Semantic Web & Javascript Disposition

Arne Hassel
Department of Informatics
University of Oslo
arnehass@ifi.uio.no

Fall 2011

Introduction

This document is the disposition to my master thesis. The thesis will be based on a framework that I'll develop alongside writing the thesis. Thus I've decided to describe the framework and its functionalities before tying these aspects to the thesis.

I've also written about applications of the framework to examplify some of its use. These ideas require functionality that I'll have in mind as I write the framework. But to ease the work I'll pick two examples.

These aspects outline the requirements for the framework and thesis. With that in mind I'll try to formalize some goals that are necessary to reach in order to say that the thesis is successful.

I conclude with what I hope to learn and how these accomplishments will contribute to academia.

Contents

1 Applications **2**

2 The framework **3**
2.1 Linking/fetching 3
2.2 Deserializing 4
2.3 The engine 4
2.4 Simple Protocol and RDF Query Language (SPARQL) 4

2.5 Application Programming Interface (API) 5
2.5.1 Documentation 5
2.5.2 Debugging 5
2.6 Storing 5
2.6.1 Working with other sites 5

3 The thesis 5
3.1 Background 6
3.2 Problem Description and Requirements 6
3.3 Tools 6
3.4 Linking/Fetching 6
3.5 Deserializing 6
3.6 The engine 6
3.7 SPARQL 7
3.8 The API 7
3.9 Discussion 7

4 Goals 7

5 Conclusion 8

1 Applications

As I've discussed in my essay, the goal for the framework is to enable developers to work with Semantic Web (SW). The framework will be based on the programming language Javascript (JS), and will first and foremost serve development using that language.

So which applications that makes use of SW are suitable to program in JS? Perhaps another related question is: What data is available to enable us to create an application upon? I could mock up my own data, but I think it's more interesting to work with real world data, as they will reveal better to me the challenges that is in working with open data.

I've taken a look at The Linking Open Data cloud diagram [1] authored by Richard Cyganiak and Anja Jentzsch. It maps open data that is available, linked and formatted with Resource Description Framework (RDF). The latest diagram comprises of 295 datasets [2] and gives an idea of how data is intertwined at current date.

Based on this diagram I've come up with some ideas on fields of interest that could be explored with the help of applications using the framework (ordered alphabetically):

- Maps: An application that fetches map graphics from a map service (e.g. Google Maps) and link the presented view with data fetched from GeoNames (`http://www.geonames.org/`).
- Movies: Based on the Linked Movie DataBase (`http://www.linkedmdb.org/`) it's possible to fetch data about movies, and that could be used to show information to users interested in looking up specific movies.
- Music: Query and search for music-related information with Music Brainz (`http://musicbrainz.org/`).
- Papers: Querying and describing information about publications from names such as Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE), through the RKB Explorer intiative (`http://www.rkbexplorer.com/`).
- Species: Based on the GeoSpecies Knowledge Base (`http://lod.geospecies.org/`), you could fetch data about species, and present that information to users.
- Wiki: Query and find information through DBPedia (`http://dbpedia.org/`).

I believe that working with Maps and Music is the most interesting fields, for two reasons:

- There seems to be a lot of data available, as well as multiple datasets
- I believe that examples are interesting for a lot of people, and therefore easier to present and gain interest

2 The framework

With the applications in mind, which functionality need to be in place? In this section I'll try to answer this question, and my thoughts on how this could and should be handled with my master thesis in mind.

2.1 Linking/fetching

In order to have data that can be processed, that data must be fetched. This part of the framework will ensure that developers can get the data, whether this is transcribed on a local file, on a remote server or fetched through a Linked Open Data (LOD)-point.

It could be interesting to implement Cross-Origin Resource Sharing (CORS) at this level (e.g. by following the tutorial at `http://www.html5rocks.com/en/tutorials/cors/`), so that the framework isn't dependent on other non-JS modules for this part. But I don't think this to be necessary as part of the thesis.

2.2 Deserializing

The data that is fetched, whether it is serialized as RDF/XML, N3, Turtle, or N-triples, needs to be interpreted to a RDF-like format that can be processed by JS. Some work has already been done here, both with libraries (e.g. rdfQuery [8] and js3 [6], which both were discussed in the essay) and with standards (e.g. RDF Interfaces 1.0 [7]).

For this part I'll assume that the developer is familiar with the resource he/she is working with, so that functionality that automatically figures what kind of resource is being processed isn't necessary. This functionality should be implemented in the long-run, but not necessarily as part of the thesis.

2.3 The engine

The framework must be able to process the data, and for this an engine is perfectly suited. Lot of work in SW have already been invested in engines, i.e. reasoners, and I can probably find good pointers in excisting work.

Considering the flora of entailment regimes, the framework will only serve a simple entailment at its core. Hopefully it is possible to create a plugin-interface, that enables other developers to contribute to more advanced entailments regimes.

There is also the possibility that the framework can outsource all of this work to existing frameworks, i.e. there's no need to develop an engine as part of this thesis. For this reason, I reason that an JS-based engine is not required as part of the thesis.

2.4 SPARQL

To query the data it is natural to use SPARQL at its core. The framework will implement the recommended version at first [5], but it would be nice if it also supported the working draft [3] as well.

As with the engine, it's probably possible to outsource this part to existing frameworks.

2.5 API

When the data is "prepared", i.e. fetched and deserialized, it is ready to be processed by the developer. An API must be developed that serves a level of abstraction that connects the developer with the core functionality.

Although not as important as the API itself (when considering the thesis), two other aspects should be noted, namely documentation and debugging.

2.5.1 Documentation

Any good API offers a good documentation. Although the methods may seem intuitive and expressive for me and my supervisors, a documentation should be provided.

2.5.2 Debugging

Considering the difficulties there can be when working with libraries that are yet unfamiliar, it is important to support good tools for debugging. I believe it is outside the scope of this thesis to create any advanced debugging-tools, but at least I can supply some good outputs, that enables developers to debug nicely with existing debugging-tools, such as Firebug for Mozilla Firefox, or the Developer Tools in Google Chrome.

2.6 Storing

Although not initially a part of the framework, developers may be interested to store the data processed in their application. The advantage by doing this with JS could be fast retrieval of stored data, since this can be stored in a format easily processed by JS.

2.6.1 Working with other sites

It would be interesting to see if the framework could work with other services, e.g. by enabling the use of the SPARQL 1.1 Graph Store HTTP Protocol [4]. This approach could enable developers to store data (and is also another reason for why CORS is interesting).

3 The thesis

It is important in a thesis that the user is presented with the necessary information required to understand the content provided. That said, I will assume that the readers have a basic understanding of Information and communication technologies (ICT), i.e. what should be required of a bachelor student in the fields of computer studies.

In addition to the sections that I'll now present, the thesis will also contain a section on acknowledgments, an abstract, an introduction, and a conclusion.

3.1 Background

This section will present the underlying factors behind the thesis, such as SW and JS.

3.2 Problem Description and Requirements

This section will describe the problem description of the thesis, as well as the requirements it will solve during the succeding chapters. This part will probably look somewhat like the part I've written on the framework in this disposition.

3.3 Tools

If another framework is included to support the framework, or if a specific tool with unique functionality have been used, this is the chapter that would describe them.

3.4 Linking/Fetching

This chapter will describe the choices I've made considering the part of the framework that enables the developers to fetch data.

3.5 Deserializing

This chapter will describe how the framework deserializes the fetched data, as well as the format to which it was deserialized. It can be interesting to see which alternatives there are to this format, and I can describe why one is favored before another.

3.6 The engine

This chapter will describe how the engine works, and how it's connected to the rest of the framework.

3.7 SPARQL

This chapter will describe how the framework enables querying with SPARQL.

3.8 The API

I think most of the interesting work will be done in developing the API. Why is one approach more favorable than the other? Is a strict approach more favorable than to a loose one, e.g. can "to much" freedom make it difficult for beginners to use the tool?

3.9 Discussion

This part will be used to discuss the framework, and what that've been discovered in the process of its' development.

4 Goals

In this section I'll describe the goals that must be met so as to say that my master thesis is complete.

The first goal would be to have a functioning framework. I'm thinking of doing this by building a framework that first and foremost act as a proxy for an excisting framework written in another language. I haven't decided yet if I want to go for Jena (Java), RDFLib (Python) or the Perl RDF project (PERL). I'll probably go for one of the two former, since these are programming languages I'm familiar with, but Perl could be interesting since one of my supervisors is very familiar with it.

When a proxy is in place, i.e. a framework that does all that is wanted and required by a RDF framework by routing the functionality of an underlying, mature framework, the first step is complete. This is the absolute minimum that must be produced during my master thesis. This is also the foundation I will build my demonstrating applications.

The following route is twofolded: Building applications that demonstrates the functionality of the framework, and replace proxy-functionality with real functionality. I believe it's impossible to create a full-fledged framework during the course of my master thesis, but I think it could be interesting to see if parts of the framework written in JS is more efficient than the functionality that works as a proxy.

I think the latter route is secondary to the former, and propose the following list as a way to measure progress:

1. A functioning framework that functions as a proxy for a mature framework written in another programming language (this point can be deepened, but I want to wait for the respons from my supervisors first).

2. One or two functioning applications that makes use of excisting, open data.

3. "Switching" parts of the framework from proxy-functions to "real" JS-functions, and measure the difference in efficiency.

5 Conclusion

I think this work will offer some interesting contributions. First of all, it's gonna be interesting to see how much I can build on existing work, and how difficult this is to setup for a developer that wants to do the same. Can I build an API on existing tools, or do I have to reinvent the wheel in JS? How do the tools cooperate? Are they effective? Will it be troublesome to connect them together, and in the end offer an API that easily offers what developers need? I believe this work will reveal some pitfalls, which can be useful in themselves, or even in an abstract form.

References

[1] Richard Cyganik and Anja Jentzsch. The linking open data cloud diagram. `http://richard.cyganiak.de/2007/10/lod/`, October 2007.

[2] Richard Cyganik and Anja Jentzsch. The linking open data cloud diagram 2011-09-09.pdf. `http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19.pdf`, September 2011.

[3] S "Harris and A" Seaborne. Sparql 1.1 query language. `http://www.w3.org/TR/sparql11-query/`, May 2011.

[4] C Ogbuji. Sparql 1.1 graph store http protocol. `http://www.w3.org/TR/sparql11-http-rdf-update/`, May 2011.

[5] E "Prud'hommeaux and A" Seaborne. Sparql query language for rdf. `http://www.w3.org/TR/rdf-sparql-query/`, January 2008.

[6] N Rixham. webr3/js3 - github. `https://github.com/webr3/js3`, November 2010.

[7] N Rixham, M Sporny, M Birbeck, I Herman, and B Adrian. Rdf interfaces 1.0. `http://www.w3.org/TR/2011/WD-rdf-interfaces-20110510/`, May 2011.

[8] J Tennison. rdfquery - rdf processing in your browser - google project hosting. `http://code.google.com/p/rdfquery/`, June 2011.