Received May 25, 2020, accepted June 6, 2020, date of publication June 12, 2020, date of current version July 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002018

PGO: Describing Property Graphs in RDF

DOMINIK TOMASZUK[1], **RENZO ANGLES**[2], **AND HARSH THAKKAR**[3,4]
[1]Institute of Informatics, University of Bialystok, 15-245 Białystok, Poland
[2]Department of Computer Science, Faculty of Engineering, Universidad de Talca, Curicó 3340000, Chile
[3]OSTHUS GmbH, 52068 Aachen, Germany
[4]Informatik III Department, University of Bonn, 53115 Bonn, Germany

Corresponding author: Dominik Tomaszuk (d.tomaszuk@uwb.edu.pl)

The work of Dominik Tomaszuk was supported by the National Science Center (NCN), Poland under Grant Miniatura 2. The work of Renzo Angles was supported by the Millennium Institute for Foundational Research on Data, Chile.

ABSTRACT RDF and Property Graphs are data models that are being used to represent Knowledge Graphs. The definition of methods to transform RDF data into Property graph data is fundamental to allow interoperability among the systems using these models. Although both models are based on a graph structure, they have special features that complicate the definition of data transformation methods. This article presents an ontology-based approach to transform (automatically) property graphs into RDF graphs. The ontology, called PGO, defines a set of terms that allows describing the elements of a property graph. The algorithm corresponding to the transformation method is described, and some properties of the method are discussed (complexity, data preservation, and monotonicity). The results of an experimental evaluation are also presented.

INDEX TERMS RDF, property graphs, data transformation, OWL, ontology.

I. INTRODUCTION

A graph is a powerful abstract model that allows representing different types of interconnected data. Its use in different application domains (like social networks, biological sciences, multimedia, and geography [1]), has motivated the production of large graph-based datasets, and the development of graph-oriented systems and technologies. In this paper, we concentrate our interest in two popular approaches for graph data management: Resource Description Framework (RDF) databases and Property Graph (PG) databases.

RDF database systems (also called Triple Stores) [2] are systems designed to store and query linked data [3]. These systems are based on Semantic Web standards, in particular the Resource Description Framework (RDF) [4], [5] and the SPARQL query language [6]. RDF is a graph data model that allows to describe the attributes and relationships of web resources in the form of labeled, directed multigraphs.

Property Graph database systems [7] are systems designed to store and query property graphs. A Property Graph [8] is an extension of a labeled directed graph where nodes and edges can have properties (i.e. property-value pairs). There are no current standards for property graph databases. However, the authors of this article are participating in working groups concerning the future standards[1]. In particular, there is an ISO/IEC project created to design a standard query language for property graphs[2].

Given the importance of RDF and PG database systems in the area of graph data management, we decided to study the interoperability between them. The interoperability between database systems is relevant for several reasons [9]: promotes data exchange and data integration [10]; facilitates the creation, reuse, sharing and querying of data [11], [12]; extends the use of available systems and tools [13]; enables a fair comparison of systems (by using benchmarks) [14], [15]; allows to explore and compare the best features of different approaches and systems [16].

In March of 2019, the W3C Workshop on Web Standardization for Graph Data[3] joined people from academy and industry related to graph data management. One of the objectives of the workshop was to create bridges between the adjacent worlds of RDF and property graphs. The attendants agreed with respect to the importance of developing standard methods to allows the interoperability between PG and RDF database systems.

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

[1]https://www.gqlstandards.org
[2]https://www.iso.org/standard/76120.html
[3]https://www.w3.org/Data/events/data-ws-2019/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

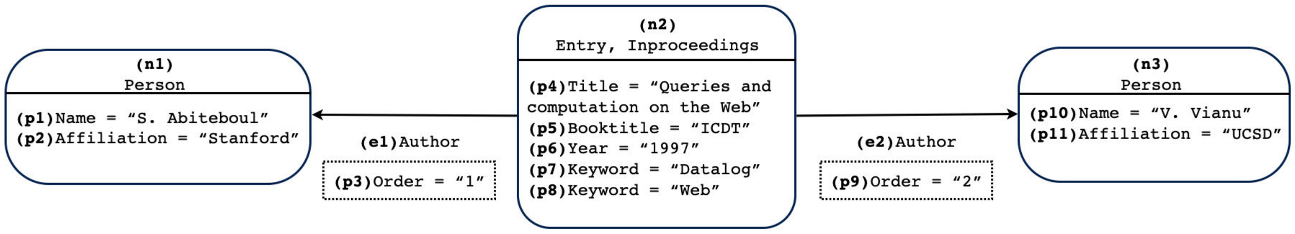

FIGURE 1. Graphical representation of a property graph.

Database interoperability concerns the development of applications that exchange and share information across the boundaries of existing databases [10], [17]. A common way to support interoperability among database systems is the definition of data transformation methods. Although the RDF data model and the property graph data model are based on a graph-like structure, they have special features that complicate the definition of data transformation methods. To the best of our knowledge, the research about such methods is very limited (see the related work in Section VI).

1) CONTRIBUTIONS

In this article, we present an automatic ontology-based method to transform property graph data into RDF data. The main contributions are the description of the proposed ontology (called PGO), the definition of the algorithm to transform PG data to RDF data using the PGO ontology, and the implementation of the proposed transformation method.

2) LIMITATIONS

According to the classification criteria presented in [18], our proposal is automatic and does not support manual customization to model domain semantics. Moreover, our proposal omits schema mapping including extracting domain information from the input schema. This is due to the lack of a uniform support for schemas in property graph database systems.

3) ORGANIZATION

The remainder of this article is organized as follows: Section II presents the main concepts related to property graphs and RDF graphs. Section III introduces PGO, an ontology to describe a property graph using RDF. Section IV includes the description of the transformation method. Section V presents the experimental evaluation of our approach. Section VI is devoted to discuss the related work. We close the article with some conclusions.

II. PRELIMINARIES

In this section we present formal definitions for the property graph data model, the RDF data model, and Web Ontology Language (OWL). These definitions will be the basis to describe our transformation method.

A. PROPERTY GRAPHS

In general terms, a property graph is a directed labelled multigraph whose main characteristic is that nodes and edges could maintain a set of properties. A property is composed of a name and a value, and such value has a specific datatype. Nodes, edges, and properties could have labels that are used to describe their roles, types or classes in the data domain. Figure 1 shows an example of property graph.

Most of the current graph database systems are designed to manipulate property graphs. However, each system supports a particular set of features allowed by the property graph data structure (see Table 1). Next, we provide a formal definition of the property graph data model which includes all the features presented in Table 1.

Assume that L is an infinite set of labels (for nodes, edges and properties), V is an infinite set of (atomic or complex) values, and T is a finite set of data types (e.g. string, integer, date, etc.). Given a value $v \in V$, the function type(v) returns the datatype of v. The values in V will be distinguished as quoted literal. Given a set S, we will use $\mathcal{P}(S)$ to denote the power set of S (i.e. the set of all subsets of S, including the empty set and S itself).

Definition 1 (Property Graph): A property graph is defined as a tuple: $G = (N, E, P, \delta, \lambda, \sigma, \rho)$ where:

1) N is a finite set of nodes, E is a finite set of edges, P is the finite set of properties, and N, E, P are mutually disjoint sets;
2) $\delta : E \to (N \times N)$ is a total function that associates each edge in E with a pair of nodes in N;
3) $\lambda : (N \cup E) \to \mathcal{P}(L)$ is a partial function that associates nodes and edges with a set of labels (possibly empty);
4) $\sigma : (N \cup E) \to \mathcal{P}(P)$ is a partial function that associates nodes and edges to a set of properties (possibly empty), satisfying that $\sigma(o_1) \cap \sigma(o_2) = \emptyset$ for each pair of objects $o_1, o_2 \in \text{dom}(\sigma)$;
5) $\rho : P \to (L \times V)$ is a total function that assigns a label-value pair to each property.

The above definition supports property graphs with the following features: each node or edge can have zero or more labels; each node or edge can have zero or more properties; each pair of nodes can be connected by zero or more edges; each node or edge can have multi-value properties (i.e. multiple properties with the same property name).

TABLE 1. Property graph features supported by graph database systems.

Systems	Node labels			Edge labels			Edges				Properties		
	Zero	One	Many	Zero	One	Many	Directed	Undirected	Multiple	Duplicated	Mono-value	Multi-value	Null-value
Neo4j	●		●		●		●		●	●		●	
Datastax		●			●		●		●	●		●	
OrientDB		●			●		●	●	●	●		●	●
ArangoDB		●			●		●	●	●	●		●	●
JanusGraph	●	●			●		●	●	●	●		●	
Amazon Neptune	●		●		●		●		●	●		●	
TigerGraph		●			●		●	●	●			●	
InfiniteGraph		●			●		●	●	●	●		●	●
InfoGrid	●		●		●		●	●			●		●
Sparksee		●			●		●	●	●	●	●		●
Memgraph	●		●		●		●		●	●		●	●
VelocityDB		●			●		●	●	●	●		●	●
AgensGraph	●		●		●	●	●		●	●		●	
TinkerGraph		●			●		●		●	●		●	
HGraphDB		●			●		●		●	●	●		●

Figure 1 shows a graphical representation of a sample property graph containing bibliographic information based on the BibTeX format. This sample graph is formally defined as $G = (N, E, P, \delta, \lambda, \sigma, \rho)$ where:

- $N = \{n_1, n_2, n_3\}$,
- $E = \{e_1, e_2\}$,
- $P = \{p_1, \ldots, p_{11}\}$,
- $\delta(e_1) = (n_2, n_1)$,
  $\delta(e_2) = (n_2, n_3)$,
- $\lambda(n_1) = \{\texttt{Person}\}$,
  $\lambda(n_2) = \{\texttt{Entry}, \texttt{Inproceedings}\}$,
  $\lambda(n_3) = \{\texttt{Person}\}$,
- $\sigma(n_1) = \{p_1, p_2\}$,
  $\sigma(e_1) = \{p_3\}$,
  $\sigma(n_2) = \{p_4, p_5, p_6, p_7, p_8\}$,
  $\sigma(e_2) = \{p_9\}$,
  $\sigma(n_3) = \{p_{10}, p_{11}\}$,
- $\rho(p_1) = (\texttt{Name}, \texttt{"S. Abiteboul"})$,
  $\rho(p_2) = (\texttt{Affiliation}, \texttt{"Stanford"})$,
  $\rho(p_3) = (\texttt{Order}, \texttt{"1"})$,
  $\rho(p_4) = (\texttt{Title}, \texttt{"Queries ...on the Web"})$,
  $\rho(p_5) = (\texttt{Booktitle}, \texttt{"ICDT"})$,
  $\rho(p_6) = (\texttt{Year}, \texttt{"1997"})$,
  $\rho(p_7) = (\texttt{Keyword}, \texttt{"Datalog"})$,
  $\rho(p_8) = (\texttt{Keyword}, \texttt{"Web"})$,
  $\rho(p_9) = (\texttt{Order}, \texttt{"2"})$,
  $\rho(p_{10}) = (\texttt{Name}, \texttt{"V. Vianu"})$,
  $\rho(p_{11}) = (\texttt{Affiliation}, \texttt{"UCSD"})$.

B. RDF GRAPHS

The Resource Description Framework (RDF) [4], [5] is a standard data model proposed by the World Wide Web Consortium (W3C) to describe resources (i.e. real or abstract things) occurring in any application domain. The "description of a resource" means an explicit representation of the attributes and relationship of the resource.

Assume that I, B and L are disjoint infinite sets, where I is the domain of web resource identifiers represented as IRIs, B is the domain of anonymous resources called blank nodes, and L is the domain of simple values called Literals.

The RDF data model is based on the notion of an RDF triple. An RDF triple is a tuple (v_1, v_2, v_3) where $v_1 \in I \cup B$ is called the *subject*, $v_2 \in I$ is the *predicate* and $v_3 \in I \cup B \cup L$ is the *object*. Here, the subject represents a resource, the predicate represents either an attribute or relationship of the resource, and the object represents either the value of the attribute or the related resource, respectively.

A set of RDF triples intrinsically represents a labeled directed multi-graph where the nodes represent subjects and objects, and the edges represent the predicates. Hence, an RDF graph is the description of resources for a given application domain.

An RDF document is a document that encodes an RDF graph in a concrete RDF syntax, such as Turtle [19] or RDF/XML [20]. In order to facilitate the presentation of examples, we will use the syntax defined by Turtle.

Syntactically, an IRI is very similar to a URL, and can be abbreviated by using a prefix definition. For instance, a full URI `http://www.example.org/person1` can be abbreviated as the prefixed name `ex:person1`, where `ex:` is a prefix that represents the namespace `http://www.example.org/`. In this article we use the prefixes and namespaces shown in Table 2.

A blank node is usually represented by an expression of the form `_:b123` where `b123` is a label that works as a local identifier. There are other ways to encode blank nodes (e.g. `[]`), but we will use the above for simplicity.

We will consider two types of literals: a simple literal which is a collection of Unicode characters

TABLE 2. Prefixes, namespaces and domains used in this article.

Prefix	IRI	Domain
`rdf`	http://www.w3.org/1999/02/22-rdf-syntax-ns#	Resource Description Framework
`rdfs`	http://www.w3.org/2000/01/rdf-schema#	RDF Schema
`owl`	http://www.w3.org/2002/07/owl#	Web Ontology Language
`xsd`	http://www.w3.org/2001/XMLSchema#	XML Schema Datatypes
`swrl`	http://www.w3.org/2003/11/swrlx	Semantic Web Rule Language (SWRL)
`pgo`	http://ii.uwb.edu.pl/pgo#	Property Graph Ontology
`ex`	http://example.com/	Namespace for examples
`skos`	http://www.w3.org/2004/02/skos/core#	Simple Knowledge Organization System
`bag`	http://www.ontologydesignpatterns.org/cp/owl/bag.owl#	Parts and Collections: Bag
`set`	http://www.ontologydesignpatterns.org/cp/owl/set.owl#	Parts and Collections: Set
`list`	http://www.ontologydesignpatterns.org/cp/owl/list.owl#	Parts and Collections: List
`foaf`	http://xmlns.com/foaf/0.1/	Friend of a Friend ontology
`dblp`	http://dblp.uni-trier.de/rdf/schema-2015-01-26#	DBLP Schema
`dbo`	http://dbpedia.org/ontology/	DBPedia Ontology

(e.g. "`Ontology`"), and a typed literal which consists of a string and a datatype IRI (e.g. "`Ontology`"`^^xsd:string`)[4].

C. OWL

The Web Ontology Language (OWL) [21] is a W3C recommendation designed to describe ontologies. Specifically, it allows to describe classes, properties, individuals, and data values.

In order to describe a domain of interest, OWL defines a set of terms – often called a vocabulary – where each term has a specific meaning, e.g. the term `owl:Class` represents the class of all classes of resources. An OWL ontology can be described using RDF with a precise formal meaning [22]. For instance, an RDF triple of the form (C, `rdf:type`, `owl:Class`) defines that C is a resource of class type.

Consider the prefix names presented in Table 2, which are used in OWL 2. Next, we present the most important OWL terms used to describe the ontology proposed in this article.

- The term `owl:Class` identifies the class of resources that are RDF classes. An RDF triple (C, `rdf:type`, `owl:Class`) defines that C is a class in the data domain.
- (C_1, `rdfs:subClassOf`, C_2) defines that C_1 is a subclass of C_2.
- `owl:DatatypeProperty` identifies the class of properties that link objects to data values. The triple expression (P, `rdf:type`, `owl:DatatypeProperty`) defines that P is a datatype property.
- `owl:ObjectProperty` identifies the class of properties that relates objects to other objects. The expression (P, `rdf:type`, `owl:ObjectProperty`) defines that P is an object property.
- (P, `rdfs:domain`, C) defines that the resource class C is the domain of the predicate P.
- (P, `rdfs:range`, C) defines that the resource class C is the range of the predicate P.
- (P_1, `rdfs:subPropertyOf`, P_2) defines that P_1 is a sub property of P_2.
- (R, `rdfs:label`, *lab*) defines that *lab* is a human readable label (name) for the resource R.
- (R, `rdfs:comment`, *com*) defines that *com* is a human readable description of the resource R.
- (C, `owl:unionOf`, *list*) defines that C is the union of the classes in the collection *list*.
- (P, `rdf:type`, `owl:AnnotationProperty`) defines that P is an annotation property.
- (R, `rdf:type`, `rdfs:Datatype`) defines that R is a (personalized) datatype.

The OWL terms described above will be used next to define an ontology for describing property graphs.

III. THE PROPERTY GRAPH ONTOLOGY (PGO)

In this section, we present our ontology for describing property graphs, called PGO. In order to create the ontology, we followed a methodology for ontology development whose stages will be described below.

A. METHODOLOGY

Several methodologies for ontology development exist [23]–[25]. They are not interchangeable so we decided to use one main methodology and additional elements from two others. The main methodology is Ontology Design 101 (OD101) [26] and the additional methodologies are METHONTOLOGY [27] and Grüninger & Fox [28].

We choose OD101 as a basic methodology because it is strongly associated with OWL. Moreover, OD101 is compatible with Protégé, a software tool that supports the creation of OWL ontologies by using a friendly user interface. METHONTOLOGY is used because it provides the most accurate descriptions of each activity (see [25]). We also use competency questions proposed in the methodology by Grüninger & Fox, because it is a good complement to OD101. We also use some formalisms introduced in [29].

The domain ontology building process is composed of seven main steps presented next.

[4]According to the W3C Specification of RDF 1.1 [5], simple literals are syntactic sugar for abstract syntax literals with the datatype IRI xsd:string.

1) STEP 1: DETERMINE THE DOMAIN

The domain of interest needs to be described and analyzed to obtain the most needed knowledge to build our ontology. We reviewed the definitions of "property graph" in the literature, as well as, extracted core knowledge about that domain. In particular, we studied the functional requirements of the property graph model and its serializations [30].

According to [29], the level of formality is also determined in this step. Such formality depends on whether terms and their meanings are codified in a formal language or a natural language. In our case, we choose the rigorously formal level that is based on Definition 1. This step is based on [26] and [29].

2) STEP 2: DETERMINE THE SCOPE

Specifying competency questions [28] allows us to determine the ontology scope. These questions and their answers are used to extract the main concepts and their properties, relations and formal axioms of the ontology.

Among the main competency questions that cover the main concepts of an ontology for property graphs are:

1) What is the base datatype for a property graph?
2) What are the components of a node?
3) What are the components of an edge?
4) What are the components of a property?
5) What is the meaning of "label"?
6) How to ensure interoperability between the ontology and other graph data formats?

This step is based on [26], [27] and [28].

3) STEP 3: DEFINE THE CLASSES AND CLASS HIERARCHY

To identify the classes in the ontology, Uschold & Grüninger mention three strategies: bottom-up, top-down, and middle-out. We followed a middle-out strategy which suggests identifying first the core of basic terms and then defining and generalizing them as required. This step is based on [26] and [29].

Assume that `pgo` is the prefix name of our ontology, and it is associated with the IRI http://ii.uwb.edu.pl/pgo#. In this step we create five OWL classes:

- `pgo:Graph` which represents an abstract graph;
- `pgo:PropertyGraph` which represents a property graph (`pgo:PropertyGraph` $\sqsubseteq$ `pgo:Graph`);
- `pgo:Node` which represents a node;
- `pgo:Edge` which represents an edge; and
- `pgo:Property` used to define properties for nodes and edges.

With respect to the class hierarchy, we just have that `pgo:PropertyGraph` is subclass of `pgo:Graph`, i.e. a property graph is a special type of graph.

4) STEP 4: DEFINE THE PROPERTIES AND FACTS OF PROPERTIES

The concepts alone will not provide enough information to answer the competency questions. Hence, in this step, we define the properties that connect the classes. For each property, we define its domain, range, and cardinality. This step is based on [26].

We identify the following properties

- `pgo:hasEdge` is used to define the edges of a property graph;
- `pgo:hasNode` is used to define the nodes of a property graph. `pgo:hasNode` should be used to define isolated nodes[5].
- `pgo:startNode` allows to indicate the source node of a property;
- `pgo:endNode` allows to indicate the target node of a property;
- `pgo:hasNodeProperty` is used to define zero or more properties for nodes;
- `pgo:hasEdgeProperty` is used to define zero or more properties for edges;
- `pgo:hasProperty` is used to define zero or more properties for nodes and edges, applying that `pgo:hasProperty` $\sqsubseteq$ `pgo:hasNodeProperty` $\cup$ `pgo:hasEdgeProperty`;
- `pgo:label` is used to define zero or more labels for nodes and edges.
- `pgo:key` allows to define the name of a property;
- `pgo:value` allows to define the value of a property.

Here we have that both, `pgo:hasNodeProperty` and `pgo:hasEdgeProperty`, are sub-properties of the property `pgo:hasProperty`. Our ontology contains a core set of terms to describe a property graph, and we can include SWRL [31] rules to infer other properties e.g. `hasEdge(g,e) ^ startNode(e,n) -> hasNode(g,n)`.

5) STEP 5: DEFINE THE DATATYPES

Usually, an ontology requires the definition of datatypes. The PGO ontology defines three datatypes: `pgo:yarspg`, `pgo:graphml` and `pgo:graphson`. These datatypes have been included to describe the terms using other data formats. These datatypes can be used to attach documentation to the instances defined in PGO. This step is based on [26].

6) STEP 6: INTEGRATE WITH OTHER ONTOLOGIES

Reusing existing ontologies is a requirement when an ontology is thought to interact with other applications. In our case, we re-used the following ontologies and terms:

- GraphJSON datatype as a subclass of `csvw:JSON`;
- GraphML datatype as a subclass of `rdf:XMLLiteral`;
- property value range to support bags (`bag:Bag`) as complex types;
- property value range to support sets (`set:Set`) as complex types;
- property value range to support lists (`list:List`) as complex types.

This step is based on [27] and [26].

[5] Isolated nodes are nodes without any incident edges.

TABLE 3. Restrictions of the properties defined by the PGO ontology.

Domain	Property	Range	Cardinality
`pgo:PropertyGraph`	`pgo:hasEdge`	`pgo:Edge`	*
`pgo:PropertyGraph`	`pgo:hasNode`	`pgo:Node`	*
`pgo:Edge`	`pgo:startNode`	`pgo:Node`	1
`pgo:Edge`	`pgo:endNode`	`pgo:Node`	1
`pgo:Edge` ∪ `pgo:Node`	`pgo:label`	`xsd:string`	*
`pgo:Edge` ∪ `pgo:Node`	`pgo:hasProperty`	`pgo:Property`	*
`pgo:Edge`	`pgo:hasEdgeProperty`	`pgo:Property`	*
`pgo:Node`	`pgo:hasNodeProperty`	`pgo:Property`	*
`pgo:Property`	`pgo:key`	`xsd:string`	1
`pgo:Property`	`pgo:value`	`xsd:string` ∪ `bag:Bag` ∪ `set:Set` ∪ `list:List`	1

7) STEP 7: CREATE THE DOCUMENTATION

Classes, datatypes and properties are usually commented by using the terms `rdfs:comment`, `rdfs:isDefinedBy` and `rdfs:label` [32]. Additionally, the term `rdfs:seeAlso` is used to include further explanation on webpages. These annotation properties are used to provide a human-readable name and description of a resource, and to include additional information about the ontology. This step is based on [27].

B. FINAL ONTOLOGY

In summary, the PGO ontology is formed by the following terms:

- **Resource classes**: `pgo:Graph`, `pgo:PropertyGraph`, `pgo:Node`, `pgo:Edge` and `pgo:Property`;
- **Object properties**: `pgo:hasEdge`, `pgo:hasNode`, `pgo:hasProperty`, `pgo:hasNodeProperty`, `pgo:hasEdgeProperty`, `pgo:startNode`, and `pgo:endNode`;
- **Datatype properties**: `pgo:label`, `pgo:key` and `pgo:value`;
- **Datatypes**: `pgo:graphml`, `pgo:graphson` and `pgo:yarspg`.

Figure 2 shows the relationships between the resource classes and properties. The domain, range and cardinality restrictions of the properties are shown in Table 3. The OWL2/RDF description of the PGO ontology is available at http://ii.uwb.edu.pl/pgo.

The operators used in PGO are atomic negation, universal restrictions, limited existential quantification, role hierarchy, datatypes, concept union and intersection. The ontology contains 78 axioms. The description logic expressivity of PGO is $\mathcal{ALH}(\mathcal{D})$ [33]. The satisfiability reasoning problem for the ontology is EXPTIME-complete: the upper bound is due to correspondence with $\mathcal{SH}$ variant [34].

In addition, the ontology defines datatypes to ensure compatibility with existing graph formats, e.g. GraphML [35], YARS-PG [30]. These datatypes ensure interoperability between PG and RDF. They can be used for additional documentation of the data and for inserting additional (system-specific) annotations, e.g. for better reverse conversion from RDF to PG. The following fragment shows a fragment of Turtle document using a SKOS vocabulary with a GraphML literal:

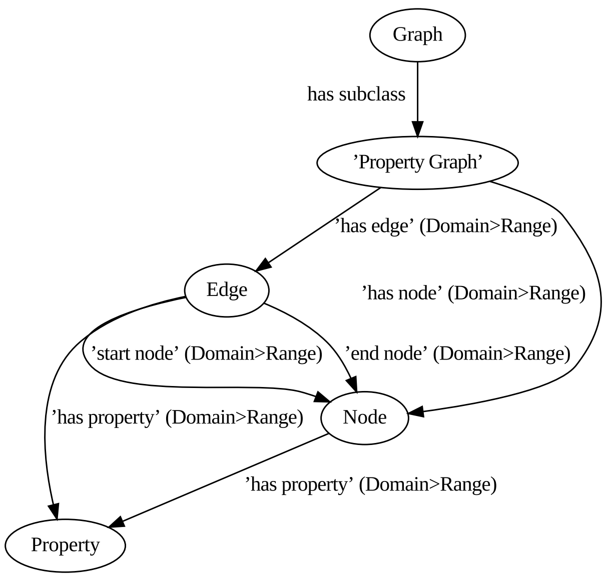
Graph
has subclass
'Property Graph'
'has edge' (Domain>Range)
Edge
'has node' (Domain>Range)
'start node' (Domain>Range)
'end node' (Domain>Range)
'has property' (Domain>Range)
Node
'has property' (Domain>Range)
Property

FIGURE 2. Classes and properties defined by the PGO ontology.

```
_:n skos:example """
<node id="1">
<data key="labelV">Person</data>
<data key="name">John</data>
<data key="age">29</data>
</node>
"""^^pgo:GraphML.
```

IV. DESCRIBING PROPERTY GRAPHS WITH PGO

This section how to use the PGO ontology to describe property graphs. Specifically, we define a data mapping to translate a property graph into an RDF graph.

A. DATABASE MAPPINGS

In general terms, a database mapping is a method to translate databases from a source database model to a target database model. We can consider two types of database mappings: *direct database mappings*, which allow an automatic translation of databases without any input from the user [36]; and *indirect database mappings*, which require additional information (e.g. an ontology) to conduct the database translation.

Let M be a database model. A *database schema* in M is a set of semantic constraints allowed by M. A *database instance* in M is a collection of data represented according to M. A *database* in M is an ordered pair $D = (S, I)$, where S is a schema and I is an instance. In this paper, we are not considering schema information. Hence, we restrict our study to databases instances.

Let M_1 and M_2 be two database models. A *data mapping* from M_1 to M_2 is a function $\mathcal{M}$ from the set of all database instances in M_1 to the set of all database instances in M_2. This notion is used next to transform property graphs into RDF graphs.

B. A DATA MAPPING BASED ON PGO

Let $\mathcal{I}_1$ be the set of all property graphs and $\mathcal{I}_2$ be the set of all RDF graphs. We will define the indirect data mapping $\mathcal{M} : \mathcal{I}_1 \to \mathcal{I}_2$ such that, for any property graph $G \in \mathcal{I}_1$, $\mathcal{M}(G)$ returns an RDF graph $S \in \mathcal{I}_2$ which is based on the PGO ontology.

Let $G = (N, E, P, \delta, \lambda, \sigma, \rho)$ be a property graph. An IRI-assignment for G is a bijective function $\phi : N \cup E \cup P \to I$ that assigns a unique URI to nodes, edges and properties in G[6]. Additionally, assume the existence of a function $\tau(v)$ which allows to transform a property value v into a string literal.

The data mapping $\mathcal{M}$ is defined by the procedure shown in Algorithm 1. The procedure initializes the set S with the RDF triple $(g, \texttt{rdf:type}, \texttt{pgo:PropertyGraph})$ that describes that g is the IRI identifier for the input property graph (line 3). After that, the procedure is divided into two phases: transformation of nodes (lines 4 to 17) and transformation of edges (lines 18 to 36). In both cases there are iterations to transform labels ($l \in \lambda(o)$) and properties ($p \in \sigma(o)$).

Note that each value property v is transformed into a simple RDF literal by using the function $\tau(v)$ (see lines 14 and 33). However, the function τ can be modified to produce typed literals. To do this, we need to assume the existence of a function f which allows to map PG datatypes to RDF datatypes (e.g. $f(\texttt{integer}) = \texttt{xsd:int}$). Then, the function $\tau(v)$ can be defined to return a typed literal of the form "v"^^f(type(v)).

Algorithm 1: $S = \mathcal{M}(G)$

```
Input: A property graph $G = (N, E, P, \delta, \lambda, \sigma, \rho)$ and a
       IRI-assignment function $\phi$.
Output: A set of RDF triples $S$.
1  begin
2      Let $g \in I$ be a IRI for $G$
3      $S = \{(g, \texttt{rdf:type}, \texttt{pgo:PropertyGraph})\}$
4      foreach $n \in N$ do
5          $(\phi(n), \texttt{rdf:type}, \texttt{pgo:Node}) \in S$
6          foreach $l \in \lambda(n)$) do
7              $(\phi(n), \texttt{pgo:label}, l) \in S$
8          end
9          foreach $p \in \sigma(n)$ do
10             $(\phi(n), \texttt{pgo:hasNodeProperty}, \phi(p)) \in S$
11             $(\phi(p), \texttt{rdf:type}, \texttt{pgo:Property}) \in S$
12             if $\rho(p) = (k, v)$ then
13                 $(\phi(p), \texttt{pgo:key}, k) \in S$
14                 $(\phi(p), \texttt{pgo:value}, \tau(v)) \in S$
15             end
16         end
17     end
18     foreach $e \in E$ do
19         $(\phi(g), \texttt{pgo:hasEdge}, \phi(e)) \in S$
20         $(\phi(e), \texttt{rdf:type}, \texttt{pgo:Edge}) \in S$
21         if $\delta(e) = (n_1, n_2)$ then
22             $(\phi(e), \texttt{pgo:startNode}, \phi(n_1)) \in S$
23             $(\phi(e), \texttt{pgo:endNode}, \phi(n_2)) \in S$
24         end
25         foreach $l \in \lambda(e)$) do
26             $(\phi(e), \texttt{pgo:label}, l) \in S$
27         end
28         foreach $p \in \sigma(e)$ do
29             $(\phi(e), \texttt{pgo:hasEdgeProperty}, \phi(p)) \in S$
30             $(\phi(p), \texttt{rdf:type}, \texttt{pgo:Property}) \in S$
31             if $\rho(p) = (k, v)$ then
32                 $(\phi(p), \texttt{pgo:key}, k) \in S$
33                 $(\phi(p), \texttt{pgo:value}, \tau(v)) \in S$
34             end
35         end
36     end
37     return $S$
38 end
```

C. EXAMPLE OF TRANSFORMATION

Let G be the property graph presented in Figure 1. The set of RDF triples (presented in Figure 3) resulting from $\mathcal{M}(G)$ will be the following:

```
S = {
(ex:g, rdf:type, pgo:PropertyGraph),
(ex:n1, rdf:type, pgo:Node),
(ex:n1, pgo:label, "Person"),
(ex:n1, pgo:hasNodeProperty, ex:p1),
(ex:p1, rdf:type, pgo:Property),
(ex:p1, pgo:key, "Name"),
(ex:p1, pgo:value, "S. Abiteboul"),
(ex:n1, pgo:hasNodeProperty, ex:p2),
(ex:p2, rdf:type, pgo:Property),
(ex:p2, pgo:key, "Affiliation"),
(ex:p2, pgo:value, "Stanford"),
(ex:n2, rdf:type, pgo:Node),
(ex:n2, pgo:label, "Entry"),
(ex:n2, pgo:label, "Inproceedings"),
(ex:n2, pgo:hasNodeProperty, ex:p3),
(ex:p3, rdf:type, pgo:Property),
(ex:p3, pgo:key, "Title"),
(ex:p3, pgo:value, "Queries and
```

[6]Note that the function ϕ can be adapted to any application domain.

```
computation on the Web"),
(ex:n2, pgo:hasNodeProperty, ex:p4),
(ex:p4, rdf:type, pgo:Property),
(ex:p4, pgo:key, "Booktitle"),
(ex:p4, pgo:value, "ICDT"),
(ex:n2, pgo:hasNodeProperty, ex:p5),
(ex:p5, rdf:type, pgo:Property),
(ex:p5, pgo:key, "Year"),
(ex:p5, pgo:value, "1997"),
(ex:n2, pgo:hasNodeProperty, ex:p6),
(ex:p6, rdf:type, pgo:Property),
(ex:p6, pgo:key, "Keyword"),
(ex:p6, pgo:value, "Datalog"),
(ex:n2, pgo:hasNodeProperty, ex:p7),
(ex:p7, rdf:type, pgo:Property),
(ex:p7, pgo:key, "Keyword"),
(ex:p7, pgo:value, "Web"),
(ex:n3, rdf:type, pgo:Node),
(ex:n3, pgo:label, "Person"),
(ex:n3, pgo:hasNodeProperty, ex:p8),
(ex:p8, rdf:type, pgo:Property),
(ex:p8, pgo:key, "Name"),
(ex:p8, pgo:value, "V. Vianu"),
(ex:n3, pgo:hasNodeProperty, ex:p9),
(ex:p9, rdf:type, pgo:Property),
(ex:p9, pgo:key, "{\AA}ffiliation"),
(ex:p9, pgo:value, "UCSD"),
(ex:g, pgo:hasEdge, ex:e1),
(ex:e1, rdf:type, pgo:Edge),
(ex:e1, pgo:startNode, ex:n2),
(ex:e1, pgo:endNode, ex:n1),
(ex:e1, pgo:label, "Author"),
(ex:e1, pgo:hasEdgeProperty, ex:p10),
(ex:p10, rdf:type, pgo:Property),
(ex:p10, pgo:key, "Order"),
(ex:p10, pgo:value, "1"),
(ex:g, pgo:hasEdge, ex:e2),
(ex:e2, rdf:type, pgo:Edge),
(ex:e2, pgo:startNode, ex:n2),
(ex:e2, pgo:endNode, ex:n3),
(ex:e2, pgo:label, "Author"),
(ex:e2, pgo:hasEdgeProperty, ex:p11),
(ex:p11, rdf:type, pgo:Property),
(ex:p11, pgo:key, "Order"),
(ex:p11, pgo:value, "2")
}
```

D. PROPERTIES OF THE TRANSFORMATION

In this section, we analyze three properties of the data mapping $\mathcal{M}$: complexity, data preservation, and monotonicity[7].

1) COMPLEXITY OF $\mathcal{M}$

Let us analyze Algorithm 1. We can observe two recursive steps. The first step (lines 4 to 17) executes a sequential revision of all the nodes in the graph, and includes recursive procedures to extract node labels and node properties.

[7]The notions of data preservation and monotonicity are based on the work of Sequeda *et al.* [36].

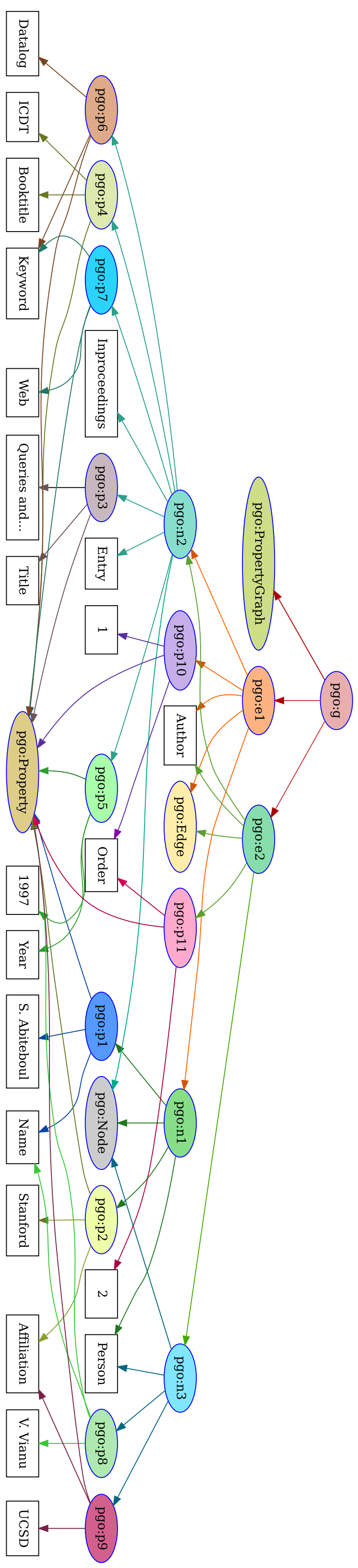

FIGURE 3. Graphical representation of an RDF graph.

Assuming that, the number of labels plus the number of properties is a constant c in average, then the complexity of this step is $\mathcal{O}(|N| \times c)$.

The second step (lines 18 to 36) performs a sequential revision of all the edges in the graph. It is very similar to the first step, but including instructions to extract the start and the end node of the processed edge. In this case, the complexity is $\mathcal{O}(|E| \times c)$ where c is the average number of labels and properties. Hence, the complexity of Algorithm 1 will be $\mathcal{O}(c(|N|+|E|))$, i.e. it is linear with respect to the elements in the property graph (i.e. nodes, edges, labels, and properties).

Additionally, we can analyze the size of the output RDF graph. Considering that we are describing the property graph, the number of elements in the RDF graph will be bigger than the elements in the original property graph. Specifically, each node requires 1 triple, each edge requires 4 triples, each label requires 1 triple, and each property requires 4 triples.

Given the above condition, we can also suppose than the disk space required to store the output RDF file will be bigger than the original property graph. However, it depends on the data format used. For example, Turtle allows a compact encoding of RDF graphs by using abbreviated IRIs.

2) DATA PRESERVATION OF $\mathcal{M}$

Let M_1 and M_2 be two database models. A data mapping $\mathcal{M}$ from M_1 to M_2 is data preserving if there is a computable data mapping $\mathcal{M}^{-1}$ from M_2 to M_1 such that for every database instance D in M_1, it applies that $D = \mathcal{M}^{-1}(\mathcal{M}(D))$.

Data preservation indicates that, for some data mapping $\mathcal{M}$, there exists an "inverse" data mapping $\mathcal{M}^{-1}$ that allows recovering recover a database instance transformed with $\mathcal{M}$. Data preservation is a fundamental property because it guarantees that the mapping does not lose data.

In order to prove that the data mapping $\mathcal{M}$ is data preserving, we need to find an inverse data mapping $\mathcal{M}^{-1}$, such that for any property graph G, it applies that $G = \mathcal{M}^{-1}(\mathcal{M}(G))$.

The inverse data mapping $\mathcal{M}^{-1}$ is defined by the procedure shown in Algorithm 2. This algorithm is divided in two steps. The first step (lines 2 to 14) iterates over the resources of type `pgo:Node`, and obtains the RDF triples describing labels and properties. The second step (lines 15 to 31) is very similar to the first step, but applied to resources of type `pgo:Edge`. In addition to the instructions to obtain labels and properties for edges, the second step obtains the start node and the end node for each edge. We can observe that both steps include instructions to re-construct all the elements (nodes, edges, properties and labels) of the original property graph. Hence, we can conclude that the data mapping $\mathcal{M}$ is data preserving.

3) MONOTONICITY OF $\mathcal{M}$

Given two database instances, I_1 and I_2 over a database model M_1, we say that I_1 is contained in I_2, denoted $I_1 \subseteq I_2$, if for every element e in I_1 it applies the e also exists in I_2. A data mapping $\mathcal{M}$ from data model M_1 to data model M_2 is considered monotone if for any such pair of instances in M_1, it applies that $\mathcal{M}(I_1) \subseteq \mathcal{M}(I_2)$.

Algorithm 2: $G = \mathcal{M}^{-1}(S)$

```
Input: A set of RDF triples $S$.
Output: A property graph $G = (N, E, P, \delta, \lambda, \sigma, \rho)$.
1  begin
2      foreach $(r_1, rdf{:}type, pgo{:}Node) \in S$ do
3          $N = N \cup \{n\}$
4          foreach $(r_1, pgo{:}label, l) \in S$ do
5              $\lambda(n) = \lambda(n) \cup \{l\}$
6          end
7          foreach $(r_1, pgo{:}hasNodeProperty, r_2) \in S$
            do
8              $P = P \cup \{p\}$
9              $\sigma(n) = \sigma(n) \cup \{p\}$
10             if $(r_2, pgo{:}key, k) \in S$ and
                $(r_2, pgo{:}value, v) \in S$ then
11                 $\rho(p) = (k, v)$
12             end
13         end
14     end
15     foreach $(r_3, rdf{:}type, pgo{:}Edge) \in S$ do
16         $E = E \cup \{e\}$
17         if $(r_3, pgo{:}startNode, r_4) \in S$ and
            $(r_3, pgo{:}endNode, r_5) \in S$ then
18             $\delta(e) = (n_1, n_2)$ such that $n_1$ corresponds to
                $r_4$ and $n_2$ correspond to $r_5$
19         end
20         foreach $(r_3, pgo{:}label, l) \in S$ do
21             $\lambda(e) = \lambda(e) \cup \{l\}$
22         end
23         foreach $(r_3, pgo{:}hasEdgeProperty, r_4) \in S$
            do
24             $P = P \cup \{p\}$
25             $\sigma(e) = \sigma(e) \cup \{p\}$
26             if $(r_3, pgo{:}key, k) \in S$ and
                $(r_3, pgo{:}value, v) \in S$ then
27                 $\rho(p) = (k, v)$
28             end
29         end
30     end
31     return $S$
32 end
```

Monotonicity indicates that a data mapping is consistent with respect to the creation of new input data, i.e. the output data increases as the input data increases. It is a desirable property because it would avoid recalculating the mapping for the entire database after new data is inserted.

Given two property graphs $G_1 = (N_1, E_1, P_1, \delta_1, \lambda_1, \sigma_1, \rho_1)$ and $G_2 = (N_2, E_2, P_2, \delta_2, \lambda_2, \sigma_2, \rho_2)$, we say that G_1 is a subgraph of G_2, denoted $G_1 \subseteq G_2$, if it applies that $N_1 \subseteq N_2$, $E_1 \subseteq E_2$, $P_1 \subseteq P_2$, $\delta_1(e) = \delta_2(e)$ for every $e \in E_1$, $\lambda_1(x) = \lambda_2(x)$ for every $x \in N_1 \cup E_1$, $\sigma_1(x) = \sigma_2(x)$ for every $x \in N_1 \cup E_1$, and $\rho_1(p) = \rho_2(p)$ for every $p \in P_1$.

In order to prove that the data mapping $\mathcal{M}$ is monotone, we need to verify that for any pair of property graphs G_1 and G_2, such that $G_1 \subseteq G_2$, it applies that $\mathcal{M}(G_1) \subseteq \mathcal{M}(G_2)$, i.e. the set of RDF triples returned by $\mathcal{M}(G_1)$ is a subset of the set of RDF triples returned by $\mathcal{M}(G_2)$.

Note that Algorithm 1 defines a sequential procedure where, for each element in the graph (node, edge and property), the algorithm generates an specific number of triples for the output RDF graph. There are not special conditions that could affect the number of triples generated for each element. It implies that the more elements the input property graph has, the more elements the procedure creates. This condition reflects the monotonic behaviour of the data mapping $\mathcal{M}$.

V. EVALUATION

In this section, we present an experimental evaluation of our implementation. Specifically, we evaluate the efficiency to transform different property graph datasets (Subsection V-A). Additionally, we present case studies and show how our proposal can be extended with additional semantics (Subsection V-B).

The following hypotheses are listed, where efficiency (in terms of storing) is related to the size of the output file, and extensibility refers to the ability to enrich data:

1) Regardless of serialization, the transformed graphs are stored efficiently.
2) Despite the lack of semantic descriptions before transformation, the data can be extended with appropriate vocabularies and ontologies.

All the input and output files of our experimental evaluation are available in [37]. All experiments were executed on an Intel Core i7-4770K CPU @ 3.50GHz (4 cores, 8 threads), 16GB of RAM (clock speed: 1600 MHz), and an HDD with reading speed rated at 160 MB/sec (we test it in `hdparm -t`). We used Linux Mint 19.1 Tessa (kernel version 4.15.0).

A. EFFICIENCY OF THE TRANSFORMATION

The transformation method presented in this article was implemented in Python, and the source code is available in GitHub (https://github.com/domel/graphConv) [38]. The current version supports the YARS-PG and GraphML data formats for the input property graph, and different RDF data formats for the output RDF graph (e.g. Turtle, JSON-LD, RDF/XML).

The efficiency of our implementation was evaluated by using four property graphs, whose number of nodes and edges is presented in Table 4. The first dataset (DS1) mainly concerns persons, their activities, and relations with other people. This dataset was crawled from the Web. The next dataset (DS2) describes a co-authorship network of scientists working on network theory and experiment, which is presented in [39]. The third dataset concerns a graph of disorders and disease genes linked by known disorder-gene associations, which is presented in [40]. The last dataset (DS4) consists into a social network produced with data generator of the LDBC Social Network Benchmark (SNB) [41].

TABLE 4. Description of datasets.

Cardinality	DS1	DS2	DS3	DS4
#nodes	9 511	1 589	1 419	327 588
#edges	11 845	2 742	3 926	1 477 965
#triples	128 136	25 986	43 422	10 337 997

For each property graph, we generated a file in YARS-PG and a file in GraphML. Table 4 shows the number of RDF triples obtained after transforming the input property graphs.

Table 5 shows the time required to transform each property graph file. The first part of the table presents transformation times from YARS-PG to RDF serializations (Turtle with labelled blank nodes, Turtle with nested blank nodes, JSON-LD, and RDF/XML). The second part of the table shows transformation times from GraphML to other RDF serializations. The generation times show that the transformations between these two models are very fast.

Table 6 shows file sizes before and after transformation. The first part of the table presents data formats for property graphs: YARS-PG and GraphML. The second part of the table shows RDF serializations, such as Turtle, JSON-LD, and RDF/XML. It is worth mentioning that Turtle syntax allows blank nodes to be written in the two forms, labelled and nested (unlabeled), and this has a significant effect on the file size.

Moreover, we decided to test our solution for compression that can be streaming to improve transfer speed, capacity, and bandwidth utilization. We use Gzip/Deflate[8] algorithm because it is widely supported by various Internet protocols. We define compression ratio cr as $\frac{compression\_size}{textual\_size}$, where $textual\_size$ is the size of a current serialization file. In Table 7 we present these compression ratios.

Table 6 shows a comparison of the data formats used in our experiments. We can see that property graph data formats allow to write graph data in a compact form. This is mainly because more RDF triples are needed to map nodes and edges. However, formats that occupy more space compress better (Table 7).

B. CASE STUDIES

The property graph model is not semantic (unlike RDF), i.e. it does not describe the meaning of the data. Dealing with semantics in such databases is possible but not supported natively. Therefore, the main inconvenience posed by our automatic mapping is the lack of support for vocabularies and/or ontologies.

An optional step after transformation is to enrich the graph with vocabularies and ontologies that will better describe the transformed data. This step is not necessary for the transformation but it is useful because it eliminates the disadvantages of property graphs (lack of support for vocabularies

[8]https://tools.ietf.org/html/rfc1952

TABLE 5. Generation times in seconds.

Input and output	DS1	DS2	DS3	DS4
YARS-PG → Turtle with labelled blank nodes [seconds]	1.317	0.124	0.168	33.231
YARS-PG → Turtle with nested blank nodes [seconds]	0.847	0.066	0.081	25.362
YARS-PG → JSON-LD [seconds]	1.101	0.083	0.112	29.968
YARS-PG → RDF/XML [seconds]	1.561	0.152	0.202	40.152
GraphML → Turtle with labelled blank nodes [seconds]	1.461	0.136	0.183	35.454
GraphML → Turtle with nested blank nodes [seconds]	0.941	0.072	0.092	28.908
GraphML → JSON-LD [seconds]	1.223	0.092	0.121	31.736
GraphML → RDF/XML [seconds]	1.751	0.161	0.223	44.014

TABLE 6. Amount of space occupied before (PG) and after (RDF) in bytes. The best results were marked with italics.

Data formats		DS1	DS2	DS3	DS4
PG	YARS-PG [bytes]	*1 882 718*	*105 297*	*132 029*	44 517 935
	GraphML [bytes]	2 875 036	265 192	338 842	140 894 498
	YARS-PG (deflate) [bytes]	*556 955*	*23 771*	*28 054*	*7 168 226*
	GraphML (deflate) [bytes]	627 827	33 127	42 420	12 735 294
RDF	Turtle with labelled blank nodes [bytes]	5 059 289	886 038	1 448 491	363 819 201
	Turtle with nested blank nodes [bytes]	*2 680 687*	*406 551*	*620 874*	*183 412 476*
	JSON-LD [bytes]	3 851 878	616 083	999 572	281 354 270
	RDF/XML [bytes]	6 970 412	704 497	1 178 567	546 411 111
	Turtle with labelled blank nodes (deflate) [bytes]	590 671	71 800	112 234	38 493 537
	Turtle with nested blank nodes (deflate) [bytes]	*417 954*	*41 935*	*61 018*	*23 990 442*
	JSON-LD (deflate) [bytes]	607 546	79 521	121 487	41 048 056
	RDF/XML (deflate) [bytes]	688 083	49 752	65 220	47 320 003

TABLE 7. Compression rates. The best results are marked with italics.

Compression ratio	DS1	DS2	DS3	DS4
cr_{yarspg}	29.58%	22.57%	21.25%	16.10%
$cr_{graphml}$	21.84%	12.49%	12.52%	9.04%
$cr^{labelled}_{turtle}$	11.67%	8.10%	7.75%	10.58%
cr^{nested}_{turtle}	15.59%	10.31%	9.83%	13.08%
cr_{jsonld}	15.77%	12.91%	12.15%	14.59%
cr_{rdfxml}	*9.87%*	*7.06%*	*5.53%*	*8.66%*

TABLE 8. Description of query features. ● – query uses the feature, ●* – query uses two triple patterns, ● – query uses three triple patterns.**

Query features	Q1	Q2	Q3
simple construct	●		
multiple constructs		●*	●**
multiple objects			●
matching literals		●	●
functions	●		
multiple vocabularies			●

and ontologies). It is worth remembering that enrichment, depending on the clauses written, may change the structure of graphs. This brings benefits, e.g. deletion of data that may no longer be needed, but also has disadvantages, e.g. it can cause that after the reverse transformation the graphs will not be the same.

Here we present a way to enrich our RDF with new vocabularies and ontologies. The user can create arbitrary queries, but we choose three that are representative (see Table 8). These queries uses SPARQL `CONSTRUCT` clause[9]. In this section, we propose three case studies to demonstrate the use of enriching semantics and its deployment according to [42] guidelines. The queries are executed in Apache Jena 3.12 and available on [37].

1) OBJECTIVES

We evaluate DS1, DS2, and DS3 in different conversions. The main hypothesis is that the conversion is possible without incurring into particular additional requirements that could change the proposed ontology.

2) RELATED STUDIES

Many academic papers include examples of using different approaches to convert RDF-to-RDF [43]–[45].

3) CASE 1 IMPLEMENTATION

The first query creates FOAF names from PGO. The query is for the DS1, where all crawled names are mapped to `foaf:name`. All RDF triples have IRI references generated as a subject using the `IRI()` function.

4) CASE 2 IMPLEMENTATION

The second query creates DBLP person instances and full names from PGO. The query is for the DS2, where the data is filtered so that it shows the authors of the publications.

5) CASE 3 IMPLEMENTATION

The third query creates DBpedia diseases with names and medicine subjects from PGO. The query is for the DS3, where names of disease are mapped to `skos:prefLabel` and medicine subjects are mapped to `dbo:eMedicineSubject`. The conversion uses two ontologies: DBO and SKOS.

[9]https://www.w3.org/TR/2013/REC-sparql11-query-20130321/

TABLE 9. Times of enriching RDF.

RDF data formats	Q1	Q2	Q3
Turtle (labbeled) [seconds]	0.224	0.284	0.599
Turtle (nested) [seconds]	0.253	0.304	0.603
JSON-LD [seconds]	0.389	0.343	0.706
RDF/XML [seconds]	0.233	0.293	0.730

TABLE 10. Transformation approaches.

Approaches	Format	Automation degree
RDF* [49]	PG	automatic
Das *et al.* [50]	PG	manual
NSMNTX [51]	PG	automatic
rdf2neo [52]	PG	semi-automatic
SRDS [53]	PG	automatic
YARS [54]	PG	automatic
*DM [16]	PG	automatic
Triplify [18]	RDB	manual
StdTrip [55]	RDB	semi-automatic
RDOTE [56]	RDB	manual
D2RQ [57]	RDB	automatic
AuReLi [58]	RDB	semi-automatic
OntoAccess [59]	RDB	manual
SquirrelRDF [60]	RDB	automatic and manual
R2RML [61]	RDB	manual
TARQL [62]	CSV	manual
CSV2RDF [63]	CSV	manual
Cruz *et al.* [64]	XML	automatic
Deursen *et al.* [65]	XML	manual
X2OWL [66]	XML	automatic
WEESA [67]	XML	automatic
JXML2OWL [68]	XML	automatic
XSPARQL [69]	XML	manual
Gloze [70]	XML	automatic
GRDDL [43]	XML	manual
SPARQL-Generate [45]	XML, JSON, CSV, HTML	manual
RML [44]	XML, JSON, CSV, RDB	manual

6) FINAL REMARKS

The three queries can be executed in different SPARQL tools. We test these queries in Sesame, Jena Fuseki, and Jena arq. We also consider different RDF serializations, e.g. Turtle, JSON-LD, RDF/XML. In Table 9 we present creation times. Turtle with labeled blank nodes had the best performance. All the results are the same, the generated RDF triples represent the same data. The output data can be represented in different RDF serializations. All queries are published in [37].

VI. RELATED WORK

In this section, we present the approaches for mapping property graph databases to RDF (Subsection VI-A), relational databases to RDF (Subsection VI-B) and the approaches for mapping XML to RDF (Subsection VI-C). A summary of approaches is shown in Table 10. It should be mentioned that some works have studied the problem of object-oriented programming languages to RDF (e.g. [13], [46]–[48]).

A. PROPERTY GRAPH DATABASES

There exist only a few proposals for the PG-to-RDF transformation, such as Das *et al.* [50] and Hartig [49], that mainly use RDF *reification* methods (including blank nodes) to convert nodes and edge properties in a property graph to RDF data. While [49] propose an in-direct mapping that requires converting to the RDF* model, [50] lacks a formal foundation.

In [49] Hartig *et al.* propose transformations between PGs and RDF. Unfortunately, this proposal uses an extension of RDF that is called RDF*. RDF* is not widely supported by the systems and cannot be implemented based on the system that has not been optimized for this proposal. The basis of the proposal is to extend the RDF data model with a notion of nested triples that is not standardized[10]. In contrast to the above approach, our proposal is based only on standards and widely supported languages.

Das *et al.* [50] present an approach based on a commercial relational database. That paper proposes three models: 1) extended reification based, 2) subproperty based, and 3) named graph based. Unfortunately, all presented model have some limitations. The first model requires reification, which does not have formal semantics and makes data difficult to query. The second model needs inference, which is time-consuming. The third model supports named graphs, which are not supported by all RDF systems.

Another group of proposals uses Neo4j databases [51], [52]. The main disadvantage of these solutions is a strong relationship with only one database.

Barrasa [51] proposes NSMNTX, a plugin that enables the use of RDF in Neo4j. This plugin allows the import and export of both schema and data. NSMNTX is basaed on neosemantics[11] and the previous author works[12]. The problem with this approach is that NSMNTX is not formally defined and the mappings do not satisfy the property of information preservation.

Brandizi *et al.* [52] propose rdf2neo, a tool that can be used to map any RDF schema to a desired PG schema. This hybrid architecture facilitates access to knowledge networks based on shared data models. This solution maintains a more complex infrastructure that works well in the paper use case, but not for more general applications.

In [53] Virgilio proposes an approach (SRDS) for converting an RDF data store to a graph database by exploiting the ontology and the constraints of the source.

Tomaszuk [54] proposes YARS serialization for transforming RDF data into PGs. This approach performs only a syntactic transformation between encoding schemes.

In [16] Angles *et al.* propose an approach that consists of three direct mappings, namely Simple Database Mapping (SDM), Generic Database Mapping (GDM), and Complete Database Mapping (CDM)[13]. This approach is well-formalized and the mappings GDM and CDM are satisfy the property of information preservation, i.e. there exist inverse mappings that allow recovering the original data and

[10]In the present year, an informal work is ongoing on standardizing RDF*.
[11]https://github.com/neo4j-labs/neosemantics/
[12]https://jbarrasa.com/2016/06/07/importing-rdf-data-into-neo4j/
[13]We put these three mappings in the table as *DM.

schema without the loss of information. Furthermore, all proposed mapping are semantics preserving, i.e. it indicates that the output of a database mapping is always a valid database. A general limitation of the three mappings is that they are not support reified RDF data and RDF inference rules.

B. RELATIONAL DATABASES AND TABULAR DATA

At the beginning, we focus on solutions [18], [55], [56] based on SQL as mapping representation. Triplify [18] is based on the mapping of HTTP requests onto database queries expressed in SQL queries, which are used to match subsets of the store contents and map them to classes and properties. The next approach is StdTrip [55], which proposes a structure-based framework using existing ontology alignment software. The approach finds ontology mappings between simple vocabulary that is generated from a database. RDOTE [56] also uses SQL for the specification of the data subset. In that proposal, the suitable SQL query is stored in a file. That approach transforms data residing in the database into RDF graph dump using classes and properties.

The next group of approaches [57], [58] uses D2RQ as mapping representation. D2RQ [57] supports both automatic and manual operation modes. In the first mode, RDFS vocabulary is created, by the reverse engineering methodologies, for the translation of foreign keys to properties. In the second mode, the contents of the database are exported to an RDF in accordance with mappings stored in RDF. Another D2RQ-based proposal is AuReLi [58], which uses several string similarity measures associating attribute names to existing vocabulary entities to complete the automation of the transformation of databases.

Another group of proposals [59], [60] use RDF. The first approach is OntoAccess [59], which is vocabulary-based write access to data. This approach consists relational database to RDF mapping language called R3M, which consists of an RDF format and algorithms for translating queries to SQL. The next proposal is SquirrelRDF [60], which extracts data from several databases and integrates that data into a business process. That proposal supports RDF views and allows for the execution of queries against it.

R2RML [61] is the best-known RDF language-based for expressing mappings from relational databases to RDF datasets because it is a W3C recommendation. R2RML is a language for specifying mappings from relational to RDF data. A mapping takes as input a logical table. In the next step a logical table is mapped to triples map that is a set of triples.

Another group are solutions that map only CSV to RDF [62], [63]. TARQL [62] converts CSV files using SPARQL 1.1 syntax. CSV2RDF [63] specification proposes the rules to be applied when converting tabular data into RDF. CSV data can be described with metadata annotations that explain its structure.

C. XML

At the beginning, we focus on solutions [64]–[68] that use existing vocabulary and/or ontology. This means that the XML data is transformed according to the mapped vocabularies. Cruz and Nicolle [64] propose basic mapping rules to specify the transformation rules on properties, which are defined in the XML Schema. Deursen *et al.* [65] propose the method for the transformation of XML data into RDF instances in an ontology-dependent way. X2OWL [66] is a tool that builds an OWL ontology from an XML data source and a set of mapping bridges. That proposal is based on an XML Schema that can be modeled using different styles to create the vocabulary structure. The next proposal is WEESA [67], which is an approach for Web engineering techniques and developing semantically tagged applications. Another tool is JXML2OWL [68]. It supports transformation from syntactic data sources in XML format to a common shared global model defined by vocabulary.

Another subgroup of approaches [43], [69], [70] supports mutual transformation. XSPARQL [69] is a query language based on SPARQL and XQuery for transformations from RDF into XML and back. It is built on top of XQuery in a syntactic and semantic view. Gloze [70] is another tool for bidirectional mapping between XML and RDF. It uses information available in the XML schema for describing how XML is mapped into RDF and back again. GRDDL [43] is a markup language that obtains RDF data from XML documents. It is represented in XSLT.

Moreover, there are some approaches [44], [45] that support XML and other formats at the same time. SPARQL-Generate [45] generates RDF from RDF datasets and a set of documents in arbitrary formats. It is designed as an extension of SPARQL 1.1. SPARQL-Generate supports XML, JSON, CSV, GeoJSON, HTML, and CBOR[14]. Another approach is RML (RDF Mapping language) [44]. It is a generic mapping language defined to express rules that map data in heterogeneous structures and serializations to the RDF data model. RML supports JSON, XML, CSV, and relational databases. RML is defined as a superset of R2RML (see Subsection VI-C). Furthermore, for a more comparison of the above mentioned approaches addressing data and query interoperability, we point the interested reader to [13].

VII. CONCLUSIONS

Interoperability is an important characteristic of databases whose interfaces are completely understood, to work with other systems. In this article, we have presented a method to transform a property graph database into an RDF triplestore. We propose an ontology mapping from PGs to RDF, which is simple and domain-independent. Our proposal was based mainly on OD101 and METHONTOLOGY. We also study fundamental properties in the context of our transformation. Moreover, our proposal allows for automatic transformations.

As part of future work, we will consider possibilities for manual and/or semi-manual transformations that will allow defining fragments which and how can be mapped. Furthermore, we will try to take into account a database schema.

[14]CBOR (Concise Binary Object Representation) is a binary representation of JSON.

REFERENCES

[1] M. Kejriwal, *Domain-Specific Knowledge Graph Construction*. Cham, Switzerland: Springer, 2019.

[2] O. Curé and G. Blin, *RDF Database Systems: Triples Storage and SPARQL Query Processing.*. San Mateo, CA, USA: Morgan Kaufmann, 2014.

[3] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space* (Synthesis Lectures on the Semantic Web: Theory and Technology), vol. 1, 1st ed. San Rafael, CA, USA: Morgan & Claypool, 2011, no. 1.

[4] D. Tomaszuk and D. Hyland-Wood, "RDF 1.1: Knowledge representation and data integration language for the Web," *Symmetry*, vol. 12, no. 1, p. 84, Jan. 2020.

[5] R. Cyganiak, D. Wood, and M. Lanthaler, *RDF 1.1 Concepts and Abstract Syntax*, document W3C, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[6] A. Seaborne and S. Harris, *SPARQL 1.1 Query Language*, document W3C, Mar. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[7] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases: New Opportunities for Connected Data*. Newton, MA, USA: O'Reilly Media, 2015.

[8] R. Angles, "The property graph database model," in *Proc. Alberto Mendelzon Int. Workshop Found. Data Manage. (AMW)*, 2018, pp. 1–8.

[9] R. Angles, H. Thakkar, and D. Tomaszuk, "RDF and property graphs interoperability: Status and issues," in *Proc. 13th Alberto Mendelzon Int. Workshop Found. Data Manage.*, Asunción, Paraguay, 2019, pp. 1–11.

[10] C. Parent and S. Spaccapietra, "Database integration: The key to data interoperability," in *Adv. Object-Oriented Data Modeling*, 2000, pp. 221–253.

[11] H. Thakkar, R. Angles, M. Rodriguez, S. Mallette, and J. Lehmann, "Let's build bridges, not walls: SPARQL Querying of TinkerPop graph databases with sparql-gremlin," in *Proc. IEEE 14th Int. Conf. Semantic Comput. (ICSC)*, Feb. 2020, pp. 408–415.

[12] H. Thakkar, D. Punjani, J. Lehmann, and S. Auer, "Two for one: Querying property graph databases using SPARQL via gremlinator," in *Proc. 1st ACM SIGMOD Joint Int. Workshop Graph Data Manage. Experiences Syst. (GRADES) Netw. Data Analytics (NDA)*, 2018, p. 12.

[13] M. N. Mami, D. Graux, H. Thakkar, S. Scerri, S. Auer, and J. Lehmann, "The Query translation landscape: A survey," *CoRR*, vol. abs/1910.03118, Dec. 2019.

[14] H. Thakkar, "Towards an open extensible framework for empirical benchmarking of data management solutions: LITMUS," in *Proc. Extended Semantic Web Conf. (ESWC)*, 2017, pp. 256–266.

[15] H. Thakkar, Y. Keswani, M. Dubey, J. Lehmann, and S. Auer, "Trying not to die benchmarking: Orchestrating RDF and graph data management solution benchmarks using LITMUS," in *Proc. 13th Int. Conf. Semantic Syst.*, 2017, pp. 120–127.

[16] R. Angles, H. Thakkar, and D. Tomaszuk, "Mapping rdf databases to property graph databases," *IEEE Access*, vol. 8, pp. 86091–86110, 2020.

[17] S. Ceri, L. Tanca, and R. Zicari, "Supporting interoperability between new database languages," in *Proc. Adv. Comput. Technol., Reliable Syst. Appl.*, 2020, pp. 273–281.

[18] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller, "Triplify: Light-weight Linked Data Publication from Relational Databases," in *Proc. 18th Int. Conf. World Wide Web*, New York, NY, USA, 2009, pp. 621–630.

[19] E. Prud'hommeaux and G. Carothers, *RDF 1.1 Turtle*, document W3C, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/

[20] G. Schreiber and F. Gandon, *RDF 1.1 XML Syntax*, document W3C, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[21] W. O. W. Group. (Dec. 2012). *OWL 2 Web Ontology Language Document Overview, W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-overview-20121211/

[22] J. Carroll, I. Herman, and P. F. Patel-Schneider, *OWL 2 Web Ontology Language RDF-Based Semantics*, document W3C Recommendation, Dec. 2012. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/

[23] C. M. Keet, *An Introduction to Ontology Engineering*. Town, South Africa: University of Cape Town, 2018.

[24] S. Staab and R. Studer, *Handbook Ontologies*, 2nd ed. Berlin, Germany: Springer, 2009.

[25] A. Gomez-Perez, M. Fernández-López, and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, E-Commerce and the Semantic Web*. London, U.K.: Springer, 2006.

[26] N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," Stanford Univ., Stanford, CA, USA, Tech. Rep. SMI-2001-0880, 2001.

[27] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "Methontology: From ontological art towards ontological engineering," in *Proc. Spring Symp. Ontological Eng.*, 1997, pp. 33–40.

[28] M. Grüninger and M. S. Fox, "Methodology for the design and evaluation of ontologies," in *Proc. Workshop Basic Ontological Issues Knowl. Sharing*, 1995, pp. 1–10.

[29] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *Knowl. Eng. Rev.*, vol. 11, no. 2, pp. 93–136, Jun. 1996.

[30] D. Tomaszuk, R. Angles, L. Szeremeta, K. Litman, and D. Cisterna, "Serialization for property graphs," in *Beyond Databases, Architectures and Structures. Paving the Road to Smart Data Processing and Analysis*. Cham, Switzerland: Springer, 2019, pp. 57–69.

[31] I. Horrocks, P. F. Patel-Schneider, T. Boley, G. Said, H. Benjamin, and M. Dean, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, document W3C, May 2004. [Online]. Available: https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[32] R. Guha and D. Brickley, *RDF schema 1.1*, document W3C, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[33] F. Baader, I. Horrocks, and U. Sattler, "Description logics," *Found. Artif. Intell.*, vol. 3, pp. 135–179, Dec. 2008.

[34] J. Hladik, "A tableau system for the description logic $\mathcal{SHIO}$," in *Proc. 2nd Int. Joint Conf. Automated Reasoning (IJCAR)*, 2004, pp. 21–25.

[35] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall, "Graphml progress report structural layer proposal," in *Proc. Int. Symp. Graph Drawing*. Berlin, Germany: Springer, 2001, pp. 501–512.

[36] J. F. Sequeda, M. Arenas, and D. P. Miranker, "On directly mapping relational databases to RDF and OWL," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 649–658.

[37] D. Tomaszuk. (Apr. 2020). *Property Graph Ontology (Experiments and Case Studies)*. [Online]. Available: https://dx.doi.org/10.6084/m9.figshare.11503110.v3

[38] D. Tomaszuk. (May 2020). *Domel/Graphconv: Converter*. [Online]. Available: https://dx.doi.org/10.6084/m9.figshare.12196122.v1

[39] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 74, no. 3, Sep. 2006, Art. no. 036104.

[40] K. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A. L. Barabási, "The human disease network," *Proc. Nat. Acad. Sci. USA*, vol. 104, no. 21, pp. 8685–8690, 2007.

[41] R. Angles, J. B. Antal, A. Averbuch, P. Boncz, O. Erling, A. Gubichev, V. Haprian, M. Kaufmann, J. L. L. Pey, N. Martínez, J. Marton, M. Paradies, M.-D. Pham, A. Prat-Pérez, M. Spasić, B. A. Steer, G. Szárnyas, and J. Waudby, "The IDBC social network benchmark," Linked Data Benchmark Council, London, U.K., Tech. Rep. CoRR abs/2001.02299, 2020.

[42] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, p. 131, 2008.

[43] D. Connolly, *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*. Cambridge, MA, USA: World Wide Web Consortium, Sep. 2007.

[44] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle, "RML: A generic language for integrated RDF mappings of heterogeneous data," in *Proc. 7th Workshop Linked Data Web*, Apr. 2014, p. 1184.

[45] M. Lefrancois, A. Zimmermann, and N. Bakerally, "A SPARQL extension for generating RDF from heterogeneous formats," in *Proc. Eur. Semantic Web Conf.*, 2017, pp. 35–50.

[46] Q. Tong, "Mapping object-oriented database models into RDF(S)," *IEEE Access*, vol. 6, pp. 47125–47130, 2018.

[47] S. Bagui and J. Bouressa, "Mapping RDF and RDF-Schema to the entity relationship model," *J. Emerg. Trends Comput. Inf. Sci.*, vol. 5, no. 12, pp. 953–961, 2014.

[48] Q. Tong, F. Zhang, and J. Cheng, "Construction of RDF(S) from UML class diagrams," *J. Comput. Inf. Technol.*, vol. 22, no. 4, pp. 237–250, 2014.

[49] O. Hartig and B. Thompson, "Foundations of an alternative approach to reification in RDF," *CoRR*, vol. abs/1406.3399, pp. 1–15, Oct. 2019.

[50] S. Das, J. Srinivasan, M. Perry, E. I. Chong, and J. Banerjee, "A tale of two graphs: Property graphs as RDF in oracle," in *Proc. EDBT*, 2014, pp. 762–773.

[51] J. Barrasa. (2020). *NSMNTX—Neo4j RDF & Semantics Toolkit*. [Online]. Available: https://neo4j.com/labs/nsmtx-rdf/

[52] M. Brandizi, A. Singh, and K. Hassani-Pak, "Getting the best of linked data and property graphs: Rdf2neo and the knetminer use case," in *Proc. Semantic Web Appl. Tools Health Care Life Sci. (SWAT4LS)*, 2018, pp. 1–4.

[53] R. D. Virgilio, "Smart RDF data storage in graph databases," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 872–881.

[54] D. Tomaszuk, "RDF data in property graph model," in *Proc. Res. Conf. Metadata Semantics Res. (MTSR)*, 2016, p. 4.

[55] P. E. Salas, K. K. Breitman, J. Viterbo F., and M. A. Casanova, "Interoperability by design using the StdTrip tool: An a Priori approach," in *Proc. 6th Int. Conf. Semantic Syst*, 2010, p. 43.

[56] K. N. Vavliakis, T. K. Grollios, and P. A. Mitkas, "RDOTE—Publishing relational databases into the semantic Web," *J. Syst. Softw.*, vol. 86, no. 1, pp. 89–99, Jan. 2013.

[57] C. Bizer and A. Seaborne, "D2RQ—Treating non-RDF databases as virtual RDF graphs," in *Proc. ISWC*, Nov. 2004, pp. 1–2.

[58] S. Polfliet and R. Ichise, "Automated mapping generation for converting databases into linked data," in *Proc. ISWC*, 2010, pp. 1–4.

[59] M. Hert, G. Reif, and H. C. Gall, "Updating relational data via SPARQL/update," in *Proc. 1st Int. Workshop Data Semantics*, 2010, p. 24.

[60] A. Seaborne, D. Steer, and S. Williams. (Oct. 2007). *SQL-RDF*. [Online]. Available: http://www.w3.org/2007/03/RdfRDB/papers/seaborne.html

[61] S. Das, R. Cyganiak, and S. Sundara, *R2RML: RDB to RDF Mapping Language*. Cambridge, MA, USA: World Wide Web Consortium, 2012.

[62] R. Cyganiak. (2015). *Tarql (SPARQL for tables): Turn CSV into RDF using SPARQL syntax*. [Online]. Available: https://tarql.github.io

[63] J. Tandy, I. Herman, and G. Kellogg, *Generating RDF from Tabular Data on the Web*. Cambridge, U.K.: World Wide Web Consortium, 2015.

[64] C. Cruz and C. Nicolle, "Ontology enrichment and automatic population from XML data," *ODBIS*, vol. 2008, pp. 17–20, Dec. 2008.

[65] D. V. Deursen, C. Poppe, G. Martens, E. Mannens, and R. V. D. Walle, "XML to RDF conversion: A generic approach," in *Proc. Int. Conf. Automated Solutions Cross Media Content Multi-Channel Distrib.*, Nov. 2008, pp. 138–144.

[66] R. Ghawi and N. Cullot, "Building ontologies from XML data sources," in *Proc. 20th Int. Workshop Database Expert Syst. Appl.*, 2009, pp. 480–484.

[67] G. Reif, M. Jazayeri, and H. C. Gall, "Towards semantic Web engineering: Weesa-mapping xml schema to ontologies," in *WWW Workshop Appl. Design, Develop. Implement. Issues Semantic Web*, 2004, p. 4.

[68] T. Rodrigues, P. Rosa, and J. Cardoso, "Moving from syntactic to semantic organizations using JXML2OWL," *Comput. Ind.*, vol. 59, no. 8, pp. 808–819, Oct. 2008.

[69] S. Bischof, N. Lopes, and A. Polleres, "Improve efficiency of mapping data between XML and RDF with XSPARQL," in *Proc. Int. Conf. Web Reasoning Rule Syst.*, 2011, pp. 232–237.

[70] S. Battle, "Gloze: XML to RDF and back again," in *Proc. Jena User Conf.*, 2006, pp. 1–11.

DOMINIK TOMASZUK received the M.Sc. degree from the Bialystok University of Technology, Poland, in 2008, and the Ph.D. degree from the Warsaw University of Technology, Poland, in 2014, both in computer science. He is currently a Researcher at the Institute of Informatics, University of Bialystok, Poland. His current research interests include semantic web, RDF, property graphs, NoSQL databases, and cheminformatics.

RENZO ANGLES received the bachelor's degree in systems engineering from the Universidad Católica de Santa María, Arequipa, Perú, and the Ph.D. degree in computer science from the Universidad de Chile, in 2009. In 2013, he held a postdoctoral research position at the Department of Computer Science, VU University Amsterdam, as part of his participation in the Linked Data Benchmark Council Project. He is currently an Assistant Professor at the Department of Computer Science, Universidad de Talca, Chile. He also participates as a Researcher at the Millennium Institute for Foundational Research on Data, Chile. His research interests include intersection of graph databases and semantic web. Specifically, he works in the theory and design of graph query languages and the interoperability between RDF and graph databases.

HARSH THAKKAR received the B.E. degree in computer engineering from the L.D. College of Engineering, Ahmedabad, India, in 2011, and the M.Tech. degree in computer science from NIT Surat, Surat, India, in 2013. He is currently pursuing the Ph.D. degree with the University of Bonn, Germany. He is also a Marie Skłodowska-Curie Alumni with the University of Bonn. He currently works as a Subject Matter Expert (SME) Consultant with OSTHUS GmbH, Aachen, Germany. He applies semantic technologies in the inter-disciplinary field of life sciences for solving real-world data-centric problems. His research interests include graph and RDF data management, benchmarking, graph query languages, and question answering.