A Method for Converting Current Data to RDF in the Era of Industry 4.0

Marlène Hildebrand, Ioannis Tourkogiorgis, Foivos Psarommatis, Damiano Arena, Dimitris Kiritsis

A method for converting current data to RDF in the era of Industry 4.0

Marlène Hildebrand[1*], Ioannis Tourkogiorgis[1], Foivos Psarommatis[1], Damiano Arena[1], Dimitris Kiritsis[1]

[1]École polytechnique fédérale de Lausanne, ICT for Sustainable Manufacturing, EPFL SCI-STI-DK, Lausanne, Switzerland
marlene.hildebrand@epfl.ch

Abstract. In the past two decades, the use of ontologies has been proven to be an effective tool for enriching existing information systems in the digital data modelling domain and exploiting those assets for semantic interoperability. With the rise of Industry 4.0, the data produced on assembly lines within factories is becoming particularly interesting to leverage precious information. However, adding semantics to data that already exists remains a challenging process. Most manufacturing assembly lines predate the outbreak of graph data, or have adopted other data format standards, and the data they produce is therefore difficult to automatically map to RDF. This has been a topic of research an ongoing technical issue for almost a decade, and if certain mapping approaches and mapping languages have been developed, they are difficult to use for an automatic, large-scale data conversion and are not standardized. In this research, a technical approach for converting existing data to semantics has been developed. This paper presents an overview of this approach, as well as two concrete tools that we have built based on it. The results of these tools are discussed as well as recommendations for future research.

Keywords: Ontology, zero-defect manufacturing, data integration, RDF, semantics, JSON.

1 Introduction

What might make Industry 4.0 a unique industrial revolution, is its capacity to rely on already existing infrastructures, just by interconnecting technologies. In that regard, the data produced by machines on assembly lines make for a particularly interesting case to investigate. Assembly lines in factories currently produce very large amounts of data, most of which is under-exploited, often simply stored to never be used again. At the same time, factories want to insure better interoperability between their services and tools. To do so, a more and more widespread approach is to add meaning to the data [1] through the use of semantics, or to be more precise, through the use of ontologies, allowing for data to be machine-understandable, and understood the same way across diverse systems of information [2][3][4]. The purpose of ontologies is to help mark-up data with meaningful metadata that will make it more easily interpretable. To

formalize this new manner to structure data, a language named RDF (Resource Description Framework) [5] has been developed, which expresses data in the form of triples subject-predicate-object, and where URIs (Uniform Resource Identifier) symbolize the instances. URIs help identify a specific instance, and link it to its attributes, or to other instances. What RDF allows us to do, is to formalize our data as a graph. This is a very powerful way of storing unstructured data, and quickly retrieve the connections between data, unlike SQL (Structured Query Language) joints that are very difficult to express and expensive.

But this raises one big issue. Graph data is fairly new and has only been widely adopted in certain niches, such as the market of social networks. In the meanwhile, relational data has remained the bread and butter of most data management systems for the past 20 years. To exchange and communicate data between different systems, formats such as JSON (JavaScript Object Notation), which relies on a key-value pair paradigm, have become widely popular. Therefore, the data produced by assembly lines is mostly structured as tables, key-value pairs or trees, and comes in formats such as SQL databases, XML (Extensible Markup Language) files, Excel sheets or JSON streams. Some of these formats are difficult to convert into graph data because of their very structure, such as SQL, and others, like JSON, may only be easy to translate in certain cases, depending on how well the schema of the data matches with the ontology.

Therefore, some pre-processing is required to be able to leverage this data, which involves mapping the existing schema of the data to the ontologies we have designed, and then implement this mapping in a computer program that will achieve an ETL (Extract-Transform-Load) procedure to in order to handle the data accordingly. This raises several questions: how do we map an existing data schema to an ontology, and how to we formalize it? And once we have formalized this mapping, how do we implement it in a tool in order to automatically convert the data?

As we will see in the next section, methodologies to map non-graph data schemas to graph data schemas, and more specifically to RDF, is a very active field of research, and several recommendations, languages and tools have already been created to achieve this mapping. However, no standard has been issued yet, and some of these tools have limitations. This brings us to the second issue, which is the one we would like to address in this paper: How do we then use these methodologies to implement a tool that can automatically convert data? How can we "generalize" this tool so that it can be applied to different data formats, different data schemas, and different ontologies? Such a tool, or at least a general conversion algorithm, would make the life of software developers easier, which is a crucial step for the more widespread adoption of RDF and ontologies.

Therefore, the current research aims to present a general, easy to implement algorithm, that can automatically convert data to RDF without being dependent on the original data schema or the targeted ontology. Section 2 describes the state of the art in data mapping. Section 3 describes the approach that was proposed for this research, and the tools that have been used. Section 4 describes two applications of this approach and Section 5 discusses the findings and their implications.

2 State of the art

Methodologies to convert non-graph data to graph data are still an ongoing field of research in the world of linked data and semantics, since the solutions to this issue could significantly increase the adoption of RDF. Mapping SQL data schemas to RDF has been a particular topic of focus, since most databases are relational. Even though the World Wide Web Consortium has issued no standard yet in that regard, it has issued a few recommendations. The most notable one deals with how to directly map SQL to RDF [6]. R2RML, a language for mapping SQL to RDF has also been issued and recommended by the Consortium [7].

Regarding on how to map JSON schemas with RDF, another language, JSON-LD (JavaScript Object Notation for Linked Data), has been created by members of the W3C [8] and has gained quite a lot of attention due to its simplicity of use and its practicality. RML, an extension of R2RML, can also be used to map JSON and other formats to RDF [9]. A language named GRDDL [10] has been developed quite early on during the development of the Semantic Web in order to map XML to RDF, but it also possible to use it for the mapping of JSON to RDF. A language named XSPARQL has also been developed to query XML and RDF and to transform data from one format to the other [11].

As we can see, there are many methodologies and languages to map existing data schemas of all kinds of formats to ontologies. The questions then arise: how do we provide this mapping to a software tool, and how do we implement the conversion of the data? One way would be to simply use these methodologies as they are, and to explicitly implement the mapping in the code. There are several problems with this approach. First, explicitly hard-coding values in a program is bad practice in the world of software development, because every evolution of the data schemas will have to be implemented in the code as well. Also, the code wouldn't be reusable for different data schemas or different ontologies. Another issue is that in the case of assembly lines, the amounts of data to process and integrate are huge, and so are the ontologies we have to map them to. Explicitly coding the mapping therefore implies that each data will go through one test for each different field of the data schema, which is doable for small mappings, but would be a tedious work and come at a great cost of performance for really big ontologies and data schemas.

What we would like to achieve instead is to come up with a general conversion algorithm, one that would be able to automatically process the data based on a mapping we would have provided it, but that wouldn't need to be aware of the details of the ontology or of the original data schema, so that we would be able to feed it any mapping, or to adapt it to any data format in input. Something lightweight that could be easily implemented in different programming languages, which would have good performances regarding time and memory consumption, and that would be easy to understand and maintain for software developers.

3 Proposed approach

As stated previously, the intent behind this paper is to present a general implementation to automatically convert non-graph data to RDF through a software tool, without

having to know the data schema or the ontology in advance, so that it could work with any mapping. We have tried to develop tools based on this implementation to convert data coming from XML files and JSON streams into RDF. Our idea was to develop a technique for mapping that would rely on an approach similar to what is currently done with configuration parameters for software tools. Instead of hard-coding the values of the configuration parameters in the software, they are usually stored in a separate file, under the form of key-value pairs. When the code is running, the value is retrieved through the key called in the code. This allows for an easy way to modify the parameters of a desktop application or a web service (e.g. the access path to a database) when necessary, without going through the expensive path of a code evolution.

Our approach regarding the formalization of the mapping was very similar in that we decided to provide it as a JSON file that the code could parse to retrieve the corresponding ontology concept for each data schema field (Fig.1). One file would be dedicated to classes and another to the properties, since the process for creating a new class instance is not the same as the one for creating the property of an instance. In the JSON files, the fields of the XML or JSON data schema would be handled as the keys, would paired to the URIs of the classes or properties relationships they correspond to in the ontology (Fig.2).

In the code of the software tool, two functions will need to be implemented: one for the creation of a new instance of a class, and another for the creation of a property of this instance. The first function will create a new URI and create an RDF triple expressing of which class this URI is an instance of. This triple will consist of this new URI as a subject, “rdf:type” as a predicate to indicate the “is an instance of” relationship, and the URI of a class as an object. The second function will create a triple that will assign a new property of a certain value to a certain class instance. To do so, it will create a triple that consist of the URI of the instance, the URI of the relationship to the property, and the value of said property.

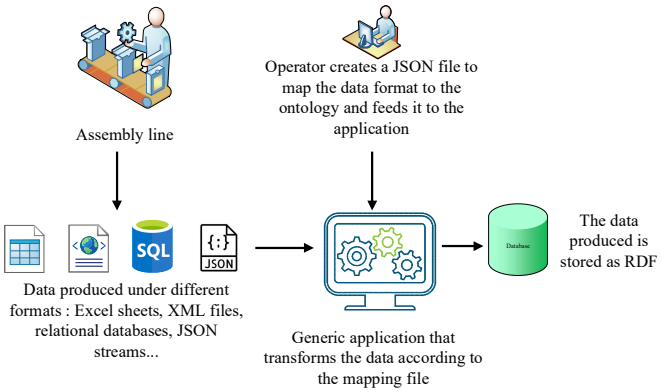

Fig. 1. General architecture of the proposed solution

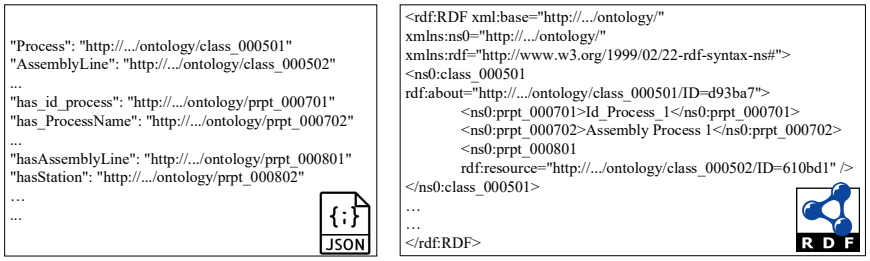

Fig. 2. Mapping JSON file (left); RDF output file (right)

The next step is to create an algorithm using these functions. The first thing this algorithm will have to do is to parse the file or the stream it will receive in input. For each new object in the file, it will retrieve the JSON key or XML field that corresponds to this object, it will check if it corresponds to a class in the class mapping file, and if it is the case, create a new class instance. If the key or field corresponds instead to a property, the algorithm will retrieve the URI of the relationship in the property mapping file and link the URI of the object to the value in the file (Fig.3). As we will see later in this paper, this approach works well granted that, in the case we are creating a property, we have a way to know which class instance it has to be linked to. This information is easy to retrieve depending on the original data schema, but it is also the only part of the algorithm that cannot be generalized.

4 Application cases

4.1 Semantic enhancement to ease aircraft design and lifecycle cost data integration

The first software tool we have developed based on this approach was for a project aimed at digitalizing the life cycle ground testing of aircrafts. The purpose of the tool was to be able to feed it an XML file containing the data to be processed to output a Turtle[1] file containing the original data but descried using RDF instead of XML. Our approach was particularly well adapted to this case, since the original XML data schema consisted of objects matching the ontology classes, and attributes matching the ontology properties. Also, objects that had to be linked together were imbricated in each other, which made it easier to link one instance to the other algorithmically. The tool we created for this use-case was developed in C# with the framework .Net Core 2.1[2]. We created two JSON files: one that mapped the names of the XML objects to the corresponding ontology class URIs, and another that mapped the XML attributes to the corresponding property URIs. The structure of the algorithm is the following: while reading the XML file, for each new XML object, the algorithm would look-up the name

[1] A syntax and file format to express data in RDF. https://www.w3.org/TR/turtle/
[2] Free, open-source framework from Microsoft. https://dotnet.microsoft.com/download

of the object in the JSON file and retrieve the corresponding class URI, and would then create a new line in the Turtle file consisting of a triple for a new instance of a class. It would then read the attributes of this object, and for each attribute, lookup the corresponding property relationship URI in the JSON file to create a triple in the Turtle file comprised of the abovementioned class instance URI, the property relationship URI, and the value of the attribute in the XML sheet.

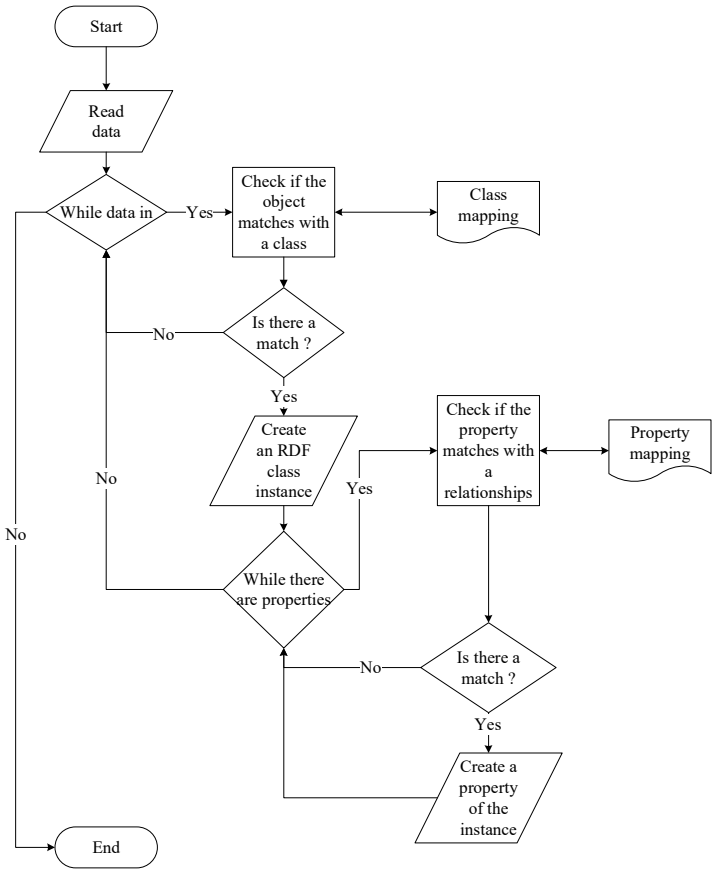

Fig. 3. Flowchart of the proposed algorithm

The only issue that needed to be solved was how to link instances together, when a property was referring to an instance and not just a literal. As stated earlier, the solution was easy due to the structure of the XML file. Whenever the algorithm was creating a new instance, it would save this instance URI, and when reading a new object and creating its new URI, it would link these two URIs together by looking up the property

relationship corresponding to the new object name in the property mapping file. Then the algorithm would be able to create a new triple with the previous instance URI as the subject, the relationship URI as the predicate, and the new instance URI as the object.

4.2 Zero-Defect Manufacturing

Another ETL tool has been developed in the context of the Z-Fact0r project. This project aims at improving the quality and the productivity of the European manufacturing industries through five multi-stage production-based strategies. These include: the early detection of the defect, the prediction of the defect generation, the prevention of defect generation as well as its propagation in later stages of the production, the reworking/remanufacturing of the product and finally the management of these strategies through event modelling, KPI monitoring and real-time decision support.

In this case, the original format was a JSON stream, therefore consisting of key-value pairs, and the created RDF triples would have to be stored in a triple store powered by Apache Fuseki[3]. In order to save the triples to the triple store, we used a library named DotNetRDF[4]. Each JSON object corresponded to the instance of a class, and certain key-value pairs corresponded to a property of this object. Others could be ignored.

If the overall algorithm remained the same, this time, attention had to be given to the fact that different JSON objects within the stream could refer to the same instance, and therefore it would be necessary to be able to retrieve the URI of a previously created instance. Our solution to this issue was to use the ID of the object as the URI to the corresponding instance. This way, when reading a new object, we would not need to know if it already existed or look for its ID: just by concatenating the base URI of the ontology and the value of the "id" key-value pair, we would be sure to refer to the correct instance.

Another issue was that the key-value pairs could refer to literal properties but also to other class instances. One way to handle that would be to test beforehand if the key does exist in the class mapping instead of the properties mapping, or to create a separate file for properties that refer to an instance of another class rather than just a literal value. Then the algorithm could distinguish this case and create an instance of the adequate class and link it to the object instance. But this raises a few questions as well. In our case, this approach worked because the value stored in these key-value pairs was the id of the instance, which we could use to create a new URI. But what if several values in the JSON object are related to this instance and not the overall object? Then it will not be possible anymore to apply a general treatment to the key-value pairs, and it will be necessary to create specific treatments for specific fields.

---

[3] HTTP interface to RDF data, developed as a servlet. https://jena.apache.org/documentation/fuseki2/

[4] .Net library for parsing, managing, querying and writing RDF. http://www.dotnetrdf.org/

5 Conclusion

In this paper, a generic algorithm for the automatic conversion of different types of data into RDF has been presented, and it was demonstrated that it can contribute to quickly and easily building ETL processes for data that has to be transformed and described with RDF. The main contribution of this research is to propose a solution that doesn't require to hard-code the mapping, and that can be reusable for different data schemas and different ontologies, and that can be easily adapted to other data formats.

Nonetheless, as we have seen, this methodology does not function with every data schema in input. Ideally, for this generic treatment to be efficient, the data structure in entrance should follow the logic of the ontology, class instances and property value should be easily distinguishable. Also, the id of an object should be easily retrievable when referring to an already existing instance, in order to facilitate the work of developers. Otherwise, they will have to look for the instance in the already saved RDF triples, which would kill performance. These issues need to be addressed through future research and tools development.

6 Acknowledgments

The work presented in this paper is partially supported by the project Z-Fact0r which is funded by the European Union's Horizon 2020 program under grant agreement No 723906.

References

1. O. Verscheure and D. Kiritsis "The Meaning of Data", WMF Report 2018
2. J. L. Martinez Lastra and I. M: Delamer, "Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap", 2006
3. S. Iarovyi, W. M. Mohammed, A. Lobov, B. R: Ferrer, J. L. Martinez Lastra, "Cyber–Physical Systems for Open-Knowledge-Driven Manufacturing Execution Systems", 2016
4. B. Kádár, W. Terkaj, M. Sacco, "Semantic Virtual Factory supporting interoperable modelling and evaluation of production systems", 2013
5. G. Klyne, J. J. Carroll, B. McBride, R. Cyganiak, D. Wood, M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", World Wide Web Consortium, 2004-2014. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/
6. J. F. Sequeda, M. Arenas and D. P. Miranker, "On Directly Mapping Relational Databases to RDF and OWL (Extended Version)," 2012.
7. S. Das, S. Sundara and R. Cyganiak, "R2RML: RDB to RDF Mapping Language," World Wide Web Consortium, 2012. [Online]. Available: https://www.w3.org/TR/r2rml/.
8. M. Sporny, D. Longley, G. Kellogg, M. Lanthale and N. Lindström, "JSON-LD 1.1: A JSON-based Serialization for Linked Data," World Wide Wen Consortium, 2018. [Online]. Available: https://www.w3.org/TR/json-ld11/.
9. A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens and R. V. d. Walle, "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data", 2014
10. D. Connolly, "Gleaning Resource Descriptions from Dialects of Languages (GRDDL)," World Wide Web Consortium, 2007. [Online]. Available: https://www.w3.org/TR/grddl/.
11. S. Bischof, S. Decker, T. Krennwallner, N. Lopes and A. Polleres, "Mapping between RDF and XML with XSPARQL", 2012