

A Query-Based Template Framework For Linked Data Applications

Sjors E.F. van Berkel
Student number 1262882

Abstract

Linked Data is a relatively novel subject in web application development. Because of this novelty, there have not been many tools around for development with this technology. In this essay, we talk about the potential of Linked Data applications, and the problem of writing a framework that enables development of such applications.

Next, we present a framework for creation of web applications from Linked Data. The framework is lightweight and simple, and uses querying and templates to transform this data to a web application. We discuss the advantages of the framework, and the anticipated problems. To illustrate utilization of the framework, we use the scenario of building a web application for an online department store.

1 Introduction

Since the introduction of the Internet, its applications have evolved constantly. Especially the evolution of the dynamicity in applications on the Internet is a noteworthy aspect. While Internet pages started out as enriched teletext pages, over the years their content has become more dynamic, leveraging large content management systems in order to maintain structure, and to separate content from technical aspects so that people with a certain expertise in content can do their jobs without having to worry about concerns irrelevant to their subjects.

The evolution of dynamicity does not stop there. Although sites have relied more and more on user or moderator input over the years, the owners of the site were still in control of the data that was input to their site. The default setup for the majority of modern websites is some sort of content management system connected to a database that is read from and written in by the content management system. Individual sites may vary, but most of the time they are a derivative of this setup. The advantages of such a controlled environment are obvious. When a site is for example your main source of income, it takes courage to pass off control, and generally it does not compromise the functionality of the website keep total control.

Another development that has emerged over the last passed years is the embedding of external feeds onto web sites. For example, it is popular to embed twitter feeds into sites that contain messages related to the site's content. Although there is some control over the filters that choose which messages to show and which not to show, this is already a control trade-off. Another example is showing RSS feeds from other sites with recent articles, because site owners think their users might value that content also. In some way, these parts embedded in sites are a recognition of the idea that sites can be more interesting when they exist in an ecosystem rather than on their own. However, integration done in this classic way is always really superficial. In fact, it is only showing external data, instead of really using it. Apart from that the integrations always feel really external too. We can do better on all of these aspects.

The idea of Linked Data is to take the idea of an ecosystem to an extreme, and outsource every possible dataset that is needed for an application, to an authority on the dataset subject, or at least to some entity that has more authority than the creator of the site itself. The underlying idea is that when information on a subject gets richer it automatically propagates to applications that use this dataset, creating a single point of data enrichment (for one subject).

While this is in many aspects a positive innovation, it also introduces some new problems that need to be dealt with for the technology to become a success. We will cover some of these problems and their solutions in the next sections.

2 Related work

This essay is built upon my years of experience in web development, and a recent introduction to the Linked Data principle. Frameworks that serve as an inspiration are:

- Django¹, a web application framework written in Python using the model view controller paradigm
- WebDSL², a domain-specific language for developing dynamic web applications with a rich data model, developed at the Delft University of Technology
- Green Valley CMS³ (formerly known as Discovery Server), a proprietary content management system using a template render engine, developed by Green Valley B.V.
- XSLT⁴, a standard way of transforming XML⁵ data into human readable HTML pages

3 Problem description

Since the introduction of Linked Data, there have been lots of impressive examples that leverage the power of distributed datasets by integrating them into one application. Although these results show the potential of Linked Data, its real potential, synergy of all kinds of open data, remains untouched. We can see a parallel with the development of standard websites, where people with knowledge of subjects could not fully cooperate due to technical limitations⁶. Also, even for experienced developers (and thus for the companies they work in), it is very unfortunate to develop a lot of code for a simple application. The question is, **can we develop a framework in which tasks that are ought to be trivial for Linked Data applications, also have trivial implementations?**

As an example we will look at the website Bol.com, an online department store that has become very popular in The Netherlands. Bol.com is an orderly catalog that gives some information about the products, as well as the price, and an indication for the delivery time. Bol has its own

¹<http://www.djangoproject.com/>

²<http://webdsl.org/home>

³<http://www.greenvalley.nl/>

⁴<http://en.wikipedia.org/wiki/XSLT>

⁵<http://en.wikipedia.org/wiki/XML>

⁶It must be said that linking datasets, and creating applications with them will probably always require some technical skill while writing content for the Internet does not

warehouse for the most popular products, and they do back-orders to other warehouses (possibly owned by other companies) for products that are not as popular.

For this case study we imagine that Bol would like to change their company, and have a smaller warehouse, and smaller administration. This would mean that a larger portion of their products would be stored in other warehouses, and preferably the metadata of the products would be provided by the vendor of the product or an external products database, rather than by an employee of Bol.

For this scenario we will assume the extreme, Bol will have no warehouse of its own and does not have any employees that fill in metadata for products. Instead, every product is stored at third party warehouses, or at product holders themselves (for example a CD is created and sold by Sony Music). For the metadata of the products there are several datasets available. Some datasets are created by the product holders, other datasets are maintained by communities on the Internet, and some are created by the third party warehouses.

4 Problem solution

4.1 Endeavor one: A model driven approach

Because Linked Data is all about semantics, it is obvious we should consider incorporating (or preserving) semantics in our framework. The way to achieve this in programming is using model driven development (MDD). In MDD, we create a model of the entities that are important for the working of the system. When the model is done, in the other stages of development, we can program in terms of the domain rather than in concepts abstract to the domain.

MDD is meant to enable rapid development and decrease the amount of errors during development by automatically inferring code from the model. Also, it is possible for non-technical people to reason about the code, which satisfies one of our main goals for the framework. While these properties are very positive ones, there are also some unwanted side effects for us to apply MDD.

One disadvantage would be that for true MDD, we would have to create models for every dataset we want to incorporate in our application. While this is normal for MDD, the goals for our framework were to enable rapid development of trivial tasks, to remove thresholds rather than introducing them.

Another problem with creating models is that normally, the entities (or objects) that come forth from these models are read/write entities. In our situation, because we can have no influence on external datasets, the external entities will only exist of a read-only mapping to the data. The problem with MDD is that it obscures the underlying code because there is confidence that the framework and models are correct. This confidence (especially in the model part) is strong because the model is developed in-house and does not change unless the developer says so (checking the model happens at compile time). However, in incorporating external data, we would introduce an external factor that can invalidate our mapping, and thus our model. Since the real code is obscured, and the different models in a system are very likely interconnected, it is very hard to tell what will fail. Even if we can see what goes wrong, which might be pretty hard, in order to fix it in the code, developers will have introduce a lot of boilerplate code. It will heavily complicate the code that (by principle of MDD) ought to be more domain centered rather than dealing with technicalities.

We can conclude that MDD might be a little on the heavyweight side, and does not live up to its potential for this type of problem. We should look at another, more lightweight approach for

our problem.

4.2 Endeavor two: A query-based template engine approach

Another approach we can try to apply to our problem is to use a template engine in combination with queries to the datasets. Generally applications built in template engines are split up into small to medium sized chunks that contain standalone functionality. For example, on most sites, a box for logging in to the site is maintained in a separate template. It should be noted that MDD approaches also work with templates in order to render their data, but the difference here is really the underlying query system as opposed to a model in MDD.

Dataset queries are superficial compared to MDD, as there is no implicit interpretation of the domain in the code. Queries work bottom-up, only specifying and using information in a template that is necessary for its functionality, whereas MDD uses a top-down approach, having a complete (though maybe abstract) view of the domain. This bottom-up approach in combination with external datasets is easier for the developer to debug, since missing data (the external dataset is offline) or incorrect data (invalid types) can be detected inline, and do not necessarily corrupt the domain space.

Although the method has some negative aspects, as we will find out later on, it performs pretty well for Linked Data applications, as we will see in the remainder of this essay.

4.2.1 Framework proposal

The idea of the framework is very simple. Let each template be called by a URL with zero or more parameters, which can be used in the template. Templates can specify inline queries that possibly use input of the supplied parameters. The output of the query is stored in a variable that can be used in the rest of the template, and passed to nested template calls as a parameter. Next to queries there is a mix of markup and logic in the templates.

To make navigation through data more simple to achieve, it is possible to specify a default template for each of the `rdf:type` values that are used throughout the datasets. By doing this, links to entities will automatically be forwarded to the right view, for example an album has a simple link to its artist(s), and this will automatically lead the visitor to a page with all the information available about this artist.

In order to gain some control over the data that is used, we can only use datasets that are specified in a settings file, we will see why this comes in handy later on (Section 4.2.3). We will use SPARQL⁷ as the query language since it is the standard for talking to Linked Data datasets. SPARQL has elaborate read and write statements, and thus it is sufficient to retrieve data and render it in our templates, and write data locally.

4.2.2 Advantages

The advantages of this framework are mostly in its flexibility and simplicity. Because of the lack of "rules", it is easy to quickly prototype an application, mix up different datasets, and present them with HTML (or at least something the browser can show, SVG or canvas) to the user. For the example of Bol, it is easy to see we can gather information about a product together by using multiple templates, an information template, a related items template, etc.

⁷<http://www.w3.org/TR/rdf-sparql-query/>

Another thing that we are able to do in the example of Bol is to display only the least expensive of offers from the multiple datasets in which a product exists, creating some competition between the sellers of the products. A different approach would be to display the prices of different sellers accompanied with the availability and shipping time.

Also in the example of Bol, when third party product libraries have gathered information about products related to a certain product, we can take advantage of this by using this information instead of generating all this information ourselves. Also, any information that is provided that can help determining duplicate entries (`owl:sameAs`), or maybe different names for the same artist can be extracted from the external datasets, making our application more coherent.

We may want to maintain local information, which is very likely in the end, for example a client database and a list of ordered products. This is also possible in our framework, because we can create templates that execute queries.

4.2.3 Anticipated problems

Two of the problems that may arise with the method of using several external datasets are speed and reliability. If at any time a dataset is offline the visitor of Bol is prevented from finding a product, or maybe finding it to be too expensive at another seller, that is online. Also, if at every user query the datasets of different sellers and other product databases have to be contacted, this will result in a very low performance. Therefore we would want to cache the external datasets on a local machine. Because we are building a framework this can be done automatically, but this also results in some problems. For example, one problem is: how long is a product price valid? To solve this problem we can put in a cache timeout for each external dataset, in consultation with the owners of the dataset. If a product is sold within the cache timeout from the last time the dataset was contacted, the seller has to sell the product for that price. During the cache time, our application will try to refresh the dataset but if it fails to do so the dataset and thus its products will ultimately disappear from our application because we cannot be sure the prices are still valid. Another solution for this problem is to force sellers to accompany their prices with a start and stop tag that indicates a time and date span in which a product may be sold for a certain price.

The second problem with caching is more difficult. How can we be certain a product will still be available from a seller if it has been cached for a considerable time? Since sellers are able to sell to multiple companies (not only to Bol), we have no idea of the speed at which their products are sold. We could ask the sellers to accompany their products with an indication of how long it will be in stock, but even then there is the problem of mass product purchases, which are impossible to predict.

Of course these problems are very specific to our example, but in general, every domain in which the framework will be used will probably have specific cache and cache invalidation problems.

A negative aspect of the framework is the amount of code that developers will eventually produce creating an application. Because every template can contain its own queries, the queries will eventually be widespread over the application. This makes it hard to process possible changes in the datasets because the developer will have to change all the files in which the dataset is used. Because this is very error-prone, it is recommended to collect all queries in a configuration file, to stimulate more reuse of queries, and create better overview for the developer. Although this is a sensible thing to do, it is still possible that many queries will be created due to subtle differences in their functions, or duplicate queries.

Because we chose not to use a MDD approach, no data is automatically linked. The result is that all linkage of data is done in the queries. This will result in elaborate queries that will become more complex when more datasets are linked. This is also a reason to stimulate reuse of queries and maintaining documentation of how datasets are linked.

5 Evaluation

In this essay, we reviewed a framework for rapid development of Linked Data applications. Although the framework seems to enable rapid prototyping, it does not entirely prevent the code from becoming convoluted when the applications grow more complex. It does so by using templates, because they are by definition made for reuse, but specifying the relationships between datasets with queries may become a hassle over time. This is the trade-off for not wanting to specify a model, for reasons described in the essay.

Because in the current design every action (e.g. writing something in the local database) also has to be triggered by a template, many templates will need to be created in order to build a full-fledged application, so this should be revised in a future version.

Also, it would be interesting to see if there are any advanced methods for caching and invalidating cache for Linked Data datasets, since this is a very important feature for the working of the framework, and the methods described in the essay are insufficient, even for the Bol.com example.

The presented framework does however fulfill its task of rapid development of Linked Data applications, and since it is hypothetical, it would be interesting to see it in operation.

If multiple parties can really agree on the format of datasets (such that they are coherent with the models), and figure out some way to guarantee uptime, it would be very interesting to investigate in MDD as a development method. Nevertheless, this seemed too naive to assume for now.

For the Bol.com example, the presented framework is surprisingly sufficient, but this is mainly due to the open-ended setup of the framework, there is still a major amount of work to be done to actually implement it. Apart from that it is doubtful that Bol would change its company strategy so much, but it could be a company strategy for other (younger or smaller) companies.

6 Conclusion

In this essay we discussed the potential of Linked Data. Thereafter we discussed the need for tools in order to enable simpler and faster development for applications incorporating these Linked Data datasets. Next, we tried to solve this problem with model-driven development, but due to the nature of Linked Data, this did not turn out to be a perfect match. Then we presented a framework that uses queries and templates in order to use information gathered from different datasets in a single web application. We discussed the possibilities and anticipated problems of such a framework with Bol.com, an online department store, as an example. Thereafter we evaluated the presented framework to conclude that there are still some open ends to solve, but the framework is already a vast improvement to having no such tools.