DE GRUYTER OPEN

Applied Computer Systems doi: 10.1515/acss-2015-0017

2015/18

Analysis of RDF Syntaxes for Semantic Web Development

Yevgeny Gryaznov[1], Pavel Rusakov[2]
[1,2] *Riga Technical University, Latvia*

***Abstract* – In this paper authors perform a research on possibilities of RDF (Resource Description Framework) syntaxes usage for information representation in Semantic Web. It is described why pure XML cannot be effectively used for this purpose, and how RDF framework solves this problem. Information is being represented in a form of a directed graph. RDF is only an abstract formal model for information representation and side tools are required in order to write down that information. Such tools are RDF syntaxes – concrete text or binary formats, which prescribe rules for RDF data serialization. Text-based RDF syntaxes can be developed on the existing format basis (XML, JSON) or can be an RDF-specific – designed from scratch to serve the only purpose – to serialize RDF graphs. Authors briefly describe some of the RDF syntaxes (both XML and non-XML) and compare them in order to identify strengths and weaknesses of each version. Serialization and deserialization speed tests using Jena library are made. The results from both analytical and experimental parts of this research are used to develop the recommendations for RDF syntaxes usage and to design a RDF/XML syntax subset, which is intended to simplify the development and raise compatibility of information serialized with this RDF syntax.**

***Keywords* – Semantic Web, RDF, graph, syntax, XML.**

I. Introduction

World Wide Web development can be divided into three main periods: Web 1.0, Web 2.0 and Web 3.0 [1], [2]. The first "version" of Web is mainly static from the user`s point of view. It is almost not possible to affect information on a Web page, because only webmaster was able to change information on a page by modifying contents directly. Web 2.0 opened new possibilities for Web interaction: dynamic and interactive Web pages. In this generation of Web users can add, edit or even delete information on a given Web page in a convenient manner using simple control elements like menus, buttons, text editor areas etc. [1], [3]. The design of a Web page itself is user-friendly and intuitive.

Semantic technologies give new perspectives for World Wide Web development. Semantic Web – Web 3.0 – is a product of injection of semantic technologies in existing Web. It is not an entirely new Web, rebuilt from scratch, but an integration of a well-known and developed "mechanical" Web 2.0 technologies and semantic tools. Such tools are intended to improve the following aspects of Web usage [4]:

1. Data integration. Data from different sources (from any sources in a perfect case) can be integrated in a whole in order to get a largest possible content and fuller knowledge base, so knowledge search and/or processing can give a better result.
2. Resource discovery and classification. Improvement of effectiveness of search engine can be achieved by applying algorithms based on knowledge processing techniques from the one side, and appropriate knowledge representation and storage tools from the other side.
3. Knowledge automatic (intelligent) processing and exchange [5], [6]. Application of knowledge-oriented data processing and storage tools is important for deployment of intelligent agent system over the Web. Such agents can substitute user`s activity in specific Web routines like information collection on a specific topic, e-shopping and such time-consuming tasks like trip planning.

The goal of this research is to analyze and compare existing RDF syntaxes based on XML to specific RDF syntaxes, and to propose a solution for better RDF/XML syntax usability in context of Semantic Web.

This paper is organized as follows. The second section describes RDF framework with an example and explains the need in RDF syntaxes. In the third section RDF syntaxes are being analyzed and compared by multiple factors. The fourth section introduces an experiment to compare syntaxes by parsing/serialization performance parameter. In the fifth section authors describe their proposed RDF/XML syntax improvement and give recommendations on other syntax usage. The final sixth section is a conclusion on this work, a comparison table and research highlights are given.

II. RDF Framework and Syntaxes

RDF framework is a simple but powerful tool for formalizing data (e.g. metadata, knowledge-like data). It is only an abstract model, which describes how to break data into small units and tie them together to make a machine-processable copy of data. RDF implementations (serializations, syntaxes) will be discussed later in this section. RDF defines a minimal unit of interconnected information: a triple (Fig. 1). A triple is an oriented two-node graph, where two nodes represent *subject* and *object*, and the oriented edge is a *predicate* – a relationship between the *subject* and the *object*. Note that the edge is always oriented from *subject* to *object*, so it is always possible to infer the role of a node.

Predicate (also called a *property*) can be used for indicating any type of relationship between two nodes. Looking from a global perspective, RDF doesn`t predefine any relationship types, it is being done by a developer of a system, which uses RDF. A *triple* also called a *sentence* or a *fact*, because it states something that is always true. The reason why RDF is

necessary and why just XML is not used, is the fact that information and knowledge is dedicated to many different domains. It means that in case of XML developers must design specific XML dictionaries for each domain. RDF allows using a unified way, independent form for a particular domain. But XML can be used as a format for writing down RDF graphs [7], [8]. [9] contains detailed information on XML. RDFS and OWL are another level of abstraction, but not syntaxes, so they are not discussed in this paper.

Fig. 1. RDF minimal information unit: a triple.

An example of RDF usage is given for better understanding. Let us have a simple relational database for scientific papers. The information in the database should be available over the Web, but imagine that the Semantic Web is already working. It is necessary to send and receive data in RDF format (it does not matter in which RDF syntax, because it is still RDF in any case) and map them into relational form. Additional information on mapping can be found in [10]. This scenario is used because it is easier to understand what information is shown in RDF form after looking at that same information written down in a convenient relational table form.

There are four tables in the database: PAPERS (Table I), AUTHORS (Table II), REVIEWS (Table III) and REVIEWERS (Table IV).

TABLE I

PAPERS TABLE IN DATABASE

ID	Name	Published	File
001	Water data storage	2011-12-22	001.PDF
002	Cipher attacks	2012-05-14	002.PDF

TABLE II

AUTHORS TABLE IN DATABASE

ID	Name	Surname	Faculty
001	Evalds	Morozovs	DITF
002	Anita	Bekereja	DITF

TABLE III

REVIEWS TABLE IN DATABASE

ID	Paper	Mark	Text
001	001	7	…
002	001	9	…

TABLE IV

REVIEWERS TABLE IN DATABASE

ID	Name	Surname	Faculty
001	Erika	Eglite	DITF
002	Edmun	Kurpni	DITF

Let us assume that the database also has additional tables to represent many-to-many relationships between AUTHORS and PAPERS tables, between REVIEWS and REVIEWERS tables. The resulting RDF graph is marked as "1" in Fig. 2. Note that for space saving purpose information is shown only about the first paper (ID=001) with one author, and information about the second review is not shown, and other shortenings are marked with "…". The information displayed with a graph in Fig. 2. is about the paper with ID=001, its author, reviews and corresponding reviewers.

As you can see, the information on a particular topic is displayed with an oriented graph. It may look frustrating for a human, but it is easy to process with a computer program (using inferencing techniques) making possible automatic data processing. Being a processable data, it can be converted into a human-readable form by using name substitution or displaying it with tables. Any RDF graph consists of triples, so the given in the example one consists of them too. Each two adjacent nodes connected with an oriented edge make a triple. For example, a sentence or *the fact* "author`s name is Evalds" has a corresponding triple in the given graph: "Authors/ID=002" – "Authors#Name" – "Evalds". Complex sentences can be displayed using multiple triples.

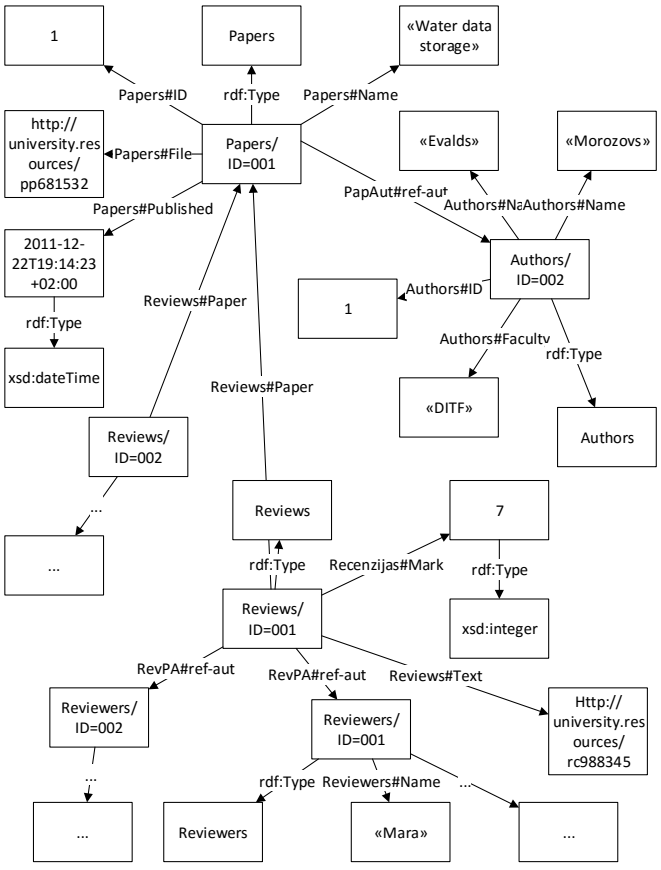


Fig. 2. RDF graph that shows information about particular scientific paper, its author, reviews and reviewers.

As far as RDF is an abstract model, it is possible to use graphs to display RDF semantic data in a more human-readable form than specific syntax, but the problem is to store and process a graph with a computer. It is a purpose RDF syntaxes are designed for. RDF syntax is a specific data format to serialize, store and exchange RDF graphs. There are

many syntaxes of different types, so classification is needed to understand types and corresponding syntaxes. First of all, syntaxes are binary or text-based. The first ones are both storage-effective and processing-effective. But they cannot be read by a human without a special tool. The examples of binary syntaxes are HDT [11] and Thrift [12]. Text-based syntaxes are both machine-processable and human-readable (using only a plain text editor). Text-based syntaxes are divided into XML-based, JSON-based and specific syntaxes. While XML and JSON syntaxes are dictionaries of these two formats, specific syntaxes are designed to be only RDF graph serializations. They are free of XML or JSON-specific nuances and are simpler to understand. This is why specific text-based syntaxes are chosen as an etalon (or standard).

This paper is dedicated to XML syntaxes, so binary and JSON syntaxes are not being discussed, but XML and specific syntaxes are being analyzed, to compare them and make a conclusion on which syntaxes should be used for a particular purpose. Syntaxes which are being discussed in this paper are:

1. XML-based: RDF/XML [13], TriX [14], RXR [15], Grit [16], Treetriples [17].
2. Specific text-based: Turtle/TriG [18], N-Triples [19] and N-Quads [20].

Each syntax`s specific nuances are being discussed in the next section, making a comparative analysis.

III. Comparative Analysis of RDF Syntaxes

In order to find out how appropriate syntaxes are for RDF graph serialization purposes, a comparative analysis is needed. In this section authors explain which factors are chosen and explain how each syntax conforms to that factor. The analysis is organized as follows: authors explain a particular factor first, and then analyze each syntax according to that factor.

The factors are:

1. Understandability to a human.
2. Syntax shortenings.
3. Graph identifiers and multiple graph serialization in a single file.
4. Support for containers and collections.
5. Reification and blank nodes.
6. Suitability for XML tools and technologies.

Note that in the examples full URIs of resources are not used, just shortened simple names.

A. Understandability to a Human

This factor shows how fast and easy RDF information (written using a particular syntax) can be read. It is important because text-based syntaxes are designed to be human-readable. Despite this factor being subjective, other authors use it to compare syntaxes [21], [14] and conclude that RDF/XML is harder to understand than TriX or Turtle. It cannot be measured, but it is possible to compare similar syntactic constructions from different syntaxes. To make comparison as objective as possible, an etalon for a single triple is defined: <subject> <predicate> <object>. This choice is based on information from a book [10], report [21], research [14], RDF specification documents [22], [23], and latest tendencies (Turtle syntax). Concluding that information, syntax should be as close to RDF abstract syntax as possible, so it doesn`t obscure a corresponding graph.

RDF/XML has a structure different from the etalon, because it defines a tree-like structure, while the etalon prescribes the triple`s elements on a single level. In RDF/XML an object is a child element of a predicate element, and the predicate element is a child of a subject element. In case of a typed literal as object value, different syntactic construction is being used: object literal is written without any tags, but predicate`s element has an attribute indicating literal`s type (XML types can be used as well). The example is shown in Fig. 3. It describes the following information: "A [scientific] paper has an author who is 27". Note that in the given example author object is also a subject from the point of view of the second fact ("author`s age is 27").

```
<rdf:Description rdf:about="Paper">
    <author>
        <rdf:Description>
            <age rdf:type="Age">
                27
            </age>
        </rdf:Description>
    </author>
<rdf:Description>
```

Fig. 3. RDF/XML triples syntax (fragment).

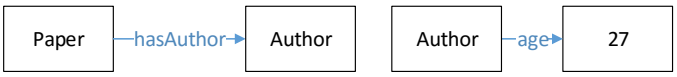


Fig. 4. RDF graph.

RXR syntax differs from the RDF/XML syntax, and is more conforming to the etalon: each element of a triple is placed on the same level under the <*triple*> XML tag. Each element has its own named XML tag. According to RXR documentation, triples cannot be included into each other, like in RDF/XML. A developer must explicitly use a resource`s URI in each triple. For example, a graph from Fig. 4 can be written in only the way shown in Fig. 5.

Trix syntax is similar to RXR, but only one tag <*uri*> is used for all elements in a triple. So that the role of an element is defined only by its position, unlike RXR, where both position and tag are used for this purpose.

Grit syntax derived hierarchical triple structure from RDF/XML. It means that object resource is being placed as an attribute value inside predicate tag, and the predicate tag is a child element of subject, see Fig. 6.

The last XML syntax called Treetriples has triples structure identical to Grit (or RDF/XML in case of shortenings used), but names of subject, predicate and object are <*s*>, <*p*> and <*o*>.

Non-XML syntaxes better suit the readability factor, because they almost perfectly correspond to the etalon. **Notation-3** and **Turtle** are similar syntaxes, because Turtle is a simplified version of the first. In Fig. 7 you can see that its triple syntax is like RDF abstract syntax, the only difference is the enclosing. Other non-XML syntaxes are identical to the

syntax mentioned above, except **TriG** which is an extended version of **Turtle** and **N-Triples/N-Quads** which are simplified versions. TriG has the same syntactic constructions Turtle has, but also allows serializing multiple named RDF graphs into one file. N-Triples is minimalistic syntax: each triple must be on a separate line and enclosed with a dot, no shortenings are allowed. N-Quads is an extended version of N-Triples, but instead of triples it uses quads: 4-tuples, where additional element can be used to specify the graph`s name.

```
<triple>
    <subject>Papers#ID=001</subject>
    <predicate>Papers#Author</predicate>
    <object>Authors#ID=001</object>
</triple>

<triple>
    <subject> Authors#ID=001</subject>
    <predicate> Authors#Age</predicate>
    <object datatype="xsd:integer">27</object>
</triple>
```

Fig. 5. RXR triples syntax (fragment).

```
<resource uri="Papers#ID=001">
    <Author ref="Authors#ID=001">
</resource>

<resource uri="Authors#ID=001">
    <Age fmt="datatype">
        <xsd:integer>27</xsd:integer>
    </Age>
</resource>
```

Fig. 6. Grit triples syntax (fragment).

```
<Papers#ID=001> <Author> <Authors#ID=001> .
<Authors#ID=001> <Age> "27^^xsd:integer" .
```

Fig. 7. Turtle triples syntax (fragment).

B. Syntax Abbreviations

This factor is not only a part of human readability, but also a part of machine-processability. First of all, shortened syntax takes less space on a storage drive or of an exchange channel. The second aspect is that more complex syntactic constructions may lead to higher computer resource usage. The first shortening is **omitting common subject or predicate resource.** Notation-3, Turtle and Trig allow grouping triples with common subject and/or predicate by omitting the subject and/or the predicate.

RDF/XML allows grouping only by subject. RXR, TriX, Treetriples, N-Triples and N-Quads do not have any shortenings of this kind.

The next shortening is specific to RDF/XML syntax: **empty property (predicate) element.** It means that a subject resource can be written as property tag attribute value. In this case the predicate element does not have an enclosing tag.

Property attribute is another RDF/XML specific shortening. It allows placing property as attribute name in the subject`s <*Description*> element, and object`s resource as attribute`s value. It is possible only if the object is string literal.

Prefixed Names allows giving alias for a common URI string. For example: *authors* is the same as http://university.papers/DB/Authors, so there is no need to write full URI like http://university.papers/DB/Authors#ID=001, but short form with a prefix can be written: *authors:ID=001*. The XML standard defines this kind of prefixed names as *QNames* [24]. Therefore, it has a restriction on element and attribute names: *QNames* cannot be used for names. It means that XSD or RelaxNG schema cannot be used to describe this document – it is not a valid XML document. This also makes problems with XSLT or XPath usage. The same problem is true for Grit syntax, because it prescribes using *QNames* as element names. The remaining XML syntaxes (RXR, TriX and Treetriples) are free of this problem, because they do not allow usage of *QNames* for element and attribute names. *QNames* has one more significant restriction: hash symbol (“#”) is not allowed. But URI comes in two forms: slash URI and hash URI. It lowers URI flexibility, because the only way to use hash URI with *QNames* is to append the hash symbol to the prefix part of URI.

Omitting blank nodes is available in RDF/XML, Turtle and TriG syntaxes. This abbreviation allows omitting blank node identifier and putting blank node`s predicate and object into a predicate element, which points to that blank node. Turtle and TriG have this abbreviation too, but it is significantly shorter.

C. Multiple Graph Serialization

TriX is the only XML syntax, which allows naming and serializing more than one RDF graph in a single file. TriG and N-Quads are non-XML syntaxes, which support named graphs. In the case of N-Quads it is made by adding one more element – graph identifier – to a triple, making it a quad.

D. Container and Collection Support

Containers (*rdfs:Container*) and Collections (*rdf:Collection*) are resources defined by the RDF schema. The purpose of both types is the same: to encapsulate multiple resources. Therefore, containers allow adding new resources, but collections do not. RDF/XML and Treetriples support both containers and collections. Container support in Turtle is mentioned in only one W3C document [25], but in all specification documents it is not mentioned [18]. Containers are part of latest RDF specification version [22], so a syntax, which corresponds to RDF specification must support containers. Therefore, N-Triples and N-Quads are designed as minimalistic as possible, so they do not support any containers, or collections.

E. Blank Nodes

It is a tool, which can be used to build facts about abstract resources. For example, there are two ways how graph in Fig. 4 can be viewed. First, “Author” is a concrete resource with unique URI, in the second case it is an unknown resource, but still unique. In the second case author node is a

blank node, because it is not mentioned which author it is. That blank node is used in the second triple, when name is given to the author. So it means: "a paper has an author, the author`s age is 27". It is not possible to infer who is that author, but his age is known. To allow multiple triples with the same blank node, blank node identifiers are used. In RDF/XML three different ways to incorporate blank nodes are available. The first one (tree-structured) is shown in Fig. 3. The second is abbreviated with property attribute. The last one is using identifiers and multiple triples. In Turtle blank nodes can be written in two ways: using omitted syntax and using identifiers and multiple triples.

F. Reification

Reification allows to identify a triple in order to create another facts (triples) about it [26]. It can be useful in RDF data storages, when additional information about statements is needed. Such information is: who is the creator of the triple, when the triple was created, etc. Normally, the reified triple is written using five facts: statement identifier definition, definition of a subject, an object, a predicate, and type definition (type *"rdf:Statement"*). RDF/XML also has abbreviated syntax, which allows using only one statement by applying an identifier to it.

G. Compatibility with XML Tools and Technologies

Since XML syntaxes are XML-based to use advantages of XML schema description languages (XSD, RelaxNG) and other technologies (XSD, XSLT, XQuery, XPath), this factor is important. RDF/XML cannot be fully defined by XML schema, because it has some syntactic constructions which cannot be described using schema languages. The reason for that is that the predicate element must be able to have any name, because in RDF predicate can have any name the user wants. For this purpose XML schema language allows using <*any*> element, which indicates that any element can be placed inside the given element. But this also means that the underlying structure (contents of the parent element) cannot be controlled at all. Anything is possible here. So from this point of view RDF/XML is useless as XML syntax, because it is not fully XML-schema compatible. Since Grit has similar syntax, it is not fully supported by XML schema, too. But other XML syntaxes use unambiguous XML element names (resource URIs are always placed inside elements as contents), so they can be fully defined by XML schema [14]. It also means that these syntaxes have better suitability to XML technologies like XSLT, XPath and XQuery, so templates and queries are simpler in this case.

IV. Experimental Speed Test

In order to find out how high or low a performance of each syntax is, an experiment was planned and done. The point of the experiment was to measure serialization and parsing (deserialization) time for the same portion of information in different syntaxes. The shorter the time is – the higher is the performance. To measure the time, a program was written in Java, using Apache Jena RDF library (version 2.13.0) [27]. This library was used because it has more syntaxes than other libraries: JRDF version 0.5.6.3 [28], Sesame version 2.8.3 [29]. The following syntaxes were used in this experiment: RDF/XML (four variants), Turtle (four variants), TriX and N-Triples. TriG (Turtle extension with named graphs support) and Notation-3 (with support for logical expressions) were not tested, because their syntaxes are identical. The chosen RDF datasets were written in N-Quads, which is an extension of N-Triples, but since the RDF data does not contain any named graphs, this syntax was not tested too.

The data used in the experiment was [30]:

- Dataset A is a fragment of DBpedia database dump in English, N-Triples syntax, 320 485 triples, file size: 37.89 MB, file name: 2013-12-22-join-*summary-dbpdia-live.nt* ;
- Dataset B is a fragment of DBpedia picture database dump (see [30] Images Dataset, data is in English), N-Triples syntax, 320 485 triples, file size: 71.33 MB, file name: *images_en.nt*;
- Dataset C is a fragment of DBpedia person database dump (see [30] Persondata Dataset, data is in English), N-Triples syntax, 320 485 triples, file size: 37.27 MB, file name: *persondata_en.nt*.

The system for the experiment had the following configuration: CPU: Intel Core i7-4770K, RAM: 8GB DDR3-1333MHz, OS: MS Windows 7 SP1 (64 bit), JRE: Java 1.7.0.45 (64 bit). Apache library does not allow separating serialization and parsing from disk input/output operations. So first, authors discovered how different type of a storage device affects serialization/parsing operations to minimize the impact of a hardware. Three types of storage devices were tested: HDD, SSD and RAM drive. The results were that the difference was small enough to be ignored (numbers showed parsing time in milliseconds). HDD: 2351.49±116.40; SSD: 2323.81±108.13; RAM drive: 2296.395±123.96. This meant that none of these drive types was a bottleneck in serialization/parsing operations. Therefore the RAM drive was chosen as the fastest solution. Next, an optimal measurement count was defined, because if a count is too small it can lower precision, if it is too high – raise the measurement time. This test was made to RDF/XML syntax and relative error (it is the absolute error divided by the magnitude of the exact value; the percent error is the relative error expressed in terms of per 100) was: 4.57% for 200 measurements, 5.28% for 150 measurements and 5.91% for 100 measurements. The variant of 150 measurements was chosen as a compromise. During further experiment, it was necessary to raise the number of measurements to 200 and 300 for TriX and N-Triples parsing, because relative error for 150 measurements was over 10 %. This solution did not help, so 150 measurements were chosen as well.

Authors measured both serialization and parsing times for every syntax and every variant of the four available (for RDF/XML and Turtle). The measurement results for each dataset were calculated and the final results are given in three tables for each dataset: Table V for dataset A, Table VI for dataset B and Table VII for dataset C. In these tables you can

see the overall time: it is the time needed to make a full cycle of parsing and then serializing (or vice versa). Since RDF/XML and Turtle syntaxes have four variants each, their full names are shown in the tables. There are charts available for parsing time (Fig. 8), serialization time (Fig. 9), total time (Fig. 10) and file size (Fig. 11). The scale has been cut for Fig. 9 and Fig. 10 for better understanding of the results, because RDF/XML results for dataset A are significantly larger than the results of other syntaxes.

TABLE V

EXPERIMENT RESULTS FOR DATASET A

Syntax	**Parsing time, ms**	**Serialization time, ms**	**Total time, ms**	**File size, MB**
RDF/XML	2347.55	19936.95	22284.50	30.56
RDF/XML Abbr.	2363.73	17704.80	20068.53	30.56
RDF/XML Plain	2423.38	1543.95	3967.33	23.49
RDF/XML Pretty	2320.57	16033.90	18354.47	30.56
Turtle	1262.27	223.53	1485.80	13.42
Turtle Blocks	1433.25	282.37	1715.62	28.30
Turtle Flat	1657.35	345.47	2002.83	37.89
Turtle Pretty	1258.85	282.37	1541.22	13.42
TriX	1159.63	1720.50	2880.13	59.28
N-Triples	999.24	85.95	1085.19	37.89

TABLE VI

EXPERIMENT RESULTS FOR DATASET B

Syntax	**Parsing time, ms**	**Serialization time, ms**	**Total time, ms**	**File size, MB**
RDF/XML	3555.66	5837.25	9392.91	55.21
RDF/XML Abbr.	3531.82	5797.25	9329.07	55.21
RDF/XML Plain	4327.66	3957.85	8285.51	65.92
RDF/XML Pretty	3574.40	5773.40	9347.80	55.21
Turtle	3841.92	965.25	4807.17	66.20
Turtle Blocks	3793.09	550.25	4343.34	61.75
Turtle Flat	3923.01	649.20	4572.21	67.10
Turtle Pretty	3854.83	967.80	4822.63	66.20
TriX	1371.85	2353.05	3724.90	87.83
N-Triples	1419.58	175.65	1595.23	67.10

TABLE VII

EXPERIMENT RESULTS FOR DATASET C

Syntax	**Parsing time, ms**	**Serialization time, ms**	**Total time, ms**	**File size, MB**
RDF/XML	2357.46	2367.55	4725.01	23.22
RDF/XML Abbr.	2333.12	2368.95	4702.07	23.22
RDF/XML Plain	2501.35	1258.80	3760.15	26.08
RDF/XML Pretty	2346.59	2357.30	4703.89	23.22
Turtle	1231.76	501.65	1733.41	27.86
Syntax	**Parsing time, ms**	**Serialization time, ms**	**Total time, ms**	**File size, MB**
Turtle Blocks	1233.59	308.70	1542.29	27.57
Turtle Flat	1430.98	367.60	1798.58	37.26
Turtle Pretty	1283.20	554.15	1837.35	27.86
TriX	1339.75	1327.65	2667.40	64.15
N-Triples	1162.49	134.30	1296.79	37.26

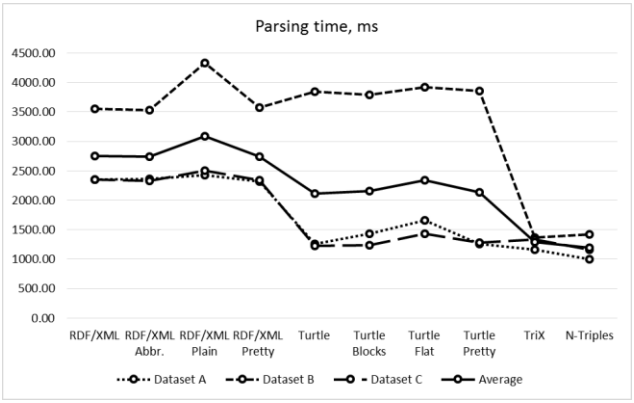


Fig. 8. Parsing time results for all syntaxes.

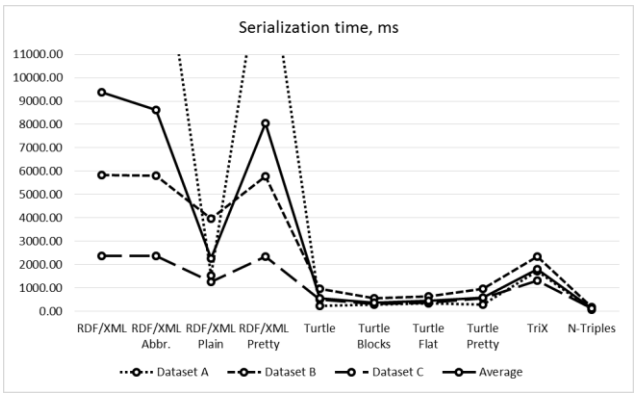


Fig. 9. Serialization time results for all syntaxes.

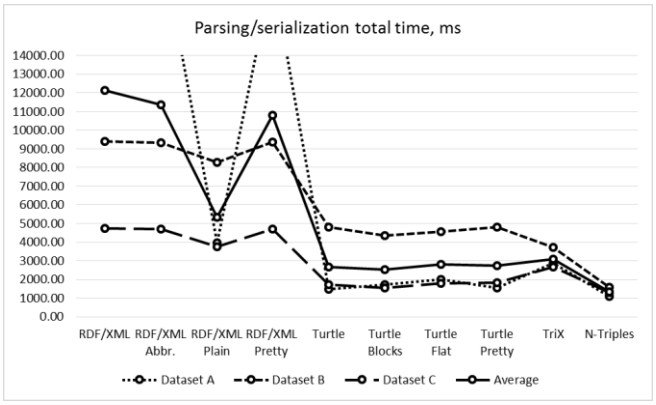


Fig. 10. Parsing and serialization total time for all syntaxes.

Since N-Triples and TriX syntax results have high relative error values, their performance is rather expected than guaranteed. Syntax performance rating is different for each dataset, which means that the performance depends on the data type and structure (triple count was the same for all datasets: 320 485), though these results are not enough to predict a tendency. The difference in file sizes for each dataset tells that this parameter is unpredictable and is dependent on data type and structure too.

Fig. 11. File size for all syntaxes.

According to the results, RDF/XML syntax is the slowest by average total time (see Fig. 10). It is true for parsing and serialization time (see Fig. 8 and Fig. 9), but parsing performance is comparable to TriX. RDF/XML Plain variant is only slightly slower than other RDF/XML variants. Though the difference is larger in the case of dataset B, where the time of RDF/XML Plain is 4327.66 ms, but the time of other RDF/XML variants is 3531.28–3574.40 ms. So RDF/XML Plain is 21.07–22.53 % slower than other RDF/XML variants. For other datasets this parameter has no significant difference. Serialization measurements show that RDF/XML Plain is 31.45–32.20 % faster for dataset B and 46.86–47.27 % faster for dataset C, in comparison to other RDF/XML variants. For dataset A difference is significantly larger: 90.37–92.26 %. It means that RDF/XML Plain variant may not be the fastest syntax when parsing performance is needed, but RDF/XML serialization variant will always be the fastest.

Turtle syntax variants do not significantly differ, but dataset B results are lower than for datasets A and C. Speaking about the total time, Turtle is as fast as TriX, slower than N-Triples, but parsing performance is worse than for both TriX and N-Triples. TriX is the fastest XML syntax in both parsing and serialization operations. N-Triples has the best parsing and serialization performance between non-XML syntaxes and all other syntaxes in this experiment. Average total time helped to notice the tendency: the simpler the syntax is, the better is the performance. The results show that XML syntaxes implemented in Jena library are slower than non-XML syntaxes. TriX is the fastest XML syntax, but N-Triples is the fastest non-XML syntax. Important note can be done about RDF/XML: RDF/XML Plain variant with the best serialization results compared to other RDF/XML variants, does not use nested *<Description>* elements, but other (slower) variants do. RDF/XML Plain variant uses property attributes to serialize object resource. It means that the only difference between RDF/XML Plain and the other variants is the usage of multilevel nested elements, and the usage of nested elements slows down serialization. It is approved by dataset B and C results: RDF/XML, RDF/XML Abbreviated and RDF/XML Pretty variants are significantly lower than for dataset A. The dataset analysis showed that datasets B and C do not contain multiple level nested triples, so serialization

was faster. So, nested XML elements lower RDF/XML serialization performance.

V. Recommendations for RDF Syntax Usage

A. Requirements for XML Syntax

The requirements were formulated for XML syntax, based on the research:

1. Can be fully defined by XML schema.
2. Simple triple syntactic form, where all triple elements are on the same level.
3. Unified triple syntax in all situations (abbreviations with nested elements are denied).
4. All XML elements have unambiguous names (URI cannot be used for XML tag names).
5. QNames cannot be used for element and attribute values.
6. Additional syntactic constructions (RDF collections, QNames, etc.) must be implemented outside XML schema, using XSLT or other transformations.

RXR, TriX and Treetriples syntaxes fill these requirements. Though RDF/XML is the only XML syntax recommended by W3C as a standard. Authors propose an improvement for this syntax, to fill requirements formulated, and the improved syntax must be compatible with the existing RDF/XML syntax version. But it is not possible to design full XML schema for this syntax, because of an unpredictable property tag name. Therefore it is not possible to put restrictions on property XML element attributes, despite the fact that they all are known (predictable). Consequently, the proposed improvement is based on simplifying triple syntax and preventing multiformity of abbreviations.

B. Proposed RDF/XML Syntax Improvement

Triple syntax. Only one triple syntactic form for typed/non-typed literal must be allowed. For triples with object URI two variants are possible. The first one (variant A) conforms to the syntax shown in Fig. 3. Its advantage is the logical uniformity of triples with URI and literal objects: in both cases the object value is placed under the property element. But it incorporates nested <*Description*> elements, which slows down serialization performance. The second variant (variant B) uses empty property element syntax and adds object URI as a value to *rdf:resource* attribute, see Fig. 11. It has two advantages: better performance (using no nested <*Description*> elements) and unified XML structure.

The blank node syntax must conform to triples syntax. To simplify the syntax, blank nodes without identifiers must be denied. This leads to the only possible syntax with identified blank nodes. Containers and collections do not complicate triple syntax and are part of RDF specification, so there is no need in removing them. Usage of full URIs instead of *QNames* is recommended. The proposed improvement is compatible with the existing RDF/XML version, it is a subset of it. Therefore, named graphs and identifiers are not recommended. The only way to implement this feature is to explicitly notify about the fact that the given syntax is a modified RDF/XML. File name extension can be used for that purpose: *"*.rdfg"* instead of usual *"*.rdf"*. But this does not solve the problem when using RDF data transfer: another technique is needed in order to inform a parser about modified RDF/XML data. This could be done using modified parser and specific directives. The remaining syntax is not affected by any changes.

To use the proposed recommendations, a table (see Table VIII) can be used. It shows corrections to RDF/XML specification items (to be found in [13]) for both proposed variants.

TABLE VIII

Proposed Variants According to RDF/XML Specification

RDF/XML specification item	**Variant A**	**Variant B**
2.2 Node Elements and Property Elements	<*Description*> element can be nested only once (in one triple).	Denied.
2.4 Empty Property Elements	Denied.	Allowed.
2.5 Property Attributes	Denied.	Denied.
2.10 Identifying Blank Nodes: rdf:nodeID	Allowed in conformance with item 2.2.	Allowed in conformance with item 2.4.
2.11 Omitting Blank Nodes: rdf:parseType="Resource"	Denied.	Denied.
2.12 Omitting Nodes: Property Attributes on an empty Property Element	Denied.	Denied.

C. Overall Syntax Usage Recommendations

Further recommendations are targeted mainly on standard developers and software developers.

XML syntaxes can be used when compatibility with XML tools and technologies (parsers, XSLT, XQuery, XPath, DTD, XSD, RelaxNG) are needed, or is more beneficial than usage of non-XML syntaxes. Otherwise, non-XML syntaxes are recommended as both easy readable to a human and high performance on a computer. TriX, Grit and Treetriples are recommended XML syntaxes, but if RDF/XML must be used, the proposed variant B is recommended. Turtle and its extensions Notation-3 and TriG are general purpose non-XML syntaxes. N-Triles and N-Quads are the simplest syntaxes, are well suited for computer processing, but not as readable to a human as Turtle.

VI. Conclusion

Authors give a mark for each syntax by each factor; marks can be viewed in the Table IX. The table contains marks for the proposed RDF/XML improvement variants A and B. The marks are relative – they display only syntax rating by given factor. The scale is: 1 – weakly implemented feature; 2 – normal implementation; 3 – best implementation. The stroke (—) means that the feature is not available in the given syntax. "N/A" – "not available" tells that information about this feature implementation in the given syntax is not available. The factors marked by a star (*) have inversed scale – the bigger the mark, the worse implementation is. The features, which are improved by the proposed solution, are in bold frame.

To conclude this research, generalized recommendations are given. RDF/XML W3C recommended version is much more multiformal than other XML syntaxes, which can cause readability issues to a human. The tool development for this syntax is a complex task too, because the developer has to cover all possible situations. However, RDF/XML is a W3C recommendation which led to its popularity, therefore a new XML syntax will cause compatibility problems. Since non-XML syntaxes are more suitable for RDF graph serialization, there is no need for another XML syntax, incompatible with existing standard. That is why authors propose their solution: RDF/XML subset, variant B, to prevent syntactic multiformity. The apparent advantage of XML syntaxes is compatibility with existing XML tools and technologies (parsers, editors, XSLT, XQuery and XPath).

Future research on this problem might include the following tasks: (1) implementing and testing the performance of RDF/XML parser designed in conformance with the proposed RDF/XML variant A or B; (2) a research focused on non-XML syntaxes based on specific factors (logic expressions, multiple graphs, usage for data streaming).

TABLE IX

GENERALIZATION TABLE FOR ALL DISCUSSED XML RDF SYNTAXES

Factor	**RDF/XML**	**Variant A**	**Variant B**	**TriX**	**RXR**	**Grit**	**Treetriples**
Conformance to the etalon syntax	1	2	2	3	3	2	3
Graph identifiers	—	—	—	3	—	—	—
Multiple graphs in a single file	—	—	—	3	—	—	—
Collections	3	3	3	3	3	3	3
Containers	3	3	3	N/A	N/A	N/A	3
Abbreviated reification	3	3	3	—	—	—	—
Blank nodes with identifiers	3	3	3	3	3	3	3
Blank nodes without identifiers *	3	—	—	—	N/A	N/A	N/A
URI using *QNames* *	3	—	—	—	—	3	—
Other abbreviations	2	—	—	—	N/A	N/A	—
Syntax multiformity *	3	2	2	—	—	—	—
Compatibility with XML tools and technologies	1	2	2	3	3	2	3
XML literals	3	3	3	3	3	3	3
Performance in Jena library	1	N/A	2	3	N/A	N/A	N/A

REFERENCES

[1] G. Cormode and B. Krishnamurthy „Key differences between Web 1.0 and 2.0," *First Monday*, vol. 13, no. 6, June 2008.

[2] D. Dinucci „Fragmented Future," *Print magazine*, pp. 220–222, Apr. 2009.

[3] T. Berners-Lee and M. Fischetti *Weaving The Web*, San Francisco: Harper San Francisco, 1999.

[4] I. Herman. (2009, Nov.) *W3C Semantic Web Frequently Asked Questions* [Online]. Available: http://www.w3.org/2001/sw/SW-FAQ

[5] D. Booth, H. Haas and others. (2004, Feb.) *Web Service Architecture* [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[6] Kumar S. "Agent-Based Semantic Web Service Selection and Composition," *SpringerBriefs in Electrical and Computer Engineering*, New York: Springer New York, 2012, ch. 3, pp. 15–25. http://dx.doi.org/10.1007/978-1-4614-4663-7_3

[7] S. Decker, F. Harmelen and others, „The Semantic Web - on the respective Roles of XML and RDF," *IEEE Internet Comput.*, pp. 1–19, Sep./Oct 2000.

[8] G. Schreiber and Y. Raimond. (2014, June) *RDF 1.1 Primer* [Online]. Available: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/

[9] T. Bray, J. Paoli and others. (2006, Aug.) *Extensible Markup Language (XML) 1.1* (2nd ed.) [Online]. Available: http://www.w3.org/TR/2006/REC-xml11-20060816/

[10] Yu L. *A Developer`s Guide to the Semantic Web*, New York: Springer New York, 2014.

[11] J. D. Fernández, M. A. Martínez-Prieto and others. (2011, Mar.) *Binary RDF Representation for Publication and Exchange (HDT)* [Online]. Available: http://www.w3.org/Submission/2011/SUBM-HDT-20110330/

[12] A. Seaborne. *RDF Binary encoding using Thrift* [Online]. Available: http://afs.github.io/rdf-thrift/rdf-binary-thrift.html

[13] F. Gandon and G. Schreiber. (2014, Feb.) *RDF 1.1 XML Syntax* [Online]. Available: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

[14] J. J. Carroll and P. Stickler. (2004, May) *HP Labs: TriX: RDF Triples in XML* [Online]. Available: http://www.hpl.hp.com/techreports/2004/HPL-2004-56.html

[15] D. Beckett. (2004) *Modernising Semantic Web Markup* [Online]. Available: https://www.dajobe.org/papers/xmleurope2004/

[16] (2010, Sep.) *Grit: Grokkable RDF Is Transformable* [Online]. Available: https://code.google.com/p/oort/wiki/Grit

[17] *Treetriples syntax specification* [Online]. Available: http://djpowell.net/schemas/treetriples/1/SyntaxSpec.html

[18] D. Beckett, T. Berners-Lee, E. Prudhommeaux and G. Carothers. (2014, Feb.) *RDF 1.1 Turtle* [Online]. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/

[19] G. Carothers and A. Seaborne. (2014, Feb.) *RDF 1.1 N-Triples* [Online]. Available: http://www.w3.org/TR/n-triples/

[20] G. Carothers. (2014, Feb.) *RDF 1.1 N-Quads* [Online]. Available: http://www.w3.org/TR/n-quads/

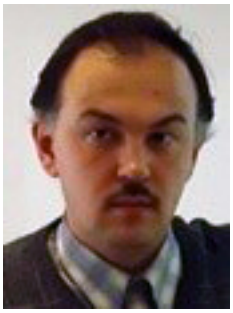
[21] D. Beckett "RDF Syntaxes 2.0" in *W3C 2010 RDF Next Steps workshop*, Stanford, Palo Alto, CA, USA, 2010.

[22] D. Brickley and R. V. Gruha. (2014, Feb.) *RDF Schema 1.1* [Online]. Available: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
[23] P. J. Hayes and P. F. Patel-Schneider. (2014, Feb.) *RDF 1.1 Semantics* [Online]. Available: http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/
[24] T. Bray, D. Hollander, A. Layman and R. Tobin. (2006, Aug.) *Namespaces in XML 1.1 (Second Edition)* [Online]. Available: http://www.w3.org/TR/2006/REC-xml-names11-20060816/
[25] D. Beckett and I. Herman. (2007) *RDF Primer – Turtle Version* [Online]. Available: http://www.w3.org/2007/02/turtle/primer/
[26] S. Powers *Practical RDF*, O`Reilley Media, 2003.
[27] *Apache: Apache Jena Core 2.13.0* [Online]. Available: https://jena.apache.org/documentation/javadoc/jena/index.html
[28] *Java RDF API Framework 0.5.6.3 JavaDoc* [Online]. Available: http://jrdf.sourceforge.net/0.5.6.3/doc/javadoc/index.html
[29] *Sesame 2.8.3 API JavaDoc* [Online]. Available: http://rdf4j.org/sesame/2.8/apidocs/
[30] (2015, Apr.) *DBpedia: Downloads 2015-04* [Online]. Available: http://wiki.dbpedia.org/Downloads

Yevgeny Gryaznov was born in 1991 in Riga, Latvia. He received his Bc. sc. ing. and Mg. sc. ing. from Riga Technical University (RTU) in 2013 and 2015 respectively. He received the Diploma with distinction: Master of engineering science in computer systems.

His fields of interest are computer science, Web technologies, design and development of information systems, data transmission, mobile technologies.

E-mail: jevgenij.grjaznov@gmail.com



Pavel Rusakov was born in 1972 in Riga, Latvia. He received his Bc. sc. ing., Mg. sc. ing. and Dr. sc. ing. from Riga Technical University (RTU) in 1993, 1995 and 1998 respectively. He received the Diploma with distinction: Mg. sc. ing.

He is an Associate Professor with the Institute of Applied Computer Systems, RTU. He is Head of laboratory and is responsible for the Professional Bachelor and Professional Master studies in the Department of Applied Computer Science. His fields of interest are computer science, programming paradigms, object-oriented approach to systems development, parallel computing, Web technologies, distributed systems, computer graphics, and protection of information.

E-mail: Pavels.Rusakovs@cs.rtu.lv