Semantic Web & Javascript
Disposition

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

Fall 2011

Abstract

This document is the disposition to my master thesis. The thesis will hopefully show how Javascript (JS)-developers can harness the power that is Semantic Web (SW), and perhaps contribute in making both JS and SW more popular among developers.

1 Introduction

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.

2 Applications

The goal for the framework is to enable developers to work with SW. The framework will be based on the programming language 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? To explain this, lets look at the areas which JS excel. JS has been bound for the web since it's conception at Netscape by Brendan Eich. It's been utilized mostly on webpages and rendered by browsers, but have lately been deployed more and more on server-level with framework such as node.js (`http://nodejs.org/`).

As with JS, I believe the future of this framework to be web-based, i.e. enable webpages to include/display data from SW.

Another related question is: What data is available to enable us to create applications 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 large datasets of some significance 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

3 The framework

With the applications in mind, what are the functionality that 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.

I've divided the functionality into five parts: Fetching, Deserializing, The engine, Querying and the Application Programming Interface (API). At this point it's important to note that the use of the word API denotes the set of objects and methods that is available to end-developers, i.e. it doesn't include the (private) set of objects and methods that is available to the framework itself. The latter is denoted as the subsystem, and although there are an array of APIs there also, the use of the word in this setting denotes the public functionality available to the end-developer.

That said, I believe the components of the subsystem (i.e. all the parts besides the API) will be designed per the Bridge pattern, i.e. enabling the two objects to vary independently [3]. The abstraction will define the methods the API will communicate with, while the concrete implementations can be switched from Proxy objects to native SW-components if they become available.

3.1 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.

My strategy for enabling this in the framework is to create an object that creates a connection to an underlying service, run on a local server. This object will tailored using the Proxy pattern, providing a surrogate for the underlying service [3].

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/`), as part of the unit that'll replace the proxy-object. But I don't think this to be necessary as part of the thesis.

3.2 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. In this part I think it's necessary to look at standards such as RDF Interfaces 1.0 [9].

As to how the framework should deserialize data, I'll assume that the developer is familiar with the resource he/she is working with. That rules out functionality that automatically detects what kind of resource is being processed. This functionality should be implemented in the long-run, but not necessarily as part of the thesis.

I've written an essay as part of the master thesis, where I discussed excisting frameworks that enable JS-developers to work with SW. Most prominent of these are rdfQuery [10] and js3 [8]), which may contribute to the framework. But as I've concluded in the essay, I suspect that none of these are mature enough to contribute in any major way.

This suspicion leads me to implement also this part as per the Proxy pattern.

3.3 The engine

The framework must be able to process the data, and for this an engine is needed. Much work have already been invested in engines for SW, 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.

The framework can outsource this work to an underlying service, i.e. there's no need to develop an engine as part of this thesis. This object would also use the Proxy pattern. For this reason, I conclude that a JS-based engine is not required as part of the thesis.

3.4 Querying

For querying the data it's natural to use Simple Protocol and RDF Query Language (SPARQL) at its core. The framework will implement the recommended version at first [7], but it would be nice if it also supported the working draft [4] as well.

As with the engine, it's possible to outsource this part to an underlying service.

3.5 The API

When the data is "prepared", i.e. fetched and deserialized, it is ready to be served to whatever the developer wants to do with it. At this point, an API must be developed that serves an abstraction of the framework.

Based on this thinking, it's natural to conclude that the API will take the form of the Facade pattern, as it provides a unified interface to a set of interfaces in a subsystem [3]. I also think that the API will take form as a Singleton, as I believe this level of simplification is easier for developers to grasp. But this remains to be seen.

The API will be the most important part of the implementation, and will be the center of attention in the thesis. The former parts I've described are what must be in place for this part to work, and I wish to experiment with different approaches in order to get this level right from the start. The hope is that the abstractions of the other parts serve a useful startingpoint for a framework whose parts can be replaced according to the users requirements, and that the API can communicate effortlessly with these nevertheless.

3.5.1 Documentation and debugging

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

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.

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.

3.6 Other functionality

In addition to the five parts I've described, I've taken the liberty to note some other functionality that could be interesting to add to the framework, given time.

3.6.1 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. Another advantage is that the data could be stored and retrieved with the help of Indexed Database API, which is native to JS and designed to satisfy offline-needs in user agents [5].

3.6.2 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 [6]. This approach could enable developers to store data (and is also another reason for why CORS is interesting).

4 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.

4.1 Background

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

4.2 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.

4.3 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.

4.4 The framework

As mentioned, most of the focus will be on the API, but I'll also write some about the other parts of the framework. I feel this is necessary to shine light upon the decissions I've done considering the abstractions.

4.4.1 Fetching

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

4.4.2 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 if one can be favored.

4.4.3 The engine

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

4.4.4 Querying

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

4.4.5 The API

I think most of the interesting work will be done in developing the API. The work will concentrate on this part, and the thesis will take on the different alternatives there are, showing the pros and cons, and tying these to the goal of the framework (to enable JS-developers to work with SW).

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? These, and other design issues will hopefully reveal themselves.

4.5 Discussion

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

5 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. When considering the underlying service, Jena (Java), RDFLib (Python) and the Perl RDF project (PERL) looks most promising. Per ce I'm looking into the latter approach, as Kjetil Kjernsmo (my main supervisor) recommends 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.

6 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? Will the framework introduce something unique into the array of tools available for SW?

I believe this work will reveal some pitfalls, which can be useful in themselves, or even in an abstract form. I also hope this work will be a part of raising the interest of both SW and JS as mature tools for developers, be it proffesional or amateurs.

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] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[4] Steve Harris and Andy Seaborne. Sparql 1.1 query language. `http://www.w3.org/TR/sparql11-query/`, May 2011.

[5] Nikunj Mehta, Jonas Sicking, Eliot Graff, Andrei Popescu, and Jeremy Orlow. Indexed database api. `http://www.w3.org/TR/IndexedDB/`, December 2011.

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

[7] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/, January 2008.

[8] Nathan Rixham. webr3/js3 - github. https://github.com/webr3/js3, November 2010.

[9] Nathan Rixham, Manu Sporny, Mark Birbeck, Ivan Herman, and Benjamin Adrian.

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