Guarantee interoperability by publishing Open Government Data as Resource Description Framework

Fabrizio Parrillo[1, *]

[1] *Department for Mathematics and Computer Science, University of Basel*

(Dated: December 28, 2018)

In our digital age, we produce new data every day. This holds also for government institutions, where most of the data created is paid from tax money. Therefore, institutions and governments all over the world now publish data on Open Government Data Platforms. This positive development should be further strengthened, by making the data available for the citizens thought application. The published data must be interoperable, to encourage developers to build applications, and decrease the administrative expense. In this paper, we show that by publishing as Resource Description Framework we can guarantee interoperability by design. Therefore, we compare two structured file formats, a delimiter-separated values file format, and Resource Description Framework.

Keywords: Resource Description Framework; Open Government Data; Interoperability

I. INTRODUCTION

Open Government Data (OGD) can be described as data and information produced or commissioned by the government, or government-controlled agencies, that is made available for public consumption [1][2].

The initial initiative was launched in 2009 by the U.S government, claiming that openness will ensure the public trust, establish a system of transparency, public participation, and collaboration. The countries Denmark, Spain, Switzerland, the U.S, and U.K elaborated openness strategies which explicitly correlated openness with innovation, new businesses, and economic benefits[3] [4].

The opendata.swiss portal is a product of the "Open Government Data Strategy for Switzerland 2014-2018" which ends at the end of the year. It is a joint project of the confederation, cantons, communes and other organizations with a mandate from the state. It makes open government data available to the general public in a central catalog which is operated by the Swiss Federal Archives[5]. Currently, there are 6'346 datasets from 24 categories which were published by 65 organizations. Thanks to the published records, 39 new apps have been developed.

The World Wide Web Consortium (W3C) recommends publishing data in a structured file format, called Resource Description Framework (RDF), which corresponds to 5 Star data on the 1-5 Star metric defined by T. Berners-Lee[7][8] . By using RFD, the data becomes machine readable in an automated and generic fashion [6]. Unfortunately, many publishers do not follow this recommendation. Table I shows, almost all data is published as 2 Star (XLS) or 3 Star data (TSV, CSV). Hence, without considering 1 Star data, Html and Services, the primary source is spreadsheet data. In most cases, Services are Application Programmable Interfaces (API).

* E-mail me at: f.parrillo@protonmail.ch

File format	Count	%
Html	3787	33.11
XLS	1359	11.88
TSV	1345	11.74
Services	915	8.00
CSV	895	7.83
PDF	893	7.81
Multiform	507	4.43
zip	507	4.43
Json	385	3.37
wms	209	1.83
wfs	140	1.22
shapefile	120	1.05
less than 1% *	337	3.30

TABLE I: Number of files published on opendata.swiss in the respective file format.
* in decresing order (KMZ, WMTS, TiFF, XML, Interlist, Wordfile, ODS, GeoJson, GeoTiff, TXT, WCS, Png)

A. Goal

The aim of this work is to emphasize the benefits of publishing OGD data as RDF. Therefore, we show how an application which uses RDF as data sources can seamlessly consume new published data or newly added data sources. Furthermore, we will define a simple cookbook example which shows how to convert "delimiter-separated values" file format (DSV), such as TSV and CSV, to RDF. That corresponds to converting of 3 Star to 5 Star data.

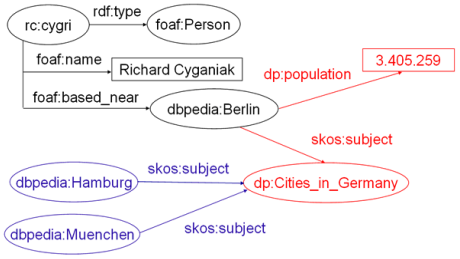

FIG. 1: Shows an example RDF graph of an person living in Berlin

II. BACKGROUND

A. Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a framework for representing information in the Web. RDF is a graph-based data model where each entry is defined by a triplet $\langle subject, predicate, object \rangle$

$$(\langle s, p, o \rangle).$$

Each item of the triplet can have one of the *RDF terms* $\mathbb{T}$, namely a blank node $b \in \mathbb{B}$, a Uniform Resource Identifier (URI) $u \in \mathbb{U}$, or a literal $l \in \mathbb{L}$. We denote $\mathbb{T}$ as the set $\mathbb{T} = \mathbb{B} \cup \mathbb{U} \cup \mathbb{L}$ and where

$$s \in \mathbb{B} \cup \mathbb{U} \qquad p \in \mathbb{B} \cup \mathbb{U} \qquad o \in \mathbb{T} \qquad [9, p.\ 11][10].$$

Figure 1 shows, an example of an RDF graph which describes a person living in Berlin. Since s and o can both be URI's, Berlin is an object in this $\langle cygri, based\_near, Berlin \rangle$ and a subject in the $\langle Berlin, subject, Cities\_in\_Germany \rangle$. Notice that the graph described by the person's RDF file contains just the back graph, but since the URI from dbpedia is used to describe the persons living place, the graph expands[11]. RDF can be serialized either as XML or turle.

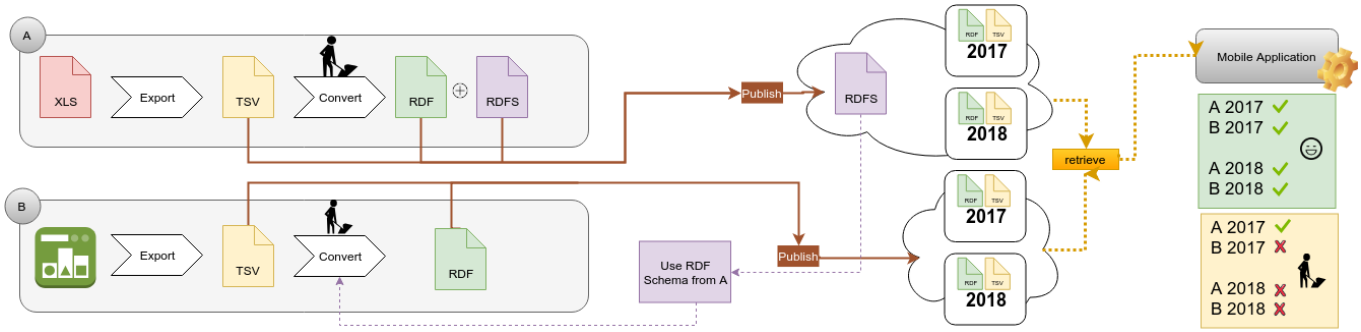

FIG. 2: Illustration of the workflow and data life cycle of the two publishers, A and B

B. Protocol and RDF Query Language

SPARQL is a query language for RDF. It can be used to express queries across diverse data sources. [9]

III. METHODS

To analyze the effects of structured data on mobile applications, we define a set D of example data sets from two weather stations A and B, which publish data for the year 2000 and 2001, denoted as 0 and 1. Figure 2 shows, both publish a data set as TSV and RDF, which the application retrieves and consumes.

$$D = \{A^0_{csv}, A^0_{rdf}, A^1_{csv}, A^1_{rdf}, B^0_{csv}, B^0_{rdf}, B^1_{csv}, B^1_{rdf}\}$$

A. Data

Figure 2 shows, both weather stations A and B have to convert their spreadsheet to TSV, which is subsequently converted to RDF. We focus on these two steps and how their output affects the application.

1. *Spreadsheet*

The test data consists of 4 spreadsheets. Each Table (II, III, IV, V) has one entry which consists of three columns; *place*, *temperature* and *date*. By introducing inconsistencies, we show the consequence which might affect the application. Therefore, the data differs in table header, formatting and column position.

The headers from A and B differ due to language, since A is located in the German part of Switzerland, while B is located in the Italian canton. Furthermore, A uses spreadsheets and B uses proprietary software to store the weather data. To conclude, the headers in B^1 changed to English due to a software update. The update also affected the formatting of the date, which changed from MM-DD-YYYY to DD-MM-YYYY. Another formatting difference between A and B is the decimal separator "." and ",". Finally, the table columns *place* and *temperature* have been swapped from A^0 to A^1.

Ort	**Temperatur**	**Datum**
Basel	25.5	6.6.2000

TABLE II: Spreeadsheet data A^0

Temperatur	**Ort**	**Datum**
25.5	Basel	7.7.2001

TABLE III: Spreeadsheet data A^1

temperatura	**posto**	**data**
4,3	Locarno	20-2-2000

TABLE IV: Spreeadsheet data B^0

temperature	**place**	**date**
4,3	Locarno	2-20-2001

TABLE V: Spreeadsheet data B^1

2. *TSV*

The spreadsheets are well-formatted, containing a header for each column in the first row. Therefore, the spreadsheets are converted to TSV by using the export feature available in Microsoft Excel. Figure 3 shows an example of the conversation for A^1_{csv} .

```
Ort     Temperatur      Datum
Basel   25.5            6.6.2000
```

FIG. 3: Converted spreedsheet data A^0 to A^0_{csv}

3. *RDF*

The primary use case of RDF is to be read by machines. Usually, the URIs, which should be used to define the triplet, are long. That makes it unpleasant to read or write RDF by hand. Therefore, we advise generating it programmatically. There are many libraries in different programming languages, which we could choose. We used RDF4j, which is a Java Library, to read, write, and query RDF files. The Algorithm 1 shows the main framework on how to convert a TSV file to RDF.

Algorithm 1 Convert TSV to RDF

```
rdfFile ← createNewRdfFile
rows ← parseTSVFile
for row ∈ rows do
    subject ← getSubjectFrom(row)
    for column ∈ row do
        predicate ← getPredicateFrom(column)
        object ← getObjectFrom(column)
        addTriplet(rdfFile, subject, predicate, object)
    end for
end for
write(rdfFile)
```

B. Test environment

Usually, the data would be hosted on a server, and a client application would retrieve the data over HTTP(S). The cost and complexity to run this experiment with a server would not contribute additional value. Therefore, the data is stored and retrieved from a local folder. Further simplifications, are made on the client application. Since we are only interested in how the application's data layer can retrieve and correctly map the data to the domain model, we just implement the data layer which we denote as *fetcher*.

From Fetcher, there are three different implementations. There are two TSV fetchers which differ in the property they use to achieve the mapping by using once the *headers*, and secondly the *column number*. Lastly, the third fetcher uses SPARQL to query and the RDF properties to map the result to the data model. Therefore, the same library RDF4J is used to provide an im-

plementation of SPARQL. Furthermore, all fetchers prior intent is to operate on the data set A^0 or B^0.

Since the client side application is fully implemented in Java, we use JUnit - which is a testing framwork for Java - to check our test cases. Each test checks if the data has been correctly mapped to the data model. The first use case tests the intraoperability by checking the republished data sets A^1 and B^1. The second use case tests the interoperability by checking on full set D.

IV. RESULTS

The main aim of this work is to compare the intra- and interoperability of structured and unstructured data. Therefore, we validated the three fetchers, fixed column (C), fixed header (H), and SPARQL (S) on intraoperability by testing $F^{A^0}_{\{C,H,S\}}$ on $\{A^0, A^1\}$, $F^{B^0}_{\{C,H,S\}}$ on $\{B^0, B^1\}$ and the interoperability by testing $F^{A^0}_{\{C,H,S\}}$ on $\{B^0, B^1\}$, $F^{B^0}_{\{C,H,S\}}$ on $\{B^0, B^1\}$. The test results are presented in Tables VI, VII and VIII.

There are six different outcomes which we observed by trying to retrieve and map the data to the domain model.

- $\checkmark_1$ Data has been mapped correctly
- X_1 No information has been parsed, resulting in a empty string
- X_2 Could not cast to float
- X_3 Mapped data to wrong context
- X_4 Semantic error
- X_5 Could not cast to date

To clarify the outcomes, we present an example for each one. It is trivial to say, that applying the fetcher on its intended data set will always work ($\checkmark_1$). If the headers differ, as in the fetcher $F^{A^0}_H$ applied on B^0, no column is extracted, which will result in an empty string for each column (X_1). If the Fixed Column Fetcher A^0 is applied on A^1, it tries to map the place, which is a string to the temperature, which in turn is a float resulting in a cast error (X_2). Furthermore, it maps the temperature value from B^0 to the place. Since the value of place is a string, the retrieved temperature value is mapped to the place (X_3). Since the date format changed from B^0 to B^1, the day and month are wrongly mapped (X_4). Finally, if different value separators are used such as in A^0 and B^0 regarding the date, it will result in a cast error (X_5).

The error rates for each fetch regarding the intra- and interoperability are presented in Table IX. The error rate shows that 2 Star data is not interoperable at all. Furthermore, they cannot guarantee seamless adoption to new published data, which can be derived by an error rate of 25% on intraoperability. On the other hand, by using 5 Star data, the error rate can be reduced to 0%.

Fetcher	*Data set*	**temp**	**place**	**date**
$F_C^{A^0}$	A^0	✓$_1$	✓$_1$	✓$_1$
	A^1	X_2	X_3	✓$_1$
	B^0	X_2	X_3	X_5
	B^1	X_2	X_3	X_5
$F_C^{B^0}$	A^0	X_2	X_3	X_5
	A^1	X_2	X_3	X_5
	B^0	✓$_1$	✓$_1$	✓$_1$
	B^1	✓$_1$	✓$_1$	X_4

TABLE VI: Test result for fetcher with fixed column.

Fetcher	*Data set*	**temp**	**place**	**date**
$F_H^{A^0}$	A^0	✓$_1$	✓$_1$	✓$_1$
	A^1	✓$_1$	✓$_1$	✓$_1$
	B^0	X_1	X_1	X_1
	B^1	X_1	X_1	X_1
$F_H^{B^0}$	A^0	X_1	X_1	X_1
	A^1	X_1	X_1	X_1
	B^0	✓$_1$	✓$_1$	✓$_1$
	B^1	X_1	X_1	X_1

TABLE VII: Test result for fetcher with fixed header.

Fetcher	*Data set*	**temp**	**place**	**date**
$F_R^{A^0}$	A^0	✓$_1$	✓$_1$	✓$_1$
	A^1	✓$_1$	✓$_1$	✓$_1$
	B^0	✓$_1$	✓$_1$	✓$_1$
	B^1	✓$_1$	✓$_1$	✓$_1$
$F_R^{B^0}$	A^0	✓$_1$	✓$_1$	✓$_1$
	A^1	✓$_1$	✓$_1$	✓$_1$
	B^0	✓$_1$	✓$_1$	✓$_1$
	B^1	✓$_1$	✓$_1$	✓$_1$

TABLE VIII: Test result for fetcher with RDF

Fetcher	**Error Rate in %**	
	Intraop.	Interop.
Fixed column	25.0	100
Fixed header	25.0	100
SPARQL	0	0

TABLE IX: Aggregaded error rates for the three fetchers.

V. DISCUSSION

As shown in Table VI and VII, where the data is published as 3 Star data (TSV), the application might result in one of the five possible errors while consuming the data. The aggregated errors for the fixed header and the fixed column both result in an error rate of 25% for intraoperability and 100% for interoperability. That is not an acceptable error rate for productive application. But we have to interpret these results with caution since the data has been chosen consciously, such that many negative effects would occur on 3 Star data fetchers. This does not mean that application consuming 3 Star data have a high probability of running into these errors. All 39 application which have been developed by data sets published on open-data.swiss do not consume 5 Star data. Nevertheless, they are fully functional and do not show any affects described in this paper. The reason for this is that the app developers have taken care to harmonize the data and make sure that the data is converted and mapped correctly to their application data model.

The strength of 5 Star data is that an application can consume data automatically. That is obviously visible in Table IX, since the error rate is zero. That is because the publisher took care to harmonize the data before publishing. One could now argue that we just want to transfer the effort from the developer to the publisher. But in our opinion, the publisher is in charge of it, since the data is published once and consumed multiple times by different applications.

A. Conclusion

We have shown that by using RDF, we can decrease the errors to zero regarding retrieving and mapping the data to the application data model. Not only can we therefore guarantee the seamless adoption to new published data, but also the combination of data from different publishers.

Unfortunately, the conversion from 1-3 Star data to 5 Star data takes some effort, which most publishers, are not willing to make. Furthermore, some background knowledge in RDF and data modeling and programming is currently needed to preform the conversion. Since publishers without a technical background are not likely to have this knowledge, they will probably decide to publish the data as 1-3 Star data [13]. That is disagreeable, since the publishers have in-depth expert domain knowledge. They would thus be the perfect candidates to accomplish the conversion and the mapping modeling.

Therefore, future research should, firstly, try to abstract the overhead of knowing XML or RDF, such that a broader group of people could profit from the benefits of publishing data as RDF. Secondly, we should further investigate how the process could be automated by using machine learning or a probabilistic model, which uses existing vocabulary and RDF data to make predictions about its semantic and structure. The user could then choose the best matching prediction. Thirdly, we should also further investigate how we could improve the integration of RDF in the developer's workflow. Therefore, one could try to generate source classes, which could be derived from the RDF Schema (RDFS), similarly, to how Java Classes are generated by the XML Schema (XSD)[12].

[1] E. Lakomaa and J. Kallberg, IEEE Access **1**, 558 (2013).

[2] K. Rexed, C. Moll, and N. Manning, (2007), 10.1787/210083427643.

[3] D. S. Sayogo, T. A. Pardo, and M. Cook, Proceedings of the Annual Hawaii International Conference on System Sciences , 1896 (2014).

[4] M. A. Elahi, Program Office E-Government Switzerland (2014).

[5] Program Office E-Government Switzerland, "Open government data strategy for switzerland 2014 – 2018," (2014).

[6] B. Hyland, G. A. Atemezing, and B. Villazón-Terrazas, *Best Practices for Publishing Linked Data*, W3C Note (W3C, 2014) http://www.w3.org/TR/2014/NOTE-ld-bp-20140109/.

[7] B. Hyland, G. Atemezing, M. Pendleton, B. Srivastava, and US Environmental Protection Agency, 3 Round Stones , IBM, *Linked Data Glossary*, W3C Note (W3C, 2013) https://www.w3.org/TR/ld-glossary/.

[8] T. Berners-Lee, *Linked Data*, W3C Note (W3C, 2006) https://www.w3.org/DesignIssues/LinkedData.html.

[9] A. Harth, K. Hose, and R. Schenkel, *Linked Data Management* (2014).

[10] M. Lanthaler, R. Cyganiak, and D. Wood, *RDF 1.1 Concepts and Abstract Syntax*, W3C Recommendation (W3C, 2014) http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[11] C. Bizer, R. Cyganiak, and T. Heath, "How to publish linked data on the web," .

[12] C. Fry and S. Ziegler, "Systemand method for xml data bnding," US 7,962,925 B2.

[13] E. Klein, , S. Haller, A. Gschwend, and M. Jovanovik,