Semantic Web & Javascript

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

Spring 2011

Abstract

In this essay I've described Semantic Web (SW) and some of the standards encapsulated. Based on this I've created a list of functionalities required of a Javascript (JS)-library working toward SW. Then I present existing JS-libraries working with these standards, and discuss how they fill certain needs in the world of SW and JS.

I've concluded that the JS-library rdfQuery is the most evolved library as of today.

Contents

Chapter 1

Introduction

Although Semantic Web (SW) is fronted by giants such as Tim-Berners Lee, it has problems gaining hold in the developers community. In my master thesis I will try to assess this problem by creating a library that deliver functionalities desired by developers within the Javascript (JS) development community.

There is a wide range of tools available for developers wanting to plug in to Semantic Web. Most high-level programming languages offer libraries, such as Jena for Java, RDFLib for Python and Redland Ruby Interface for Ruby. But it seems to be no library available for JS that offers a complete experience when working with SW.

This essay is part of the work undertaken on my master thesis. My master thesis will strive to explore the gap between JS-developers and SW, in the process offer a solution, and empower developers to tap into the power that is SW.

First it describes the term SW as well as some of the problem it faces. Next I explain some of the standards developed and their manifestations into the world of developers. Lastly I take on JS and the existing libraries available, dissecting them to reveal their cons and pros.

Chapter 2

Semantic Web

SW is perhaps easiest described as the web for machines. That is not to say that the "other" web (lets call it the human web) are unreadable by machines. After all, everything served by machines breaks down to ones and zeros. But the human web aren't designed for data processing by computers, it is constructed to inform and/or entertain the human end-user.

A lot of effort are put into the human web to maximize output for end-users. As popularity grows, more data are being invested, and semantics expand. Domains of knowledge intersect and new uses are discovered. Developers try to maximize these intersections by constructing endpoints that other developers may access to retrieve and/or exchange information.

Many of these Application Programming Interfaces (APIs) are part of the non-human web in that the data is processed before presented to the user. But they are not part of the Semantic Web. To use any of these APIs you must know the interface and interpret the meaning by human semantics. To be truly semantic, a model must be able to convey meaning that can be constructed by machines.

That means data that are to be part of the SW must not only be able to read by machines, but *the meaning (i.e. semantics)* about the data must also be conveyed.

Chapter 3

Standards

SW offers a wide range of standards, but for the purpose of this essay, I will explain two of them, namely Resource Description Framework (RDF) and Simple Protocol and RDF Query Language (SPARQL).

3.1 RDF

Htizler et.al. explains *Resource Description Framework* as a formal language for describing structured information [15, p. 19]. That means it has no constraints in what information it can describe, as long as its structured, which is ideal for computer consumption. Meta-information can be described alongside regular information, which makes it fit for semantic web.

RDF documents consists of directed graphs, where nodes are objects (or resources, as they are often called) and the edges show a directed relation from one resource to another. Resources are named with Unified Resource Identifiers (URIs), which is a generalization of the perhaps more familiar Unified Resource Locators (URLs). URI are explained in Figure 3.1.

Not all resources in RDF are best described as URIs though, sometimes there are need for absolute values. In those cases, it is more convenient if there is a shared understanding of the values. Literals are designed for this purpose, and can describe numbers such as 42 so that they have the same numeric meaning across all interpretors. Literals have the form "*value*"^URI, where a literal not followed by ^URI means that it is untyped, and interpreted as a string. An example is given in Figure 3.2.

URI have the form *scheme* : [*//authority*]*path*[*?query*][*#fragment*], where the parts in brackets are optional.

- **scheme:** The scheme classify the type of URI, and may also provide additional information on how to handle URIs in applications.

- **authority:** An authority is the provider of content, and may provide user and port details (e.g. arne@semanticweb.com, semanticweb.com:80).

- **path:** The path is the main part of many URIs, though it is possible to use empty paths, e.g., in email adresses. Paths can be organized hierarchically using / as separator.

- **query:** The query can be recognized with the preceding ?, and are typically used for providing parameters.

- **fragement:** Fragments provide an additional level of identifying resources, and are recognized by the preceding #.

Figure 3.1: Shortened explanation on URI given by Hitzler et.al. [15, p. 23].

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/#> .

ex:Everything ex:isAnsweredBy "42"^^xsd:integer .
```

Figure 3.2: A literal serialized in Turtle (TTL).

Lets say we know that Arne regards Bjarne as a friend, and that subjects and objects connected by a friend-property are persons. In addition, lets say that the class Person is a subclass of class Human. This can be serialized as:

```
:arne foaf:friend :bjarne .
foaf:friend rdfs:domain foaf:Person ;
        rdfs:range foaf:Person .
foaf:Person rdfs:subClassOf :Human .
```

Now, can we infer that Bjarne is a human from this? By using forward reasoning, we know that Bjarne is a person (since foaf:Person is range of foaf:friend). We also know that a Person is a Human, so by this reasoning we can say that Bjarne is a human.

Figure 3.3: An example of forward reasoning.

3.1.1 Semantics

In addition to formalize the structuring of data, RDF has a vocabulary of concepts. Listing and explaing these concepts fall outside the scope of this essay, but suffice to say is that they bring a specific set of formal logic. Also, these concepts enable developers the basic tools to create ontologies.

The vocabulary present in RDF are to limiting for most though, and Resource Description Framework Scheme (RDFS) and Web Ontology Language (OWL) exists to provide the capabilities of vocabularies, taxonomies and ontologies [14].

3.1.2 Entailment regimes

RDF delivers a way of entailment, which means a way to deduce knowledge. It is one of many entailment regimes that can be applied to infer knowledge. Other regimes are simple entailment, RDFS, D-Entailment, OWL 2 RDF-Based Semantics, OWL 2 Direct Semantics, and RIF Core (all but first are explained in SPARQL 1.1 Entailment Regimes [9]).

Based on these entailment regimes, knowledge can be inferred. This is done by forward and backward reasoning. Reasoning engines can implement one or a hybrid of the two, and use them to answer the validity of a statement.

Forward reasoning works by inferring knowledge on the given data, and keep going at it until the statement deduced. An example is given in Figure 3.3.

Backward reasoning works by inferring knowledge from the given statement that is to be checked. When a set of statements are found that match the given data, the reasoning engine can conclude that the statement is valid.

For this example we use the same data as in Figure 3.3. Now we want to know if Arne is a human. By using backward reasoning, we try to find if there are data that support this statement, and we note that in order for it to be true, we should find something connected to the class Human. We see that Person is a subclass of Human, and further that Person is domain of friend. Now we see that Arne is the domain of :Arne foaf:friend :Bjarne, which enables us to validate the validity of Arne being a human.

Figure 3.4: An example of backward reasoning.

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:ex="http://example.com/">

  <rdf:Description rdf:about="http://example.com/arne">
    <foaf:knows>
      <rdf:Description rdf:about="http://example.com/
          bjarne">
      </rdf:Description>
    </foaf:knows>
  </rdf:Description>
</rdf:RDF>
```

Figure 3.5: An example of RDF/XML.

An example is given in Figure 3.4.

3.1.3 Serializations

To exchange the data we need to serialize the information modeled.

The RDF/XML Syntax Specification became a W3C Recommendation February 10th 2004. Extensible Markup Language (XML) is the most common serialization of RDF [15], and delivers complete coverage of the RDF data model. XML is hierarchy-based, while RDF is graph-based, but that can be solved. More problematic is the dis-allowance of colons in attribute-values, which forces us to write the full URI for a given subject (see Figure3.5).

When we compare Figure 3.5 to Figure 3.6, we see that triples can be written easier (not to say read and understood easier).

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .

ex:arne foaf:knows ex:bjarne .
```

Figure 3.6: An example of Turtle.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.com/>
SELECT ?persona ?personb
WHERE {
        ex:Arne foaf:knows ?personb .
}
```

Figure 3.7: An example of SPARQL. This query will return a list of all persons Arne knows.

The serialization used in 3.6 is Turtle, which have no standard as of yet (although a W3C Team Submission was submitted March 28th 2011 [13]). The language has gotten hold in the SW community, and has become a part of another standard, namely the standard describing SPARQL. Which leads us to next section.

3.2 SPARQL

SPARQL is a query-language for data structured in RDF. It has an Structured Query Language (SQL)-like structure that enables data to be retrieved from triple stores, either as result sets or graphs. The endpoint queried does so by matching the statements given in the query.

The SPARQL standard supports the queries SELECT, ASK, CONSTRUCT and DESCRIBE. The two former returns result sets, while the two latter returns graphs. Examples are given in Figure 3.7.

SPARQL became a W3C recommended standard January 15th 2008, but an update is on its way. It adds functionality such as aggregates, subqueries, property paths, and more [11], but is also a part of a larger set of documents that each extend SPARQL in their own way. Perhaps the most exciting to mention are SPARQL 1.1 Update [12] and SPARQL 1.1 Graph Store HTTP Protocol [10], that respectively allows manipulating data with SPARQL and a way to exchange SPARQL queries RESTfully.

Chapter 4

Javascript

JS is a beloved child with many names. In its inception it was named Mocha, later renamed LiveScript and finally to Javascript. But it is also known as JScript, and was standardized by Ecma International with ECMAScript [5]. It quickly gained traction for its easy inclusion into webpages, but was long rediculed and misunderstood by developers [1].

That changed with Asynchronous Javascript and XML (AJAX), which reinvigorated the developers interest into what this prototype-based, object-oriented scripting language, that also is considered a functional programming language [4].

4.1   Libraries

There have been several attempts at creating JS-libraries that enable developers to interface data from SW. I've described some of the features of SW, and list some required of wanted functionalities in Figure 4.1. Based on this list, I present my results in Table 4.1.

4.1.1   JS3

JS3 was first committed November 19th, 2010 and last updated three days later. It describes itself as "An insane integration of RDF in ECMAScript-262 V5" [7]. It sports an API for manipulating RDF values and to some extent graphs. But it has no parsing, reasoning or querying capacities.

4.1.2   Jstle

Jstle was first committed April 21st, 2010, and last updated three days later. It describes itself as "Jstle is a terse javascript RDF serialization language"

Figure 4.1: A list of functionalities required or sought after in JS-libraries working with SW.

[6]. It's a proof of concept, and seems to provide a Turtle-like representation of RDF in JS. But no support for parsing, reasoning or querying.

4.1.3   rdfQuery

The rdfQuery-project has been going strong since October 17th, 2008. It was last updated on June 24th, 2011. It states "rdfQuery is an easy-to-use JS-library for RDF-related processing" [8].It has a representation of RDF and also sports an API that enables developers to manipulate it. It parses RDF/XML as well as RDFa, and some ability to parse RDF/JSON. It also offers a plugin that enables the insertion of rules, that might allow reasoning and entailment. It doesn't support querying with SPARQL, but offers its own API.

4.1.4   Simple javascript RDF parser and query thingy

The development of the Simple javascript RDF parser and query thingy seems to be around November 5th, 2005, which makes it the oldest library I've found working with RDF. It's latest version came out May 25th, 2006, and it doesn't seem to have any big usage. It supports loading and parsing of RDF/XML-documents, and a crude API for querying.

Capability Name	Representation API	Parsing			Reasoning			Querying API
		XML	JSON	RDFa	RDF	RDFS	OWL	
JS3	1							
Jstle	1							
rdfQuery	1	1	1^1	1	1^2	1^2	1^2	1
Simple ...		1						1^3
Tabulator ...		1						

1. rdfQuery supports parsing RDF/JSON, but cannot serialize it quite so well (data is lost in the process)

2. rdfQuery doesn't support any entailment regimes out of the box, but it seems possible to add rules with the RdfRulesPlugin

3. The Simple javascript RDF parser and query thingy has a very crude functionality for querying

Table 4.1: The results obtained in this essay

4.1.5 Tabulator RDF Parser

The Tabulator RDF Parser is part of the Tabular project [2], which is a generic data browser [3]. It seems to be able to parse RDF/XML quite well, but offers no other functionalities.

4.2 Results

Table 4.1 shows the analysis summarized. I've omitted the columns on parsing of Turtle and N3 as well as the capacity of using SPARQL to query, since no libraries analyzed supported that anyway.

It's fair to say that rdfQuery is the best candidate for developing with RDF so far.

Chapter 5

Conclusion

In this essay I've explained the basics of Semantic Web. Based on the features I've tried to itemize a list of functionalities that should be handy when working with a JS-library. I've analyzed five different libraries, and concluded that rdfQuery seems to be the most likely candidate to work with in the future.

As fodder for further research, I think a more structured analysis of the functionalities required is necessary, perhaps analyzing libraries in other programming languages such as Jena for Java.

I also think that a further research into APIs is required, and think that analyzing several JS-libraries, ranging from once-popular and popular-and-still-going-strong.

Bibliography

[1] Javascript: The world's most misunderstood programming language. `http://www.crockford.com/javascript/javascript.html`, January 2001.

[2] An introduction and a javascript rdf/xml parser. `http://dig.csail.mit.edu/breadcrumbs/node/149`, July 2006.

[3] Tabulator: Async javascript and semantic web. http://dig.csail.mit.edu/2005/ajar/release/tabulator/0.7/tab.html, January 2006.

[4] Douglas crockford on functional javascript. `http://www.blinkx.com/watch-video/douglas-crockford-on-functional-javascript/xscZz8XhfuNQ_aaVuyUB2A`, October 2008.

[5] Javascript creator ponders past, future. `http://www.infoworld.com/d/developer-world/javascript-creator-ponders-past-future-704`, June 2008.

[6] dnewcome/jstle - github. `https://github.com/dnewcome/jstle`, April 2010.

[7] webr3/js3 - github. `https://github.com/webr3/js3`, November 2010.

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

[9] Sparql 1.1 entailment regimes. `http://www.w3.org/TR/sparql11-entailment/`, May 2011.

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

[11] Sparql 1.1 query language. `http://www.w3.org/TR/sparql11-query/`, May 2011.

[12] Sparql 1.1 update. `http://www.w3.org/TR/sparql11-update/`, May 2011.

[13] Turtle - terse rdf triple language. `http://www.w3.org/TeamSubmission/turtle/`, March 2011.

[14] J Hebeler, M Fisher, R Blace, and A Perez-Lopez. *Semantic Web Programming*. Wiley Publishing, Inc., 2009.

[15] P Hitzler, M Krotsch, and S Rudolph. *Foundations of Semantic Web*. Chapman & Hall/CRC, 2010.