Pattern-Based Ontological Transformations for RDF Data using SPARQL

Marek Suchánek
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
email: marek.suchanek@fit.cvut.cz

Robert Pergl
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
email: robert.pergl@fit.cvut.cz

***Abstract*—RDF data are being described by ontologies (OWL or RDFS) to state the meaning and promote interoperability or so-called machine-readability. However, there are many overlapping ontologies that one can use for a single dataset. To overcome this issue, mappings between ontologies are made to capture the relations forming the overlaps. Such mappings can be used with inference and reasoning tools, but rewriting rules must be applied to transform the dataset. This work proposes a new way of transformations builds on top of the SPARQL query language. It uses defined RDF patterns representing modules that can be interrelated. Our method's primary focus is for larger-scale transformations where existing methods require hard-to-maintain, i.e., non-evolvable mapping definitions. A brief demonstration, as well as a comparison with other transformation languages, is provided.**

***Index Terms*—ontology, transformation, RDF, OWL, versatility, evolvability**

I. Introduction

The technologies around the Resource Description Framework (RDF), including the Web Ontology Language (OWL), are more and more used in various domains and setups. It is a versatile instrument for capturing any knowledge. As RDF is the keystone of Semantic Web and Linked Data, it helps to form interoperable metadata and data. The OWL ontologies and RDF schemas help to describe the internal structure and meaning of related RDF data. Without such descriptions, the RDF data are just a set of triples. On the other hand, with a used ontology, it is possible to make assumptions about types, properties, and even infer new data from existing.

Therefore, more and more ontologies are introduced to different domains, projects, and purposes. Sometimes those ontologies are just in the form of dictionaries, where specific terms are defined for referring to them. In contrast, others are full-fledged ontologies with a definition of properties and special relations, such as subclassing. The overlaps between the ontologies are inevitable just as it needs to evolve over time. Again, the principles of linked data can help with that. Two ontologies can be easily linked together, for example, to define equality of terms just as any other relation.

Issues arise when the underlying RDF data needs to be transformed between such ontologies based on the mapping. The first issue is with the evolvability of the transformation itself, i.e., how to handle changes from the linked ontologies. Then, there are issues with possibilities of executing the transformation over datasets that might contain non-mapped entries. Finally, the consistency between the original and the transformed RDF dataset is also an important issue. There are different approaches to the RDF transformations that we will review and evaluate in this paper, mainly regarding the evolvability.

Section II briefly describes the necessary terminology and existing solutions for transforming RDF data that we use as a source of valuable information and experience. The proposed method for pattern-based ontological transformations if presented in Section III. In Section V, we summarize and discuss the results and outline possible future steps to enhance RDF transformation's evolvability.

II. Related Work

In this section, we briefly introduce important terminology and provide an overview of the current possibilities for RDF data transformations.

A. RDF, RDFS, and OWL

RDF is a set of specifications based on a simple idea that everything can be described using so-called semantic triples or triplets – subject, predicate, object [1]. Such triplets can also be expressed as a graph; therefore, we use also terms as knowledge graph or graph database when talking about RDF data. As it was initially intended as a metadata model, the triples consist of identifier, more specifically Uniform Resource Identifiers (URI), to specific resources and possibly literals on the object position. RDF Schema (RDFS) [2] or the Web Ontology Language (OWL) [3] can be used to define classes, properties, or constraints to bring structure into otherwise unstructured RDF data. Although the OWL capabilities in terms of expressiveness and versatility are much higher than RDFS, both are expressed again as RDF data using triplets. Nowadays, RDF technologies are used widely in many activities and for many kinds of data [2]. RDF, RDFS, and OWL are the basis of Semantic Web and Linked Data but are also used in data-intensive research (e.g., biology or chemistry) or in conceptual modelling for specifying dictionaries and taxonomies [4].

 ISBN: 978-1-61208-850-1

RDF provides versatility in capturing any information both as data and related metadata. For example, there are various well-defined ways of capturing versions of data. Dublin Core [5] defines `hasVersion`, `isVersionOf`, or `valid` terms for annotating the data. There are even more elaborated vocabularies, such as Changeset [6], that allows annotating data with a changelog. However, for OWL ontologies, the version information is usually included both in metadata but also as part of URIs, e.g., `http://purl.org/dc/elements/1.1/date` that is a persistent snapshot, and without version part, the identifier targets the latest, which can change over time. Moreover, OWL provides additional properties for versioning, such as `versionInfo` or `incompatibleWith`.

B. Ontology Mapping

OWL itself defines a fundamental way of mapping. Again, due to the versatility of RDF and the fact that OWL ontologies are captured as RDF data, the mapping can be done using well-defined properties. Using the core principles of RDF, it is possible to import two (or more) ontologies and create a mapping between them or create an extension build on top of them.

For capturing that two individuals have the same identity, `owl:sameAs` can be used. It is essential to mention here that even OWL Classes are themselves individuals, so the property can be used to express class equality on the identity level. For just stating that two classes are equivalent (but not necessarily have the same identity), `owl:equivalentClass` is appropriate. There are many more standard properties to define different relations of (in)equality. As it is still just RDF with given meaning to property, one can define own mapping properties but need to explain the purpose and determine how it should be implemented.

C. STTL: SPARQL-Based Transformation

The approach of [7] is based on the SPARQL Protocol and RDF Query Language. As SQL is well-known and used for querying relational data, SPARQL works similarly (even in terms of syntax) for RDF data, i.e., semantic triples. The SPARQL-Based Transformation (STTL) uses SPARQL extension `TEMPLATE` with additional functions, such as `st:apply-templates`. The transformation allows to query and prepare data for a template using a standard `SELECT` query. Then, it applies a template by the additional functions. Output can be practically any textual format as a template is just a string with marks where place specific data as shown in Figure 1. The authors present examples, including RDF-to-HTML or RDF-to-RDF/Turtle.

Nevertheless, for just RDF-to-RDF transformations, the standard `CONSTRUCT` query can be used. It avoids installing the extension, yet it provides a simple way of a mapping definition. The RDF Data are prepared using the `WHERE` clause and then used to build new triples. One can see such a SPARQL query also as a text; therefore, it can be synthesized using another tool or script and even executed programmatically.

```
TEMPLATE {
    "The triple is: " ?x ", " ?p ", " ?y "!"
}
WHERE {
    ?x ?p ?y
}
ORDER BY ?x ?p ?y
```

Fig. 1. Example of STTL template query

D. RML: RDF Mapping Language

The RDF Mapping Language (RML) [8] is a generic mapping language, based on and extending R2RML. Although it is still in state of unofficial draft, it captures interesting and novel ideas related to RDF mapping. The main focus is on transforming data from textual formats, such as CSV, XML, or JSON into RDF. The mappings in RML are captured as RDF triples as shown in Figure 2. First, there is a definition of a data source, such as CSV file. Then, there are several mappings for subjects, predicates, and objects that capture what classes and properties are used and how the data are queried from the data source [9] [10].

```
<#PersonHobbyMapping>
  rml:logicalSource [
    rml:source "http://ex.com/people.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$.person.hobby.[*]"
  ];

  rr:subjectMap [
    rr:template "http://ex.com/{Hobby_ID}";
    rr:class ex:Hobby
  ].
```

Fig. 2. Example of RML mapping for JSON source

E. TRIPLE Language

A Query, Inference, and Transformation Language for the Semantic Web (TRIPLE) [11] is an older representative of RDF transformation language. In contrast to the previous two approaches, TRIPLE has its foundations in mathematics and logics. It can be expressed both through RDF (e.g. RDF/XML) and mathematical syntax as it is an extension of Horn logic similar to F-Logic. The models (i.e. sets of triples) are first class citizens in TRIPLE. It has several useful functions for defining the transformations, such as reification of statements or path expressions. Unfortunately the tooling support is not easily available but it has been also used for business agents [12].

III. TRANSFORMING RDF DATA USING PATTERNS AND SPARQL

In this section, we describe our design of the pattern-based ontological transformations for RDF data using SPARQL as

well as a prototype implementation. The advantages of our approach when compared to using plain SPARQL or other solutions are discussed in Section V. First, we outline set the problem statement and related requirements. Then, steps of our method are explained in the subsequent subsections.

A. The Problem and Requirements

Our work aims to design a transformation method that takes a set of triples as input together, and using a set of transformation rules produces a new set of triples. Although SPARQL `CONSTRUCT` query allows defining such rules, it does not provide re-usability of definitions nor modularization. Our solutions must promote evolvability by design as a fine-grained modular structure concerning the Normalized Systems Theory [13], especially the Separation of Concerns and Data Version Transparency principles. However, SPARQL already provides a fair way of querying RDF data that should be re-used for defining the transformation patterns as visualized in Figure 3.

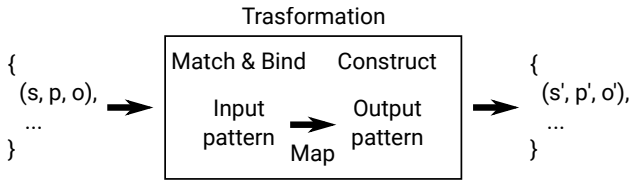

Fig. 3. Architecture of the pattern-based RDF transformation

B. Patterns Mapping Definition

In our method, each transformation pattern is formed by two sets of triples – input and output. The principle and syntax are the same as in SPARQL; the input pattern can bind variables in any position (subject, predicate, object). Then, the output triples can contain those variables multiple times. RDF data are traversed, and the input pattern is filled, i.e., variables are substituted by values. When the whole input pattern is matched, the output pattern is populated with the values for variables and RDF data are added to the result dataset. For defining both input and output patterns, standard RDF prefix definitions might be necessary (but separately for each). So far, the operation is the same as with a set of SPARQL `CONSTRUCT` queries.

To introduce a fine-grained modular structure from trivial cases, each pattern has separated input and output parts, as shown in Figure 4 and Figure 5. The variable names from the input part are exposed, and the output part can use them. It must be defined what input is used (cases with multiple input parts are explained further). Using variables in this way can be seen as a coupling inside the module.

C. Pattern Modules and Submodules

The definition of input pattern must take into account that RDF/OWL works with open-world assumptions. For example,

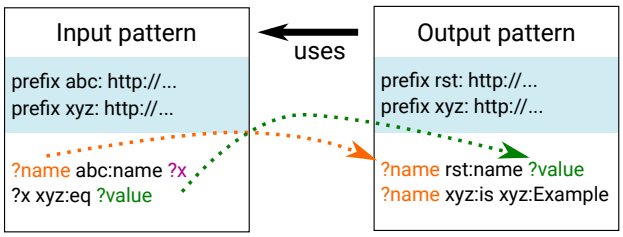

Fig. 4. Concept of relating input and output patterns

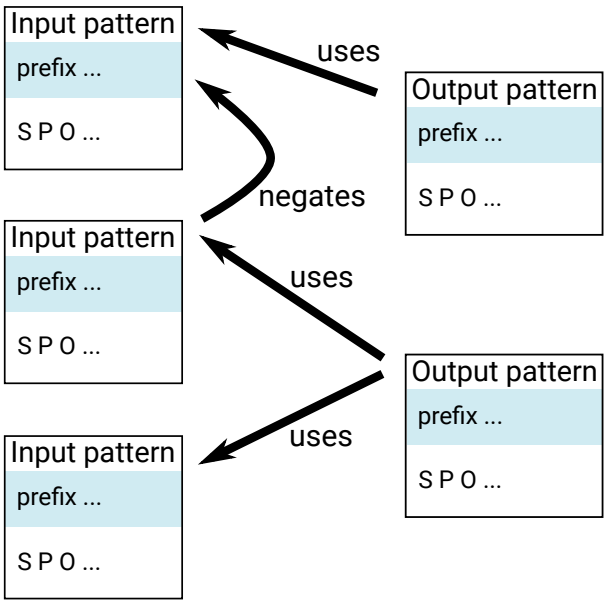

Fig. 5. Combining input patterns for output patterns

if the second pattern should be used only if the first one is not matched, it must include negating the first input triples. To do this efficiently, the input triples must be shareable across patterns. By importing triples, one must comply with its defined variable names or rename them. In a similar way, it can serve to separate the concerns when the same input produces multiple different outputs for a single dataset, e.g., a name of a person is turned into the name of the personal folder at HR and the name on a cup in a shared kitchen.

We call a set of related patterns in this way a *patterns module* and each of its part is a submodule. In a trivial case without any shared part, a pattern forms itself (input and output triples with prefix definitions) a module. As input can re-use multiple other inputs (with renaming the variables), the output can also define various input definitions to be used, as shown in Figure 5.

 ISBN: 978-1-61208-850-1

D. Generating and Executing SPARQL CONSTRUCT

The pattern definitions and modules capture the knowledge of what inputs should be used for producing what outputs. With that, it is possible to generate a set of SPARQL `CONSTRUCT` queries to be executed in arbitrary order over an input RDF dataset. There are several steps to this procedure for each patterns module:

1) Resolve imports in all input definitions, including variable renaming.
2) For each output definition, import input definition(s) including variable renaming.
3) Merge used prefixes and use renaming mechanism for conflicts (when name and URI do not match).
4) Generate SPARQL CONSTRUCT query with input part in WHERE clause.
5) Execute the query over the input dataset and add result into output dataset.

E. Prototype Implementation

A prototype of the method has been implemented in Python using rdflib [14] and Jinja2 [15] templating language as a CLI application. It takes a folder with pattern modules and an RDF source (a file or SPARQL endpoint) and produces an output RDF file. The project folder has a subfolders per each module, where files are prefixed by `input_` and `output_`. The re-usability of definitions is realized through variables and imports in Jinja2. Each of the modules forms a single Jinja2 environment where each output file is loaded with substituted Jinja2 variables. The compiled transformation pattern is then turned using an internal template into a SPARQL query and executed over a graph imported from the input file or using the given SPARQL endpoint.

The patterns are executed actually in alphabetical order as that is the way the filesystem is traversed. We added a possibility to randomize the order of transformation to demonstrate that it is irrelevant. After all of the patterns are successfully applied, the output file is written out. Several issues might appear when running the transformation. For instance, the pattern's definition may contain a syntactic error; in that case, the application reports where the problem is. For the SPARQL endpoint, there might be connection issues. Finally, there might also be a problem when output definition is using a variable that is not defined in any of the inputs.

IV. Example Case: FOAF and vCard

To demonstrate the use of our method and prototype, we want to transform a dataset about people created using Friend-of-a-Friend ontology FOAF [16] into a dataset according to the vCard ontology [17]. Some of the presented patterns are intentionally not optimal in terms of complexity to show the use of additional features. The examples put together related patterns forming a single module, as explained previously. Individual patterns are marked using comments starting with # symbol.

A. Simple Personal Information

The first example in Figure 6 shows a trivial case where is just a single input pattern and a single output pattern. For every person with a name from the input dataset, a name according to vCard is created. The used syntax is simplified RDF Turtle with custom directives. Here `@input` serves to specify the only input pattern based on its name. No variable renaming is used. During the transformation, this case is simply turned into a SPARQL query, i.e., prefixes are merged together (no conflicts), the input pattern is inserted into `WHERE` clause, and the output becomes the body of `CONSTRUCT`.

```
# Input pattern: input_simple_foaf
@prefix rdf:
 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

?person rdf:type foaf:Person .
?person foaf:name ?name .

# Output pattern: output_simple_vcard
@input: input_simple_foaf .
@prefix vcard:
 <http://www.w3.org/2006/vcard/ns#> .

?person vcard:fn ?name .
```

Fig. 6. Trivial case for single input pattern to single output pattern

B. Negated Input Pattern

Figure 7 demonstrates a case where the second input pattern negates the first one using `@negate` directive. The SPARQL query takes the first input pattern and wraps it using `NOT EXISTS` construct. This case also has a conflict in prefixes as the second input pattern is using a different FOAF version. During the processing, the conflicting prefixes are renamed before turned into a SPARQL query.

```
# Input pattern: input_foaf_person_and_org
@prefix rdf:
 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

?agent rdf:type foaf:Person .
?agent rdf:type foaf:Organization .

# Input pattern: input_foaf_membership
@negate: input_foaf_person_and_org .
@prefix foaf: <http://xmlns.com/foaf/0.2/> .

?agent foaf:member ?member .

# Output pattern: output_simple_vcard
@input: input_foaf_membership .
@prefix vcard:
 <http://www.w3.org/2006/vcard/ns#> .

?agent vcard:member ?member .
```

Fig. 7. Example of using a negated input

 ISBN: 978-1-61208-850-1

C. Multiple Input Patterns

The final example Figure 8 shows multiple input patterns used for an output pattern. It creates both person triples and organization triples (with names). The second `@input` directive is enhanced with a variable renaming dictionary to avoid name clashes. In this case, all variables are renamed, but it is not mandatory, and only some can be renamed. Both input patterns use the same FOAF prefix, and thus it will be merged without the need of solving conflicts.

```
# Input pattern: input_foaf_person
@prefix rdf:
 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

?person rdf:type foaf:Person .
?person foaf:name ?name .

# Input pattern: input_foaf_organization
@prefix rdf:
 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

?organization rdf:type foaf:Organization .
?organization foaf:name ?name .

# Output pattern: output_complex
@input: input_foaf_person .
@input: input_foaf_organization
        {organization: org, name: orgName} .
@prefix rdf:
 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix vcard:
 <http://www.w3.org/2006/vcard/ns#> .

?person rdf:type vcard:Individual .
?person vcard:fn ?name.

?org rdf:type vcard:Organization .
?org vcard:title ?orgName.
```

Fig. 8. Example with multiple inputs and renaming variables

V. Discussion

This section summarizes the advantages of our approach and confronts it with the other existing solutions introduced in Section II. The planned future steps are outline as well.

A. Evolvability and Maintainability

Our solution is designed to allow the definition of interlinked patterns for transformations. Each pattern relates an input set of triples where some variables are bound, and the output set of triples is used. If the patterns are not related to each other, there are no ripple effects internally. Also, thanks to the declarative approach of RDF, the order of triples and, thus, an order of transformation execution is irrelevant. To mitigate ripple effects of relating the patterns through sharing (or negating) input triples, we introduced pattern modules where if an input set is changed, it requires a change in submodules but not other modules. To allow transformation with intermediate layer of temporary triples, our solution can be used in a pipeline, e.g., first, transform input to some intermediary triples and then transform intermediary triples to output. The order of such transformations is, of course, necessary, but internally the order of executing a transformation pattern is still irrelevant, i.e., there are no dependencies.

As for the practical maintainability of the transformation defined as a set of patterns, the key is the overall solution's simplicity. Each pattern can be defined as a standalone file with input and output or re-use other definition on input. When there is an update of some of the imports (e.g. a target ontology changes), it spreads over patterns as a cross-cutting concern. However, only patterns related to the target ontology are required to be updated, which cannot be avoided. Versioning of a set of patterns (a project) can be done easily through standard VCS tools, including branching or version tagging.

B. Comparison to Existing Solutions

When compared to the solutions mentioned in Section II, our approach has a different focus. RML [8] targets transforming any data to RDF. To use it for our purpose, we would need to define a mechanism for using RDF as a data source and navigating through it, which would be additional overhead. Moreover, mappings' definitions do not provide such maintainability as in our case, despite other similarities. STTL [7] allows us to query RDF data and transform it into any textual format, which would again cause additional overhead here as we would need to synthesize RDF. This functionality is supported directly by SPARQL `CONSTRUCT`.

Mappings on the OWL-level, such as `owl:sameAs`, can be used together with our method as is shown in Section IV. It provides a generalized way of executing transformations based on those mappings (and others defined in transformation patterns). Finally, TRIPLE [11] has a concept of models as the building block, whereas we similarly use patterns. Our solution can be in mathematical terminology seen as a set of functions (each realizing a pattern transformation). It takes a set of triples and gives a set of triples. On the contrary to this last method, we also implemented the prototype actually to execute the transformations.

C. Future Steps

The presented approach, together with the prototype implementation, can be already used for executing the transformations, as shown in Section IV. However, we plan to enhance the ease of writing the transformation patterns as well as executing the transformation over various kinds of data sources (e.g. local files or SPARQL endpoints). The first of the use cases that will be taking advantage of our transformation solution is working with different conceptual models encoded in RDF. The ultimate goal here is to develop mappings between metamodels so the models can be integrated and transformed on the semantical level. As we expect non-trivial sets of transformations patterns required to achieve the goal, it will also serve as a verification and a good source of potential enhancements to our method. Finally, we plan to continue

 ISBN: 978-1-61208-850-1

in this effort to create a user-friendly editor for specifying, testing, and executing the pattern transformations.

VI. Conclusion

This paper proposed and demonstrated a new approach for transforming RDF data using patterns specified for SPARQL `CONSTRUCT` queries. It was shown that our solution's evolution and maintainability is improved as the core principles of Normalized Systems were taken into account. When compared to other existing methods for transforming RDF data, the primary focus is slightly different. Whereas other targets transform other data formats into RDF or vice versa, our contribution is supporting the transformation of RDF data between different ontologies where basic mapping predicated from OWL is not sufficient. Moreover, our approach's advantages are the most visible when used with more complex use cases where many and overlapping transformation patterns are needed. Finally, we plan to continue in this effort to create a user-friendly editor for specifying and testing the pattern transformations.

Acknowledgment

The research was supported by Czech Technical University in Prague grant No. SGS20/209/OHK3/3T/18.

References

[1] A. Hogan, *The Web of Data*. Springer International Publishing, 2020.

[2] Dan Brickley and Ramanathan V Guha and Brian McBride, "Rdf schema 1.1," *W3C recommendation*, vol. 25, pp. 2004–2014, 2014.

[3] P. Hitzler et al., "OWL 2 Web Ontology Language Primer," *W3C recommendation*, vol. 27, no. 1, p. 123, 2009.

[4] S. Powers, *Practical RDF*. O'Reilly Media, Incorporated, 2003.

[5] Dublin Core Metadata Initiative, "Dublin Core™ Metadata Element Set, Version 1.1: Reference Description," 2012. [Online]. Available: https://www.dublincore.org/specifications/dublin-core/dces/ 2021.03.30

[6] S. Tunnicliffe and I. Davis, "Changeset," 2009. [Online]. Available: https://vocab.org/changeset/ 2021.03.31

[7] O. Corby and C. Faron-Zucker, "STTL - A sparql-based transformation language for RDF," in *WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015*, V. Monfort, K. Krempels, T. A. Majchrzak, and Z. Turk, Eds. SciTePress, 2015, pp. 466–476. [Online]. Available: https://doi.org/10.5220/0005450604660476 2021.03.29

[8] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, , and R. V. de Walle, "RML: A generic language for integrated RDF mappings of heterogeneous data," in *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014*, ser. CEUR Workshop Proceedings, vol. 1184. CEUR-WS.org, 2014. [Online]. Available: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf 2021.03.25

[9] B. D. Meester, P. Heyvaert, R. Verborgh, and A. Dimou, "Mapping languages: Analysis of comparative characteristics," in *Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics co-located with 16th Extended Semantic Web Conference (ESWC 2019), Portorož, Slovenia, June 3, 2019*, ser. CEUR Workshop Proceedings, vol. 2489. CEUR-WS.org, 2019, pp. 37–45. [Online]. Available: http://ceur-ws.org/Vol-2489/paper4.pdf 2021.03.22

[10] P. Heyvaert, A. Dimou, R. Verborgh, E. Mannens, and R. V. de Walle, "Towards approaches for generating RDF mapping definitions," in *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015*, ser. CEUR Workshop Proceedings, vol. 1486. CEUR-WS.org, 2015. [Online]. Available: http://ceur-ws.org/Vol-1486/paper_70.pdf 2021.03.22

[11] S. Decker et al., "TRIPLE - an RDF rule language with context and use cases," in *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*. W3C, 2005. [Online]. Available: http://www.w3.org/2004/12/rules-ws/paper/98 2021.03.28

[12] M. Sintek and S. Decker, "Using TRIPLE for business agents on the semantic web," *Electronic Commerce Research and Applications*, vol. 2, no. 4, pp. 315–322, 2003. [Online]. Available: https://doi.org/10.1016/S1567-4223(03)00040-1

[13] H. Mannaert, J. Verelst, and P. D. Bruyn, *Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design*. Kermt (Belgium): Koppa, 2016.

[14] RDFLib Team, "rdflib 5.0.0," 2020. [Online]. Available: https://rdflib.readthedocs.io 2021.03.31

[15] Pallets, "Jinja Documentation (2.11.x)," 2021. [Online]. Available: https://jinja.palletsprojects.com 2021.03.31

[16] D. Brickley and L. Miller, "FOAF Vocabulary Specification 0.99," 2014. [Online]. Available: http://xmlns.com/foaf/spec/ 2021.03.31

[17] R. Iannella and J. McKinney, "vCard Ontology - for describing People and Organizations," 2014. [Online]. Available: https://www.w3.org/TR/vcard-rdf/ 2021.03.31

 ISBN: 978-1-61208-850-1