SSWS 2011

RDF Literal Data Types in Practice

Ian Emmons, Suzanne Collier, Mounika Garlapati, and Mike Dean

Raytheon BBN Technologies, Inc., Arlington, VA 22209, USA
`{iemmons,scollier,mgarlapa,mdean}@bbn.com`

Abstract. One of the more mysterious aspects of RDF (Resource Description Framework) is typed literals. For instance, confusion over the difference between a plain character string ("foo") and a string that is explicitly typed ("foo"^^xsd:string) is common. Also, questions often arise about comparisons between literals of the various numeric types (e.g., long, integer, decimal, and float). This paper explores how several popular triple stores handle literals via direct testing, and also compares their behavior to the relevant standards. Along the way, we highlight a number of implementation inconsistencies and some surprising aspects of the standards themselves.

1 Introduction

One of the more mysterious aspects of RDF (Resource Description Framework) [14] is typed literals. For instance, new and even moderately experienced Semantic Web practitioners often ask questions like these:

– What is the difference between a plain character string ("foo") and a string that is explicitly typed ("foo"^^xsd:string)?
– Under what conditions can literals of the various numeric types (long, integer, decimal, float, etc.) be compared?
– Can literals with different lexical forms but the same value, such as "47"^^xsd:decimal and "47.0"^^xsd:decimal, be compared?

While using our own Parliament triple store [1], we have wrestled with these questions from time to time, and so decided to investigate this topic thoroughly. In particular, we wanted to understand how other triple stores implement literal data types, and compare this against both our own implementation and the standards themselves. We give a summary of the relevant portions of the standards in Section 2, and then present our empirical results in Section 3. Along the way, we highlight a number of implementation inconsistencies and some surprising aspects of the standards themselves.

2 Literal Data Types in the Standards

All literals in RDF have a lexical form encoded as a Unicode string, and are either typed or plain [14]. A plain literal is a lexical form with an optional language tag ("foo" or "foo"@en), whereas a typed literal is a lexical form together with a data type URI ("foo"^^xsd:string). A data type defines a value space, the set of possible values, and a lexical space, the set of valid lexical forms, for literals of that type. RDF defines a set of data types by borrowing from the XSD specification [3], and most typed literals use one of these data types. Figure 1 shows the complete XSD type hierarchy. RDF defines one other data type, rdf:XMLLiteral, and also allows user-defined data types.

In addition, some RDF serializations such as Turtle [2] allow unquoted literal forms. In reality, these are not separate kinds of literals, but syntactic shortcuts for certain typed literals. For instance:

– An unquoted 3 is a shortcut for "3"^^xsd:integer
– An unquoted 3.14 is a shortcut for "3.14"^^xsd:decimal
– An unquoted true is a shortcut for "true"^^xsd:boolean

Because SPARQL syntax is based on Turtle, these shortcuts are valid within SPARQL queries [20].

In the absence of entailment, RDF and OWL maintain that literals are only equal when both their lexical form and data type are equivalent. The RDF specification states that two literals are equal if and only if all of the following conditions hold [14]:

– The two lexical forms compare equal, character by character
– Either both or neither have language tags
– The language tags, if any, compare equal
– Either both or neither have data type URIs
– The two data type URIs, if any, compare equal, character by character

This means that any two typed literals must have exactly the same lexical form and data type URIs to be considered equal. OWL follows similar equality rules, declaring "Two literals are structurally equivalent if and only if both the lexical form and the data type are structurally equivalent; that is, literals denoting the same data value are structurally different if either their lexical form or the data type is different." [15]

Although these are the strict rules of literal equality, by applying the RDF D-Entailment regime to XSD data types, the equality rules become more flexible [11]. Note that this entailment regime applies only to XSD

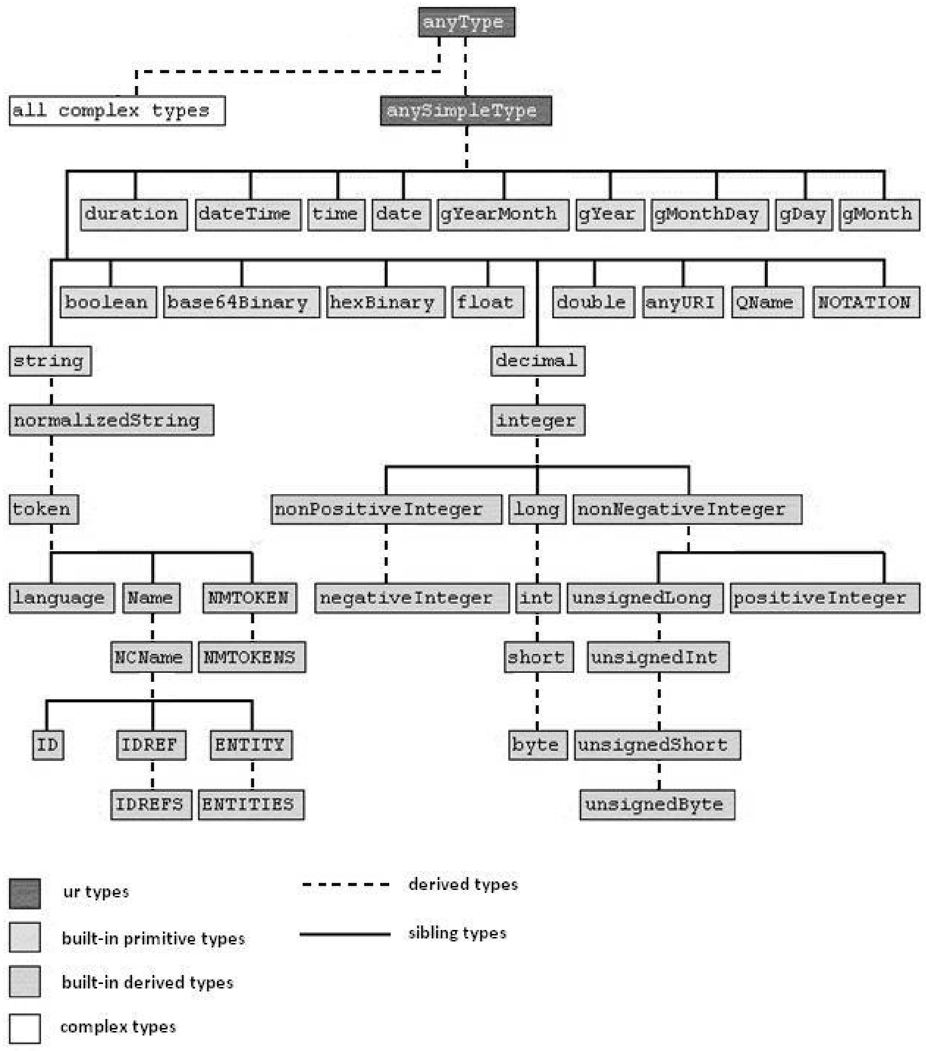

Fig. 1. Type Hierarchy of XSD Data Types [3]

data types, and adds nothing to comparisons of literals with language tags. Thus this paper will not discuss language-tagged literals any further. XML schema data types are defined in a hierarchical structure, with base types and derived types (see Figure 1). An XML schema derived type refers to a subset of the value space of its base type. Therefore, two literals that have the same primitive base data type and the same lexical forms are equal [3]. For example, because both int and byte are derived from decimal, "25"^^xsd:byte is equivalent to "25"^^xsd:int. Additionally, RDF Semantics explicitly equates plain literals ("foo") and literals of type string ("foo"^^xsd:string):

> The value space and lexical-to-value mapping of the XSD data type xsd:string sanctions the identification of typed literals with plain literals without language tags for all character strings which are in the lexical space of the data type, since both of them denote the Unicode character string which is displayed in the literal [11].

The D-entailment regime also states that if two lexical forms map to the same value and have the same data type, then they entail each other [11]. For example, "14"^^xsd:decimal and "14.0"^^xsd:decimal are lexicographically different; however, because 14 and 14.0 map to the same value, these two literals are equivalent under this entailment rule.

Within SPARQL Filter clauses, additional type promotion rules apply, since the functions and operators are defined by the XML Query Language (XQuery) Operator Mapping. XQuery performs type promotion and subtype substitution as necessary in order to compare the values of operands separately from the data types [4]. The outcome (as seen in Section 3.3) is potentially different results for the following two SPARQL queries:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?x WHERE {
   ?x ?y "47"^^xsd:decimal .
}

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?x WHERE {
   ?x ?y ?z .
   FILTER( ?z = "47"^^xsd:decimal )
}
```

There is considerable opportunity for confusion here: We found that even Raytheon BBN's most experienced Semantic Web practitioners were surprised to learn that these queries are not equivalent.

Implementations of the recommendations are not required to support D-Entailment on XSD data types, but it is helpful. D-Entailment helps to reduce the unintended effects of syntactic choices made by authors, and makes data more interoperable. For instance, authors may choose to represent 12 as a decimal instead of an integer. However, when another user queries for "12"^^xsd:integer they would expect to find a match. Of the triple stores that we studied, Jena most strictly followed the RDF D-Entailment regime on XSD data types. AllegroGraph chose to only follow the string entailment rules, equating "foo" with "foo"^^xsd:string, but not allowing equivalence between different numeric types. Parliament, AllegroGraph, and Jena TDB follow strict literal equality. All of the triple stores we tested follow the SPARQL filter clause type promotion rules.

3 Existing Practice

An analysis of literal data types in relation to triple stores is important because most triple stores lack documentation on their storage and treatment of literals. Some users may expect triple stores to treat various literal forms (for example "foo"^^xsd:string versus "foo" or "3.14"^^xsd:decimal versus "3.14"^^xsd:float) as equivalent values. However, triple stores vary in their implementation. To analyze the relationship between literals and triple stores, we stored various literal representations in several different triple stores, and then queried for them to see what combinations of literals each triple store would match.

3.1 Methodology

In order to assess how triple stores handle a diverse set of literals, we created a test harness to drive each of several triple stores through a series of tests and compile the results. The particular triple stores we selected for testing are discussed in Section 3.2, and the results of our testing are discussed in Section 3.3. The harness stores a set of literal statements in a triple store and then runs a series of queries to assess which pairs of literal forms that triple store can match. Our test harness code and a spreadsheet containing our results can be found on our web site, here:

`http://asio.bbn.com/2011/10/ssws/LiteralDataTypes.zip`

To begin, the test harness reads RDF from an input file that contains a consistent selection of typed, untyped, and irregular literals. (By "irregular", we mean typed literals whose lexical form are inconsistent with their type, such as "foo"^^xsd:integer. Such literals are interesting because they are valid RDF, even though most people would regard them as an error.) The input file containing these literals was in Turtle format when possible, but we used N-Triples for triple stores that do not support Turtle.

Based upon a W3C compilation of XSD data types and their mappings and relevance to the standards RDF, OWL, SPARQL, and RIF [23], we chose a sampling of XSD data types to test. The literals we tested fell into these categories:

- Strings: both plain and typed
- Numbers: decimal, integer, long, int, double, and float
- Dates and times: dateTime, date, time, and gYear
- Miscellaneous: anyURI, hexBinary, base64Binary, and boolean
- Irregular: Literals whose lexical form is not within the lexical space of their data type, such as "foo"^^xsd:integer

The test harness adds one literal from the input file at a time to the triple store under test, and then a preselected set of simple queries and filter queries are run against the literal. By *simple query,* we mean a query whose where clause consists of a single triple pattern whose object is the literal being tested. In contrast, a *filter query* is a query whose where clause consists of a single triple pattern with a variable in the object position, along with a filter that restricts that variable to be equal to the literal under test. Examples of simple and filter queries (specifically querying "47"^^xsd:decimal) can be seen in Section 2. After the queries are run on the current literal statement, the triple store is cleared and the process is repeated until all the literal statements in the input file have been tested. For each query, if its result set is non-empty, then the spreadsheet cataloging the results indicates this with a "Yes," while "No" indicates an empty result set.

3.2 Software Tested

We tested the literal handling behavior of five triple stores: Jena's in-memory store, Jena TDB, Parliament, AllegroGraph, and OWLIM.

Jena In-Memory: Jena is an open source Java framework for building Semantic Web applications [13]. It can read and write RDF in many file formats, and it also includes a SPARQL query processor called ARQ. Its highly layered and pluggable architecture makes it an ideal front end for other triple stores as well — Mulgara, Virtuoso, OWLIM, AllegroGraph, and Parliament can all use Jena in this fashion. When used on its own, it provides an easy-to-use and well documented in-memory triple store. Jena's in-memory store accepts irregular literals.

Jena TDB: TDB is an optional subsystem of Jena for persisting RDF and OWL data that allows for high performance and large scale storage and query [13].

Parliament: Parliament is an open source, high performance, and standard compliant triple store for the Semantic Web, written by Raytheon BBN Technologies [1]. It pairs Jena's query processor with an innovative back-end store, and customizes Jena's query processor to optimize queries so as to derive maximum benefit from the back-end's data organization. For this paper in order to meaningfully derive how each triple store uniquely handles literals, the Jena API/interface was not used in conjunction with any other triple store other than itself, Jena TDB and Parliament. The test harness loaded Parliament from a Turtle file.

AllegroGraph: AllegroGraph is a high performance database and application framework for the Semantic Web from Franz [8]. It supports various clients (Python, Java, Jena, Lisp, Ruby, etc.) and has well documented examples and tutorials for implementation. AllegroGraph can read and write RDF in RDF/XML and N-Triples file formats but not Turtle. For this reason we were unable to add unquoted literals to AllegroGraph. AllegroGraph also will not accept irregular data types such as "foo"^^xsd:integer. Consequently, there are some blank cells in the AllegroGraph column of the test results.

OWLIM: OWLIM is a triple store from Ontotext [16]. Just as Parliament uses Jena as a front end and query processor, OWLIM uses another popular Java framework for Semantic Web applications called Sesame [18]. There are several versions of OWLIM available with varying levels of scalability and price points. OWLIM-Lite was used in this study in conjunction with the Sesame library and the SPARQL query language. It supports many file formats, including Turtle, used here. OWLIM cannot accept irregular literals such as "foo"^^xsd:integer.

Other Triple Stores: We chose the triple stores above because of their popularity, free availability, and relevance to our work here at BBN. We also tried to achieve a diversity of query processors, while still emphasizing Jena because it is used in our own triple store, Parliament. There are several other well-known triple stores that we would like to test, but did not due to time constraints. These include (but are not limited to) Virtuoso, Mulgara, and Oracle.

3.3 Analysis of Results

This section gives an overview of the results of our testing. Our complete results can be found on our web site, here:

`http://asio.bbn.com/2011/10/ssws/LiteralDataTypes.zip`

Keep in mind that Jena and Parliament accept all literals. AllegroGraph does not support Turtle syntax, and therefore does not accept unquoted literals (47, true). Also, OWLIM and AllegroGraph do not accept irregular literals (e.g., “foo”^^xsd:integer). The empty blanks in the spreadsheet are indicative of these limitations.

Most of the literal types chosen are siblings in the inheritance hierarchy, but the types decimal, integer, long, and int form an inheritance chain. The following sections will explain how each category of literals (numbers, strings, times, miscellaneous, and irregular) was handled by the various triple stores.

Numeric Types: With the numeric types (float, double, decimal, and all the types derived from decimal), the following kinds of numeric comparisons are of particular interest:

1. Literals that match exactly, according to the strict rules of RDF without entailment
2. Literals with identical lexical forms but differing types that share a common primitive base data type, e.g., decimal and integer
3. Literals with identical lexical forms but differing types that do not share a common primitive base data type (most often, this means sibling types), and yet whose types are intuitively compatible, e.g., float and integer
4. Literals with identical types but differing lexical forms that map to the same point in value space, e.g., “47”^^xsd:long versus “+47”^^xsd:long

Table 1 summarizes the results for numeric literals in terms of these categories. (The numbers in the table correspond to the categories of comparisons in the list above.)

Strings: The key question with strings is which triple stores see plain literals (“foo”) and typed strings (“foo”^^xsd:string) as equivalent. Table 2 summarizes how strings are handled by the various triple stores.

Triple Store	**Query Type**	**Matched Relationships**	**Unmatched Relationships**
Jena in-memory	Simple	1, 2, 4	3
Jena TDB, AllegroGraph, Parliament, OWLIM	Simple	1	2, 3, 4
Jena in-memory, Jena TDB, AllegroGraph, Parliament, OWLIM	Filter	1, 2, 3, 4	

Table 1. Comparison Results for Numeric Literals

Triple Store	**Query Type**	**Typed String versus Plain Literal**
Jena in-memory, AllegroGraph	Both (simple and filter)	Match
Jena TDB, Parliament, OWLIM	Simple	No Match
Jena TDB, Parliament, OWLIM	Filter	Match

Table 2. Comparison Results for Strings

Temporal Types: The temporal XSD types (dateTime, date, time, and gYear) are treated by all the triple stores in the same manner. For all triple stores, and in both simple and filter queries, only exact matches are found.

anyURI: XSD type anyURI acts much like the temporal data types in that only exact matches result in a positive comparison, except in one case: Jena's in-memory store finds matches between anyURI literals and strings (either typed or plain) in simple queries only. Within a filter, anyURI literals and strings do not match in any triple store.

hexBinary and base64Binary: Across the board, the XSD types hexBinary and base64Binary compare equal only in the case of exact matches. Interestingly, a hexBinary literal will not compare equal to the base64Binary literal that represents the same octet sequence. For example, 1111 in hex converts to ERE= in base64, but "1111"^^xsd:hexBinary does not match "Ere="^^xsd:base64Binary in any triple store. This is due to the fact that hexBinary and base64Binary are sibling data types and do not derive from a common primitive base data type.

Boolean: All triple stores match unquoted booleans (true) and typed booleans ("true"^^xsd:boolean) in both simple and filter queries (except AllegroGraph as it does not accept unquoted literals). This is not a surprise, because unquoted booleans are simply a syntactic shortcut for typed booleans in Turtle. A more interesting result is that no triple store matches unquoted or typed booleans against typed or plain string literals with the same lexical form (e.g., “true”^^xsd:string and “true”).

Irregular Literals: OWLIM and AllegroGraph do not recognize and will not accept irregular data types such as “foo”^^xsd:integer. With other triple stores, such literals compare equal in the case of exact matches. The more interesting result is what happens when comparing an irregular literal against a string (either typed or plain) with the same lexical form. Table 3 shows what happens in this case.

Triple Store	**Query Type**	**Typed or Plain String versus Irregular Literal**
Jena in-memory	Simple	Match
Jena in-memory	Filter	No Match
Jena TDB, Parliament	Both	No Match
AllegroGraph, OWLIM	Both	N/A — unable to store irregular literals

Table 3. Comparison Results for Irregular Literals

Overall in our tests, Jena’s in-memory store stood out as the most accepting of types that displayed a conceptual similarity (e.g., “foo” versus “foo”^^xsd:string or “47”^^xsd:decimal versus “47”^^xsd:integer) for both simple queries and filter queries.The Parliament triple store is most consistent with the way in which AllegroGraph and OWLIM treat literals in that these three triple stores are less permissive for most conceptually similar data types.

4 Related Work

Most of the related work in this domain is in the language specifications themselves [14] [24] [11] [20]. Because these specifications can be complex, the W3C produced a Working Group note, “XML Schema Datatypes in RDF and OWL” [6], which clarifies typed literal equality via the D-entailment regime.

Other work emphasizes the importance of using D-entailment in dealing with inconsistencies of instances [19] and specifics of the entailment regimes [12]. Many triple store studies have also been completed; however, they have mostly focused on performance [21] [22]. Although not directly targeted at RDF data types, Garcia-Castro and Gomez-Perez explored the interoperability of semantic web technologies using OWL. They created an interoperability benchmark to evaluate tools and they concluded that interoperability between Semantic Web tools is very low [9].

5 Conclusions

The treatment of typed literals by triple stores is fairly consistent, with the exception of Jena's in-memory store. These triple stores differ in their treatment of conceptually similar types for a simple query. Jena TDB, Parliament, OWLIM, and AllegroGraph implement strict literal equality. The Jena developers, on the other hand, stated "the formal semantics of both RDF and OWL makes it clear that entailment is a core feature of the Semantic Web recommendation" [5]. Consequently, the Jena in-memory model uses the D-Entailment regime to equate literals with derived types [10]. Jena also implements the D-entailment rule which equates literals with identical types and values but different lexical forms. Additionally, both Jena's in-memory store and AllegroGraph equate typed and untyped strings in simple queries. It is surprising that the AllegroGraph developers decided to implement D-Entailment for strings, but not for numeric types. Jena's in-memory store is also more permissive with the XSD data type anyURI, when matched against string type in simple queries. Interestingly, Jena's in-memory store matches irregular literals with strings with the same lexical form. This seems to deviate from the D-entailment rules. All triple stores followed the same conventions for temporal, hexBinary, base64Binary and boolean types. Overall, Jena follows D-Entailment, allowing more flexibility in user input, while the other triple stores tend to follow strict literal equality.

It is also crucial to add that in contrast to simple queries, filter queries recognize sibling and derived type relationships in query results for most triple stores. This indicates that filter queries have a more permissive path in parsing derived and sibling types. As defined by SPARQL [20], filter queries use the XML Query language (XQuery) type promotion [4]. This results in differences in query responses between apparently equivalent filter queries and simple queries.

There are several possibilities for expanding on this research in the future. The most obvious is to expand the testing to include more triple stores (such as Virtuoso, Mulgara, and Oracle) as well as SPARQL front ends to relational databases (such as D2RQ, Revelytix' Spyder, and BBN's Asio). Another possibility is to profile the use of typed literals in various subsets of the Semantic Web community to better understand how aspects of the type system affect real users. Some work has been done in this regard based on the billion triples challenge corpus from ISWC 2008 [7], but a more in-depth look at current data is warranted.

Finally, it would be interesting to better understand how the disparity between literal matching in simple queries and filter queries affects query optimizers. For instance, a filter that compares a variable for equality to a known URI can usually be optimized away by directly substituting the URI in place of the variable throughout the rest of the query. However, we have shown that a filter testing for equality to a literal cannot be so easily optimized away. This may impact how developers write their queries and improve overall performance time.

As evidenced by semantic tools research [9] and our studies, semantic technologies lack interoperability due to conflicting standards and developer design decisions. In order for the semantic web scalability to become a reality, interoperability is essential. The design decisions made by developers not only effect the results users receive via queries, but the performance of their system. OWLIM specifically states in their documentation that they do not use the D-entailment regime because the performance penalty is too high [17]. In further studies we would like to use larger data sets to investigate how the triple stores implementation of RDF data type equality effects their scalability. This data will help users decide which triple store to select based on performance and treatment of RDF literals.

Differing standards in the treatment of typed literals pose issues for users who expect conceptually similar types to match in their queries. Also the triple stores we studied differed greatly in their RDF implementation, yet their design decisions were undocumented. The different implementations lead to varying results, and it is pertinent that users know what they are using. As the semantic web becomes an increasingly popular framework for users, syntactic differences in typed literals are inevitable. It is critical that the community be aware of the inherent differences in the treatment of typed literals in order to promote correctness and maintainability of implementations, uniformity of practice, and simplicity of standards.

References

1. BBN Technologies: Parliament, `http://parliament.semwebcentral.org/`
2. Beckett, D., Berners-Lee, T.: Turtle — Terse RDF Triple Language, `http://www.w3.org/TeamSubmission/turtle/`
3. Biron, P.V., Malhotra, A. (eds.): XML Schema Part 2: Datatypes Second Edition. W3C (October 2004), `http://www.w3.org/TR/xmlschema-2/`

4. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J. (eds.): XQuery 1.0: An XML Query Language (Second Edition). W3C (December 2010), `http://www.w3.org/TR/xquery/`
5. Carroll, J.J., Dickinson, I., Dollin, C.: Jena: Implementing the Semantic Web Recommendations. Tech. rep., HP Laboratories Bristol (December 2003), `http://www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf`
6. Carroll, J.J., Pan, J.Z. (eds.): XML Schema Datatypes in RDF and OWL. W3C (March 2006), `http://www.w3.org/TR/swbp-xsch-datatypes/`
7. Dean, M.: Toward a Science of Knowledge Base Performance Analysis. In: Invited Talk, 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008). p. 20. Karlsruhe, Germany (October 2008), `http://asio.bbn.com/2008/10/iswc2008/mdean-ssws-2008-10-27.ppt`
8. Franz, Inc.: AllegroGraph, `http://www.franz.com/products/allegrograph/`
9. Garcia-Castro, R., Perez, A.G.: Interoperability results for Semantic Web technologies using OWL as the interchange language. Journal of Web Semantics: Science, Services and Agents in the World Wide Web November, 278–291 (2010)
10. Glimm, B., Ogbuji, C. (eds.): SPARQL 1.1 Entailment Regimes. W3C (May 2011), `http://www.w3.org/TR/sparql11-entailment/\#DEntRegime`
11. Hayes, P. (ed.): RDF Semantics. W3C (February 2004), `http://www.w3.org/TR/2004/REC-rdf-mt-20040210`
12. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Web Semantics: Science, Services and Agents on the World Wide Web 3, 79–115 (2005), `http://www.websemanticsjournal.org/index.php/ps/article/download/66/64`
13. HP Labs Semantic Web Research: Jena, `http://www.hpl.hp.com/semweb/`
14. Klyne, G., Carroll, J. (eds.): Resource Description Framework: Concepts and Abstract Syntax. W3C (February 2004), `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`
15. Motik, B., Patel-Schneider, P.F., Parsia, B. (eds.): OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C (October 2009), `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`
16. Ontotext: OWLIM, `http://www.ontotext.com/owlim/`
17. Ontotext: Primer Introduction to OWLIM, `http://owlim.ontotext.com/display/OWLIMv41/Primer+Introduction+to+OWLIM`
18. OpenRDF: Sesame, `http://openrdf.org`
19. Polleres, A., Hogan, A., Harth, A., Decker, S.: Can we ever catch up with the web. Semantic Web Journal 1, 45–52 (2010)
20. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`
21. Revelytix, Inc.: Triple store evaluation analysis report. Tech. rep., Revelytix, Inc. (September 2010), `http://www.revelytix.com/sites/default/files/TripleStoreEvaluationAnalysisResults.pdf`
22. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triplestore technologies for large data stores. In: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops. pp. 1105–1114. Springer, Vilamoura, Portugal (2007), LNCS 4806
23. W3C: XSD Datatypes (June 2011), `http://www.w3.org/2011/rdf-wg/wiki/XSD_Datatypes`
24. W3C OWL Working Group (ed.): OWL 2 Web Ontology Language Document Overview. W3C (October 2009), `http://www.w3.org/TR/owl2-overview/`