

SPARQL Diamonds

an architectural pattern for rapid development of data-driven applications

Danny Ayers, 2021
hyperdata.it
danny.ayers@gmail.com

Abstract

This document proposes a software-architectural pattern as a means of providing solutions to recurring application development demands in the context of the 'Web of Data'. As such a pattern it can be described in the abstract, but it is felt that it has most utility in concrete scenarios involving the Web browser in conjunction with SPARQL servers and the Web at large. With this in mind two simple applications for this environment will be described.

The infrastructure components of a 'SPARQL Diamond' are a SPARQL server and a Web browser. A text templating engine intermediates SPARQL queries from the user interface and the server endpoint (request). Templating is again applied between the server (response) to deliver desired HTML rendering of the results.

The approach could potentially be used to create content-delivery sites with the same general functionality as typical database-backed systems. However it is felt that it particularly lends itself to rapid prototyping and the enhancement of existing pages with 3rd party information.

The primary intent here is to demonstrate that by exploiting widely-deployed systems together with a small number of utility libraries to implement the SPARQL Diamonds pattern, data-driven

applications can be created with minimal effort. Reference will be made to the current paucity of adoption of Semantic Web technologies, with the proposition that the description of simple patterns can reduce the demand on developers.

Keywords: SPARQL, RDF, Web, software pattern, Wikidata

Introduction

The phrase ‘The Semantic Web’ has been fairly widely known in the Web development community since at least the 2001 publication of the Scientific American article of that name. The core idea is to evolve the World Wide Web from a primarily document-oriented system to one which could manage arbitrary data using the same fundamental concepts (notably resources with their identifiers and representations as described in the HTTP protocol [<https://httpwg.org/specs/> HTTP Documentation : IETF RFCs and IANA registries]). A more intuitive label for this evolution was later coined : ‘Linked Data’.

In 2001 the necessary technologies were still in their infancy so it is no surprise that there was little adoption at the time. But with the SPARQL 1.1 specifications [<https://www.w3.org/TR/sparql11-query/>] achieving W3C Recommendation status in 2013, arguably all the pieces were in place for general adoption. However, relative to other contemporary developments in Web technology, uptake has been minimal.

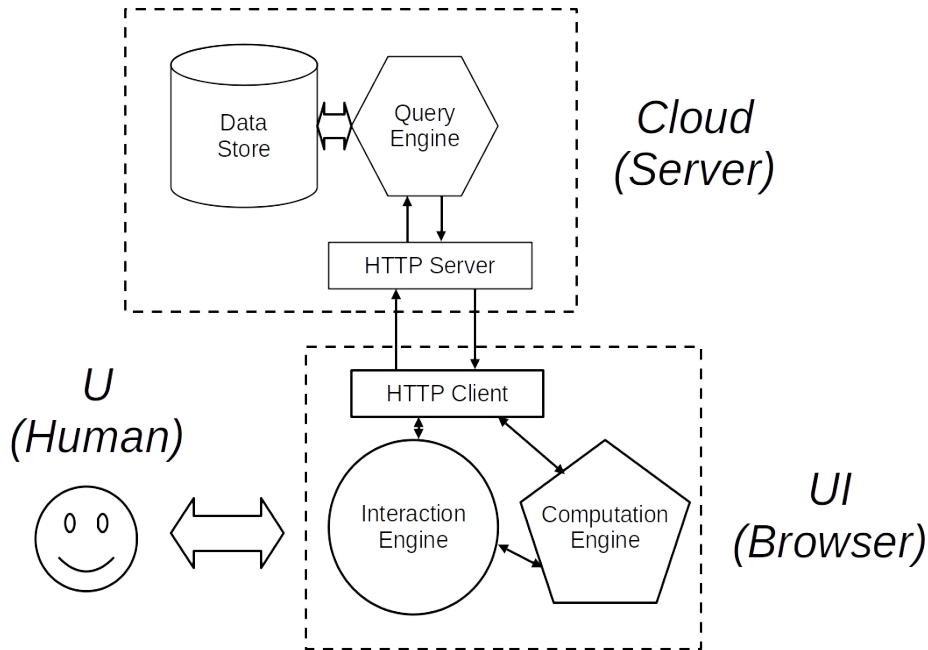
There are no doubt many factors that have contributed to this state of affairs, but one is fairly certain : the perceived complexity of developing a Linked Data-oriented application. In the early days at least, this was a fair perception : the benefits could be obscured by having to deal with a confusing format (RDF/XML) and the mere existence of more arcane technologies such as OWL and its associated reasoners. Where there has been marked success it has often been somewhat ‘through the back door’, notably with site developers exploiting page metadata to provide *Search Engine Optimization*. The inclusion of, for example, schema.org terms in a HTML page can improve that page’s visibility. Meanwhile the schema.org vocabulary is based on the RDF data model, developed using RDF Schema.

A secondary goal is to make life easier for developers unfamiliar with Semantic Web technologies by minimizing incongruencies

The technique has no doubt been used elsewhere before,

Operating Environment

The environment of SPARQL Diamonds is shown in Fig. 1.



This is very similar to a traditional database-backed Web site.

The key difference is that instead of the query and presentation construction being carried out server-side, here it is carried out against a SPARQL Server, that work is carried out at the client. Although the block labelled *Cloud (Server)* shows typical internal subcomponents, the details of implementation are irrelevant. All that matters is some degree of support for the SPARQL Protocol [<https://www.w3.org/TR/sparql11-protocol/>]. In broad terms the *Interactions Engine* is the HTML DOM (and its representation to the end user), the *Computation Engine* being the Javascript interpreter typical of current browsers.

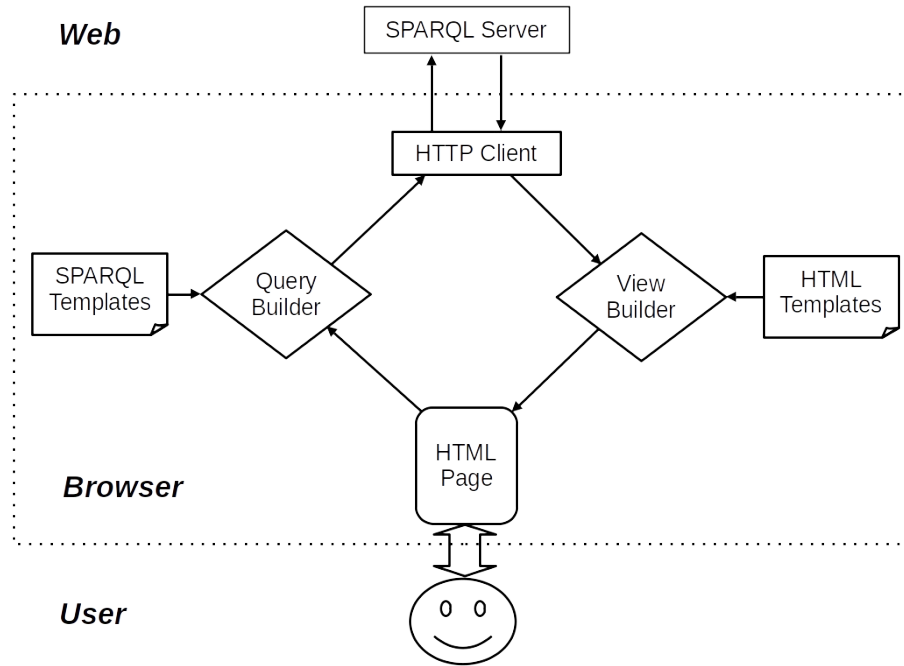
As a Web-based system, SPARQL Diamonds are clearly dependent on the HTTP protocol. More specifically, they are inclined towards the REST [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm] architectural style.¹

¹ Strictly speaking, SPARQL servers fall outside of the REST style, notably due to the use of the SPARQL Query Language inside the query part of URLs rather than REST's notion of potentially opaque identifiers that act

Client-side, the configuration described here will be that of a typical Web browser. But this doesn't preclude the use of the Diamonds pattern in a different environment such as a dedicated App for a mobile device. The only requirements are a HTTP client, a processing component to manipulate data and control its flow and some means of interacting with the end user.

The Diamond

Figure 2 shows the SPARQL Diamonds pattern in graphic form. The components labeled Web, Browser and User correspond to the three blocks in Figure 1. The Diamond resides in the Browser block. As with most Web sites, a HTML page acts as means for the end user to interact with the system. Similarly, a HTTP client interacts with servers on the Web. Here, within the *Computation Engine* above



Linked Data

purely as names for given resources. But in practice this can be viewed as an implementation detail - the combination of endpoint URL with query could be interpreted as merely an arbitrary string identifying a resource, or even aliased from a more human-readable URL.

Templating Engine

Templates are incredibly useful in every industry, and within software; text search-and-replace is an invaluable tool. Given that the protocols and formats around the Web can generally be expressed as text, templating should be useful. Take for example the following :

```
<a href='~{link}~'>~{link}~</a>
```

The character string here is HTML, but on rendering directly would make little sense. But with replacement using predetermined regular expressions, where *link* is a named string variable and `~{ }~` are chosen delimiter sequences, it can become :

```
<a href='http://example.org'>http://example.org</a>
```

In the same fashion, this :

```
SELECT ?x WHERE {  
  ?x rdfs:label "~{label}~"  
}
```

Can become :

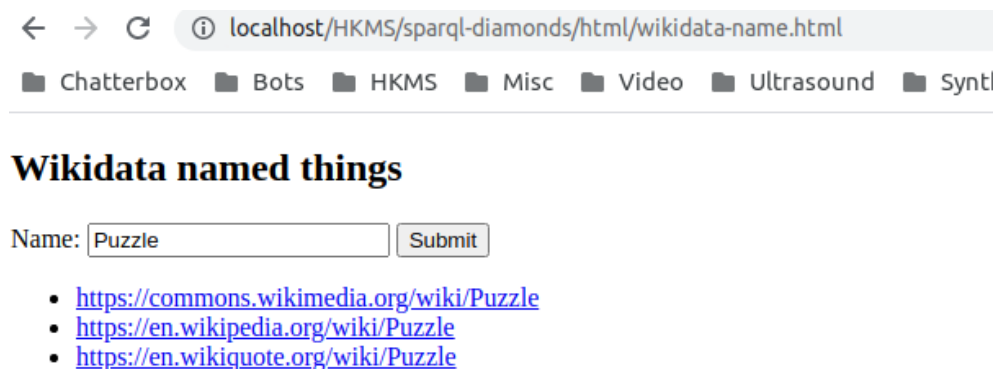
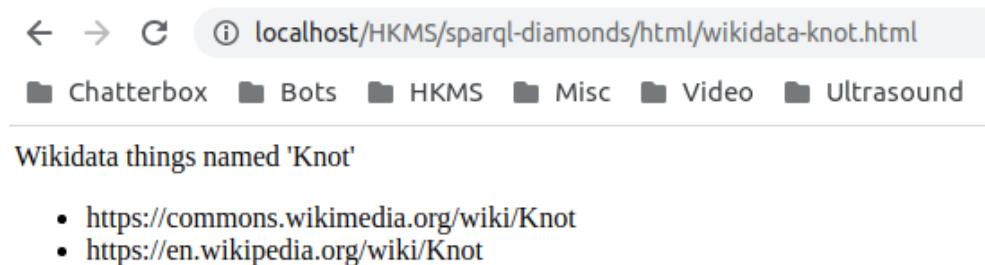
```
SELECT ?x WHERE {  
  ?x rdfs:label "Cat"  
}
```

In contemporary Web browsers, such replacement is trivial – contemporary Javascript supports string replacement natively, but it is extremely easy to make mistakes in such situations, handling text naively, SQL Injection attacks being a case in point. Fortunately libraries have been developed to overcome most mishaps.

A community specification, ‘Moustache’ was used here, so-named for the style of delimiters : `{{ }}`. Alas, exactly the same delimiters are used throughout SPARQL. Fortunately the Hogan `hogan.js` library from Twitter, implementing “logic-less templates” (ie. text search & replace) allows for arbitrary delimiters, so as seen above, the unlikely character sequence of `~{ ... }~` was chosen to avoid conflicts with SPARQL syntax, or confusion in the eyes of the developer.

Implementation

// replace



Alternate Approaches

The SPARQL Diamond is obviously only one of a multitude of possibilities for interfacing with an online data store. One configuration deserves special mention, that of maintaining an RDF model at client. Several RDF API libraries are available [https://www.w3.org/community/rdfjs/wiki/Comparison_of_RD-

FJS_libraries]. This approach offers considerable advantages in terms of flexibility, for example with a SPARQL CONSTRUCT request, the resultant RDF can be merged directly into the local model. However this flexibility is typically at the cost of increased programming complexity.

Another set of solutions are possible by employing a server-side system between the SPARQL server (or other RDF store) and HTTP interface, with the browser being solely for user interface.

// footnote In fact the 'Diamonds' approach described here originated in an experimental server-side content management system [<https://github.com/danja/seki>]. After the 'lightbulb moment' of realising the processing could be shifted to the browser, a simple Wiki was built in this fashion [<https://github.com/danja/foowiki>], with a significant reduction in infrastructure requirements and considerably reduced development and deployment time.

how do other people do it?

Pros and Cons

The benefits of SPARQL Diamonds largely stem from the decoupling of the query and presentation phases of development from the operational wiring. A (very loose) analogy can be made with the Model-View-Controller of object-oriented programming. The model is implied by the query template (and instantiated by the results). The view is defined by the HTML template. The controller is determined by a handful of functions within the page Javascript (calling background libraries) which typically will require little or no change across application scenarios.

- Simplicity : the developer can focus on formulating queries and result presentation
- Largely declarative : the data and its presentation are primarily determined by the templates
- Relatively fast and lightweight in operation
- Compatible with well-known techniques and standard libraries (eg. jQuery)

RDF-unaware : the data structure operated on by the client is that of JSON rather than RDF meaning that graph manipulation techniques aren't available

Most of the time is spent navigating the data

error handling
security

Future Work

Three related avenues for future work are suggested. The first, most apparent and practical, is to hone the implementation as described here. Secondly, it may be useful to explore other potential deployment scenarios in which the SPARQL Diamonds pattern may be useful, such as in server-side systems and mobile device Apps. But perhaps much more importantly is to discuss the situation as alluded to in the Introduction here, whereby ‘regular’ Web programmers encounter incongruities between the techniques they are familiar with and those of Semantic Web technologies. It is reasonable to believe that the identification and documentation of architectural patterns in object-oriented programming served to save many thousands of programmer-hours in that context. Perhaps a similar approach could be useful with Linked Data.

References

[1] Berners-Lee, "Design Issues : Linked Data", 2006. <https://www.w3.org/DesignIssues/LinkedData.html>