Flat triples approach to RDF graphs in JSON

Dominik Tomaszuk
Institute of Computer Science, University of Bialystok, Poland

Abstract. This paper describes a syntax that can be used to write Resource Description Framework (RDF) graphs for presentation and editing purposes. It propose a alternative mean of serializing RDF triples using JavaScript Object Notation (JSON), a lightweight representation format which emphasizes legibility and brevity. RDF/JSON is a textual syntax for RDF that allows RDF graphs to be completely written in a compact form. This means the format is quick and easy to read and write. This new serialization format is developed in response to a demand from a wide range of users, who do not have an XML, Notation3 and Turtle background.

Keywords. Semantic Web, Resource Description Framework (RDF), JavaScript Object Notation (JSON), relational model

1. Introduction

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is effortless for humans to read and write. It is easy for machines to parse and generate. JSON is based on a subset of the [2]. JSON is a text format that is completely independent from programming languages. These properties make JSON a perfect data-interchange language.

JSON [3] is built on two structures: a collection of name/value pairs and an ordered list of values. In most languages, collection of name/value pairs is realized as an object, struct, record, hash table, associative array, dictionary, or keyed list. In various languages, ordered list of values is realized as an array, list, vector, or sequence.

JSON's basic types are numbers, strings, booleans, arrays, object and null. Number in JSON is integer, real, or floating point[1]. The string is double-quoted Unicode with backslash escaping. The boolean contains true or false value. The array is an ordered sequence of values, comma-separated and enclosed in square brackets. The object is collection of key:value pairs, comma-separated and enclosed in curly brackets. Null contains empty value.

Main advantage of JSON is that it translates directly into universal data structures.

2. RDF/JSON serialization

RDF as an abstract model has several serialization formats. XML is one of the formats for storing and transmitting data. A quick read through of the W3C RDF web pages [4] leaves no room for doubt that the preferred RDF syntax is RDF/XML, but RDF is not strictly an XML format.

The verbosity of XML, complexity of N3, non-readable for humans of Turtle and N-Triples causes that there is no simple syntax model that can be quickly edited in a concise manner.

1 The octal and hexadecimal formats are not used.

Considering the above difficulties it is proposed to introduce a new, universal syntax representing RDF graphs in JSON. Such syntax, being the equivalent to RDF model, would be more legible and brief. The proposed RDF/JSON is a lightweight textual syntax that can be easy modified by humans, servers and clients. The advantage of this syntax is that it can easily convert other syntaxes, eg RDF/XML using XSLT [5] or XQuery [18, 19] to its own format. Another benefit of serializing RDF graphs in JSON is that there are many software libraries and build-in functions which support the syntax [8]. One more advantage of RDF/JSON is that this syntax does not have XML and N-triples restrictions. Another benefit of the serialization is that it may be easily manageable by ECMAScript [2].

This paper presents the option of either requesting RDF as an output, or responding RDF as an input from JSON type services in order to make the data accessible in scripting language environments without the overhead of other syntax parsers.

Listing 1 presents an example of JSON syntax.

```
{
  "triples" :
        [
          {
            "subject" : { "type" : "uri" , "value" : "http://example.org/" } ,
            "predicate" : { "type" : "uri" , "value" : "http://purl.org/dc/elements/1.1/creator" } ,
            "object" : { "type" : "literal" , "value" : "John Smith" }
          }
       ]
}
```

Listing 1. Simple RDF/JSON document

2.1. Flat triples approach

Clients and servers need to process data into triples, and denote those triples in some type of data structure. In addiction this data structure should be easy to read and write to humans and machines. This is the area I see for serialization RDF/JSON through the flat triples approach. The proposal shows the RDF triples in a simple understandable, and easy to serialization manner. It describes JSON structure for RDF graph that expresses the whole RDF model that does not lose information.

This method uses only JSON root object named 'triples', and arrays of objects symbolizing subject, predicate and object. The objects should contain keys named 'type' and key named 'value'. The members of these objects are unordered. Values of key named 'type' are defined in table 1. Values of key named 'value' depend on types. When type equals literal it optionally can be define with language, than it should contain key named 'xml:lang'. Allowed value of 'xml:lang' is defined in [1] and if supplied, it must not be empty. When the type equals 'typed-literal' it should contain key named 'datatype'. Value of 'datatype' should be declared as URI. When the type is 'uri', value should be full URI, not just short QName.

types	*subject*	*predicate*	*object*
uri	yes	yes	yes
bnode	yes	no	yes
literal	no[2]	no	yes

typed-literal	no[3]	no	yes

Table 1. Types of RDF terms

Algorithm 1 presents an idea of generate flat triples.

```
Input: triples
Create a JSON root object named triples
Create a JSON array in root object
Foreach triples do
  Create a JSON object named 'subject' in array
  Get the subject from a triple
  Add a key/value pair to object named 'subject' with key being the string named 'type' and value being one of the types from table 1
  Add a key/value pair to object named 'subject' with key being the string named 'value' and value being lexical value of the subject
  Create a JSON object named 'predicate' in array
  Get the predicate from a triple
  Add a key/value pair to object named 'predicate' with key being the string named 'type' and value being 'uri'
  Add a key/value pair to object named 'predicate' with key being the string named 'value' and value being lexical value of the predicate
  Create a JSON object named 'object' in array
  Get the object from a triple
  Add a key/value pair to object named 'object' with key being the string named 'type' and value being one of types from table 1
  Add a key/value pair to object named 'object' with key being the string named 'value' and value being lexical value of the object
  If the triple value is plain text and has language then
    Add a key/value pair to object named 'object' with key being the string named 'xml:lang' and value being the language token
  If end
  If the type triple is a typed literal then
    Add a key/value pair to object named 'object' with key being the string named 'datatype' and value being the the datatype URI
  If end
Foreach end
```

Algorithm 1. Generating RDF/JSON

2.2. A relational model of RDF terms in RDF/JSON

The flat triples approach used in RDF/JSON propose could be a substitute of the relational model.

The relational model for database management is a database model based on first-order predicate

2 A future Working Gropu with a less restrictive charter may extend the syntaxes to allow literals as the subjects of statements.

3 A future Working Gropu with a less restrictive charter may extend the syntaxes to allow typed literals as the subjects of statements.

logic, first formulated and proposed in 1969 by Edgar F. Codd [21, 22].
In set theory, an n-tuple is a sequence of n elements, where n is a positive integer. An RDF tuple is a partial function from variables to RDF terms.

An RDF tuple is totally different from an RDF triple. A triple always has three parts with not changing names: subject, predicate and object, while an RDF tuple can have any number of components with names not based on any principle. A triple is a statement – it implies a semantic relationship between its parts. A tuple does not carry meaning, it is just a container than maps some variables to some RDF terms [11].

However, any triple could be represented as a tuple. There is a one-to-one mapping from RDF triples to tuple with subject, predicate and object.

Sample of mapped triple to tuple present in fig. 1.

$$t = \begin{cases} ?subject \rightarrow \text{http://example.org/} \\ ?predicate \rightarrow \text{http://purl.org/dc/elements/1.1/creator} \\ ?object \rightarrow \text{John Smith} \end{cases}$$

Fig. 1. Tuple with subject, predicate and object

The triples are 3-tuples and the well-known relational model of data which is explicitly designed to represent tuples and the collections of them.

2.3. Depending on the SPARQL

RDF tuples are just term of SPARQL solutions. The variables in a tuple are named its attributes.

An RDF relation is a set of RDF tuples. It can be described as a table or just another flat format. Each row is an RDF tuple and each column is an attribute, named by a variable.

A graph relation is any relation whose heading is {?subject, ?predicate, ?object}. Each of the attributes must be bound in every tuple. Every tuple of subject, predicate and object is similar to an RDF triple. Every graph relation has an equivalent RDF graph and inversely.

The term tuple is used universally in relational algebra. To obtain RDF/JSON serialization projection operator has been used.

In relational algebra, a projection is a unary operation written as $\pi_{a1,a2, \ldots, an}(R)$ where $a_1, a_2, \ldots, a_n$ is a set of attribute names. The result of such projection is defined as the set obtained when the components of the tuple R are restricted to the set $\{a_1, a_2, \ldots, a_n\}$. The projection operator restricts the relation to subset of its attributes [12].

Listing 2 presents an idea of flat triples generated using projection operation that could be compatible with RDF/JSON syntax. The evaluation of triple pattern is the set of mappings that make triple pattern to match the graph and have as domain the variables in triple pattern. All mappings are compatible if they agree in their shared variables.

```
SELECT ?subject ?predicate ?object WHERE { ?subject ?predicate ?object }
```

Listing 2. SPARQL query that generated flat triples

3. Comparison with other JSON serialization approaches

There is several other existing approaches to the problem of representing RDF data in JSON.

One of the approach is Talis resource-centric serialization of RDF in JSON [14]. This proposal represents a set of RDF triples as a series of nested data structures. Each unique subject in the set of triples is represented as a key in JSON object. The value of each key is a object whose keys are the URIs of the properties associated with each subject. The value of each property key is an array of objects representing the value of each property. Main advantage of this approach is that it similar to RDF/XML. Disadvantages are that it is more difficult to process because it is less regular then flat approach and this incompatible with SPARQL Query Results in JSON [13].

Another approach is RDF in canonical JSON objects – RDFj [15]. This proposal is is a very close relative of RDFa [16]. It allows JSON objects to be part of the graph, which is particularly useful when functions are part of the JSON object. This proposal has several complicated expressions, eg '$' to set the subject, 'a' to indicate a type, 'context' to indicate context information and 'graph' to indicate a graph. Main advantage of this approach is that it can be transform into something that could be used more broadly in JavaScript programming [2]. Main disadvantage is that it is very complicated so this serialization is difficult to write and read. It is also incompatible with SPARQL Query Results in JSON [13]. Additionally it is difficult to generate the serialization from SPARQL query.

4. Conclusions

The problem of how to serialize RDF has produced many proposals. Most of them are hard to read and write by humans and machines, hence making the problem seem difficult. I have produced a thought-out and simple proposal. I suggest that it is time that the Semantic Web community support a simple serialization such as mine. I believe my JSON syntax is an interesting approach.

RDF/JSON is a textual syntax for RDF that allows RDF graphs to be completely written in a compact form and easily processed. I propose algorithm that generate these flat triples. This algorithm does not answer how to treat the exceptions like literals as subjects or blank nodes as predicates. Further improvements are in process.

Another way to handle this syntax by mapping the RDF triples by a tuple using the relational model. Next I plan to relate my RDF/JSON to SPARQL Query Results in JSON Working Draft [13] and extend of the RDF containers and RDF collections. My syntax is still under development.

Acknowledgements

The author would like to thank Ivan Herman from the World Wide Web Consortium. Professor Henryk Rybinski's comments and support were invaluable.

References

[1] H. Alvestrand, Tags for the Identification of Languages. Internet Engineering Task Force, 2001.

[2] M. Cowlishaw, ECMAScript language specification. International Organization for Standardization, 1998.

[3] D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON). Internet Engineering Task Force, 2006.

[4] D. Beckett, RDF/XML Syntax Specification (Revised). World Wide Web Consortium, 2004.

[5] M. Kay, XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, 2007.

[6] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie and J. Siméon, XQuery 1.0: An XML Query Language. World Wide Web Consortium, 2007.

[7] J. Robie, The syntactic Web: Syntax and semantics on the Web. In Extreme Markup Languages'2001, 2001.

[8] P. Lakshman and A. Wirfs-Brock, ECMAScript language specification 5th Edition. Ecma International, 2009.

[9] E. F. Codd, Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks. IBM Research Report, 1969.

[10] E. F. Codd, A Relational Model of Data for Large Shared Data Banks, in Communications of the ACM, 1970.

[11] R. Cyganiak, A relational algebra for SPARQL, HP Labs, 2005.

[12] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, Addison-Wesley, 2010.

[13] K. G. Clark, L. Feigenbaum and E. Torres, Serializing SPARQL Query Results in JSON. World Wide Web Consortium, 2007.

[14] K. Alexander, RDF JSON Specification. Talis, 2008

[15] M. Birbeck, Description of the RDFj syntax. Backplane, 2009.

[16] B. Adida, M. Birbeck, S. McCarron, S. Pemberton, RDFa in XHTML: Syntax and Processing. World Wide Web Consortium, 2008.