Data.

Semanlink, the first of these experiences, is a free tagging tool developed by the author, where tags are SKOS-like concepts, identified by URIs, all dereferenceable [5].

The second was a prototype repository of repair and diagnostic operations, modeled with OWL, and developed to provide a probabilistic diagnostic tool with data [6].

In these experiences however, implementing Linked Data principles had not been the central point of the work. Our concern in this new work was to emphasize the publishing and consuming of linked data. The objectives were to better explore the topic, and to highlight the benefits of the method, in a corporate context. We also wanted to produce guidelines and sample code for Renault developers: if Semantic Web technologies are to be used in projects, we'll need them trained to these techniques.

2.2 Chosen use case

As a use case we chose a repository created recently by the department in charge of after-sales repair documentation. Basically, it is a dictionary of the terms that documentation writers may use when describing repair methods. The first purpose of this repository is to enforce an homogenous naming scheme of things throughout the whole documentation. These terms are translated into the many different languages the documentation is produced in, and they are classified in a SKOS-like hierarchy. Finally, the repository also contains a link to a data set produced by a different department, the department in charge of spare parts: a list of so-called "generic parts" is associated to each such term. A "generic part" is a part seen through its function in the car (for instance "engine", "air conditioning compressor", "right front wing" ; several spare part references correspond to a "generic part").

We chose this repository for several reasons. First, it's simple enough, yet significant, and it is supposed to be well managed. Second, there is no way to access its data from another application: it is not available as a service, hence, publishing this data as Linked Data corresponded to an actual need. Third, a dump as XML is available. It was therefore easy to produce RDF out of it. Finally, we could envision interesting use of this data, that we would be able to demonstrate in the context of our work:

- access to repository's RDF data from another application ;
- inclusion of RDFa data in repair methods generated as XHTML pages (repair methods are produced as small chunks of XML, and they contain references to the terms of the repository. Having RDF statements inside the page allows for interesting features using javascript)

2.3 Development environment

Java is the main language used for software developments at Renault, and therefore the obvious choice for our work, given the high quality of Jena's implementation of the semantic web stack.

2.4 First steps

2.4.1 Minting URIs for the items of the repository

Minting URIs for the items of the repository was not a big deal: as they already had an id, it was just a matter of choosing a namespace where we would be able to publish the data: the URI of an item of the repository is just the concatenation of that namespace, of the item's class name (the name of the table in the database the item is primary key of) and of the item's id. Regarding the "generic parts", which, as we mentioned earlier, do not pertain to this repository, they should be published by their owner (the parts department), under their own namespace. Of course it is not the case and, to get started quickly, we minted URIs for these external items inside the same namespace. It will be possible to fix this later, for instance using owl:sameAs statements to link them to their true URI when their natural owner makes them available as Linked Data.

2.4.2 Hash or slash URIs

Another decision concerns the type of URI to use. Best choice depends on the size of the repository. Hash URIs are simple, but they suppose that the whole file gets downloaded when dereferencing one of them. It is therefore better in our case to use slash URIs, which can be dereferenced one by one.

2.4.3 Information and Non-Information Resources

Clearly, the items of the repository are "real-world things", not "information resources": to respect the web architecture, we must implement the "HTTP-range 14" resolution when dereferencing them.

Using Non-Information Resources (NIR) requires the creation of three different URIs: one for the resource itself, one for the RDF data describing the resource, and one for an HTML description. There are several ways that make sense for the naming of these URIs. We used [namespace]:[resid] for the NIR, [namespace]:[resid].rdf and [namespace]:[resid].html for the two corresponding information resources.

2.4.4 Producing the RDF

An XML dump of the repository being available, it was easy to produce RDF out of it - a really simple RDF, by the way, without blank nodes, for instance.

2.5 The "LED" Servlet

URIs of the data set can be made dereferenceable by using a servlet, the "context path" of which matches the namespace chosen for the repository.

The servlet must of course have access to the RDF data of the repository. As the data set contains no more than 60,000 statements, handling it as a Jena memory model, loaded at startup, was not a problem.

2.6 Dereferencing URIs of NIRs

When an agent gets the URI of an NIR, the servlet must respond with a 303 HTTP status code, and include a redirection to the URI of an information resource that best fits the preferences expressed in the "accept" HTTP header of the request - that is, in our case, the URI of either the RDF or the HTML describing the NIR.

A servlet has access to the HTTP headers of a request. It can therefore analyze its accept header to decide what it should return.

2.7 Answering requests for the RDF about an NIR

When the client requests for the RDF data about an NIR, (that is, when it dereferences the ".rdf" URI), answering is just a matter of extracting the statements of interest from the Jena Model, and of returning them, which is made easy by Jena serializers. As our repository is very simple, there is no question about the content to be returned: all statements of the form <nirURI,?,?> and <?,?,nirURI>. We just added the statements defining the links between the three resources nirURI, nirURI.rdf and nirURI.html. We can think about including labels of linked resources. This can be a useful optimization when we know that the RDF returned will be displayed as HTML. We didn't do it by default, only for resources linked by certain properties. (The question of the amount of information to be returned about one URI is briefly discussed later: in other cases, the behavior adopted here would lead to a uselessly huge number of statements).

2.8 Answering requests for the HTML about an NIR

Generating HTML about a resource from its RDF description is of course one important topic. It can be done on the server, for instance using JSP. (That's what we had done in our previous experiments with Linked Data publishing, such as Semanlink). But it can also be done on the client, thanks to the javascript RDF parser made available with the Tabulator project [7].

2.9 Generating a page out of RDF data using javascript on the client.

That's the road we chose, because it allows for a nice architecture:

- it provides a clean separation of "Views" and "Model" (in MVC parlance), with easier reuse of GUI widgets,
- it decreases the load on the server,
- it gives the possibility to change the display on the client without sending a new request to the server,
- it allows to incrementally load RDF.

The principle is simple: a request for the HTML about an NIR returns an HTML page which is almost an empty box (or a template), containing a call to a javascript function that takes the URI of the RDF data as argument. It is this javascript that downloads the RDF data, and displays it.

2.9.1 Parsing problems with Internet Explorer

We faced a difficulty with Tabulator's javascript RDF parser, as it didn't support Internet Explorer (neither 6 nor 7) at the time we undertook our work (this was Tabulator 0.8). We had to correct that, as most people use Explorer in our company, and it wouldn't be acceptable to propose a solution that doesn't work with it. The patch is available for download by interested people.

2.9.2 Generic display

In a first step, we didn't do much more than displaying the RDF about the described resource in a very generic way, that is listing values, property by property.

2.9.3 Tree-like GUI widget

On important part of the repository is the classification of the terms in a SKOS-like hierarchy. It is supposed to provide a way for a human user to easily navigate inside the corpus of data and help her finding the term she's looking for. As navigating a hierarchy of concepts to access data is so common, we developed a dedicated tree-like widget, that we took care to make reusable (Semanlink GUI for instance could use it, instead of having its trees created by the server. We'll see another reuse example later).

2.10 A simple RDF browser

We had not mentioned it yet, but what we described until now supports only dereferencing URIs from our own data set, and we had limited our GUI based on this constraint. It is indeed standard javascript security to forbid connections to servers other than the one the page comes from. It is possible to bypass this constraint, as Tabulator does, using Firefox with one of its default settings changed. This was not an option in our case: we had to work with standard version and configuration of browsers. Dereferencing URIs from other servers therefore meant implementing an usual trick (namely, an HTTP proxy: requests to dereference URIs outside our domain must be sent to the servlet, which forwards them to the actual server, and then returns the result).

Implementing this trick is all what it needs to transform the solution into a very simple, yet generic RDF browser: RDF can be downloaded from anywhere, and displayed using javascript.

2.11 Getting linked data from another application

An important aspect of Linked Data is the fact that it provides the implementation of a service to which applications can connect over HTTP to get data and use it as they see fit. It was important to demonstrate that this can actually be done without difficulty, at a low cost in terms of development for the client application. In order to provide sample code, we implemented two kinds of connections to the repository's data: one in Java, using the Jena API to parse and use the RDF, the other in Javascript.

For this demonstration, we used a completely unrelated tool that we had built some time ago for the parts department. It is a web application that computes and displays information about "generic parts": the user enters the code of a "generic part", and the application displays a list of corresponding spare part references.

The first feature we added to this application, in Java, is simply the possibility to list the spare part references corresponding to a term of our repository: the "part" servlet connects to the repository, dereferences the term, parses the returned RDF, extracts the corresponding list of "generic parts", and for each, computes the list of spare part references. For the second feature, we reused our tree widget: the user can use it to navigate down the hierarchy of concepts of our linked data set, to choose either a "generic part" or a term, and to get the list of corresponding spare part references.

2.12 Conclusion concerning this prototype

We implemented in Java and Javascript an example of a repository published as linked data. It is composed of a servlet that uses a Jena Model containing the RDF data, and that ensures the dereferencing of URIs, respecting the principles regarding Non-Information Resources. Basically, this servlet only produces RDF output. The display in HTML is done in Javascript. This is enough to build a very simple but generic RDF browser, and to publish the repository's data, providing the functionalities of a REST web service. We demonstrated how to connect to it from a program, and how to use its data.

We now plan to include a SPARQL endpoint, to complete our accompanying how-to.

We think this is a fairly noticeable achievement, for a relatively simple development, which is almost completely reusable, and easily extensible. One obvious idea is to improve the RDF browser, implementing, for instance, templates allowing to customize the display depending on the type of the resource that gets dereferenced.

The solution should be compared side by side with more traditional or advertised ways to proceed, such as SOAP web services. Let us just note some points. This approach respects the web architecture, and this has its benefits: caching, for instance, which is important for a repository like this one, whose data barely changes. At the opposite of WS-* services, this service only uses HTTP get, and therefore benefits from the standard HTTP cache mechanism. The last point we would like to insist on is related to the use of RDF, which is a generic data model. We do not have to learn a special syntax to be able to use the service: we directly manipulate the data, not a specialized API. Furthermore, any chunck of RDF extracted from the repository could be transfered from application to application, possibly aggregated with more data, and still remain completely understandable with just a standard RDF parser. This reduces the cost of development in client applications (a question that seems to be sometimes overlooked when speaking about services).

This concludes what had to be said about this prototype, and we are now going to discuss some points about Linked Data, in no particular order.

3. RDF FORMS

Quoting [2]: "The Semantic Web [...] is about making links, so that a person or machine can explore the web of data." Beside hypertext (“href”) links, forms are an important feature of the web. How does this transpose to the web of data? Shouldn't there be a standardized way to "include forms" in RDF data? This would allow a server of RDF data to require some input, with a well defined meaning, from its clients (should they be humans or machines).

We had a use case in the field of technical after-sale documentation. The repository of repair and diagnostic operations mentioned earlier can be used by humans (mechanics looking for information about how to repair a car) and by programs: for instance a program that computes estimates needs to get the list of parts that are necessary for a given repair on a given car, and the time needed to perform the repair.

Suppose for instance that you have to replace the engine of a car, and that you want to get information about how to do that. The repository contains a concept "ex:engine_removal". Now, for any given car, there is one and only one method to (correctly) take apart the engine. You are not concerned by the information that you would get by dereferencing "ex:engine_removal", rather by the subset that is relevant to your car. This subset depends on the characteristics of that car. It would not make sense to return all the information about all the cars. Not only because it would be a waste of bandwidth, but also because it would be difficult for the client to understand this information. (This information would indeed contain "conditional links", things such as: “if condition then statement”, where condition is a Boolean function of the characteristics of the car).

Typically, the service extracts from its underlying database the list of all documents matching "ex:engine_removal". Each record has a property "condition" (a Boolean function of the technical characteristics of a car which returns true when the document is relevant for a given car). To filter the list, the service has to evaluate these Boolean expressions. Let's say that the engine removal depends on the model and the engine type of the car. We could ask the user to enter those values returning some RDF such as :

```
<rdf:Description
  rdf:about"ex:engine_removal">
  <form:hasForm><form:Form>
      <form:param
        rdf:resource="ex:model"/>
      <form:param
        rdf:resource="ex:engine_type"/>
  </form:Form></form:hasForm>
</rdf:Description >
```

If the client program knows the conventions of this "form vocabulary", it can understand that it has to provide value for model and engine_type. How it determines these values varies (if there is a human user, it can generate an HTML form). The point is that the client program knows (or can discover) the exact meaning of the parameters of the form. If it is able to provide the answer, it will then construct a URL including the values and dereference it to get the data it is looking for.

We chose this example, though it is a bit long, on purpose. The European Commission has emitted a directive that requires from automotive constructors that they publish their technical documentation about repairs. A specification by an OASIS technical committee describes how this should be achieved, using RDF for metadata. The protocol for this situation could be improved using “RDF Forms”.

We described here a form with a GET method, but POST methods are of course interesting as well.

4.FINDING THE URIS OF “REAL WORLD THINGS”

This is probably the major difficulty with Linked Data on the web. The publishers of the large data sets of the LOD project accomplished an important effort to interlink their data, and they built a huge source of identities for real-world things. But how can a user discover how to say “Paris” or “Hamlet”?

4.1 The case of enterprise data

Let’s note that this is not really a problem in the corporate context. Companies indeed do have large and standardized vocabularies, that are shared throughout their whole organization, to name many of the things they manipulate in their operations. Of course, everything is not perfect: different departments sometimes give a different meaning to the same term, and this may be a cause of misunderstandings and problems. We believe that adopting URIs for identification would reduce the number of such cases: URIs including the designation of their “owner”, they are incentive to check for the real meaning of the thing before deciding to use it.

4.2 On the web

The question is of first importance. If we don’t have tools to help a user discover the URIs for things, she won’t be able to write statements using a shared vocabulary. Best she’ll be able to do is to write statements using her own vocabulary. She shouldn’t be left helpless, because this hurts the chances of the semantic web to be largely adopted.

We think that the publisher of the massive linked data sets should try to provide tools to help connect to them, and find the URI of things.

The problem is of course difficult, but pragmatic tools can be of great help. If I type "Paris" in Wikipedia, I get to a page about the capital of France, with a link to a disambiguating page. Implementing such a service, returning its results as a small chunk of RDF, should not be a problem for the large LOD projects, and this would be very useful.

More complex tools can be thought of. If a data set provides a SPARQL endpoint, some interesting tools can be developed by third parties.

4.3 Semanlink

We will try to participate to this effort. As noted earlier, Semanlink publishes data as RDF, following Linked Data principles. We hope to be able in the next months

- to work on the linking of Semanlink to the LOD data sets,
- to make results of the tag search available as RDF,
- to build tools that help discover whether a given concept is actually used in a Semanlink data set,
- and to work on the interlinking of several Semanlink databases. We hope that this will eventually provide a test bed for semi-automatic reconciliations of portions of independently developed vocabularies.

4.4 How to get the URI of an NIR when you know the URI of the corresponding HTML ?

Most of the time, what you see of an NIR is the HTML page that gets displayed in your standard web browser when dereferencing its URI. How do you get back from this HTML to the URI of the NIR? Today, there is no easy way to get it. If Linked Data publishers follow the recommendations of [3], the URI of the RDF data describing the NIR can be extracted from the HTML page, but not the URI of the NIR. If you're interested in it (for instance because you want to write a statement involving this NIR), you have to download the RDF, parse it, and find the statement linking the NIR to the HTML. It would be easier to have the statement describing the link between the HTML page and the NIR directly in the HTML (probably as RDFa) We think that suggesting data publishers to do so would be a nice addition to the "How to Publish Linked Data on the Web" document [3].

5. AMOUNT OF DATA TO RETURN WHEN A URI GETS DEREGERENCED

This is a known issue with Linked Data, with several aspects. In particular, it is not always practical, or wise, to blindly return all triples containing the URI that is dereferenced. This information can be huge, or computing the statements involving some special properties may be a heavy task. So, returning all the statements that a server is able to provide can be a waste of bandwidth, or of server resources, if the client is not interested with them. What could the server do, when such a URI gets dereferenced, to avoid waste (that is, to avoid returning all statements involving the URI), yet let the client know that more information is available and could be returned if needed? Some interesting suggestions were made, based basically on the idea: "look up there to get triples involving that property". These suggestions imply a new convention (that is, the definition of a few properties). We would appreciate an agreement of the Linked Data community on such a convention.

6. CONCLUSION

We are convinced that Linked Data principles are good for the web as well as for companies' information systems. In a work of evangelization about Semantic Web technologies inside our company, we published as linked data a repository, without difficulty. We showed how to connect to the resulting service from an outer application. The code written is largely reusable, and expandable. And we will soon reuse it to publish new sets of linked data. For instance, we will shortly implement the publishing as RDF of data served by a SOAP web service. Increasing the number of use cases should convince about the flexibility of the method. We'll continue our work in the field of after-sales technical documentation. This field looks like a typical use case for Semantic Web technologies, given the number of business objects that have to be shared among the many systems involved, in a corporation-wide process. But it is also a field that is slow to evolve, partly for the same reasons.

Concerning Linked Data on the web, we think that the community should work on defining a small vocabulary to better handle some problems that surface (such as the amount of data to return when dereferencing a URI), or to provide new functionalities (such as "RDF Forms").

7. REFERENCES

[1] Berners-Lee, T., "Business Model for the Semantic Web", 2001 http://www.w3.org/DesignIssues/Business

[2] Berners-Lee, T. "Linked Data", http://www.w3.org/DesignIssues/LinkedData.html

[3] Bizer, C., Cyganiak, R., Heath, T., "How to Publish Linked Data on the Web" http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/

[4] Feigenbaum, L., "Semantic Web Technologies in the Enterprise", 2006 http://www.thefigtrees.net/lee/blog/2006/11/semantic_web_technologies_in_t.html

[5] Semanlink http://www.semanlink.net

[6] Servant, FP "Semantic Web Technologies in Technical Automotive Documentation" - CEUR-WS.org/Vol-258 - OWL: Experiences and Directions 2007 http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-258/paper04.pdf

[7] "Tabulator: Generic Data Browser" http://www.w3.org/2005/ajar/tab