g2gml Documentation

g2glab

Jan 04, 2021

Contents

CHAPTER 1

Quick Start

1.1 Sandbox

To understand how G2GML works quickly, please visit the sandbox.

- http://purl.org/g2gml

Some usage samples are provided. Notice that in order to use the sandbox effectively, knowledge about RDF and SPARQL are strongly recommended.

1.2 Command Line Usage

1.2.1 Installation

Set an alias to run a docker container:

```
$ alias g2g='docker run --rm -v $PWD:/work g2glab/g2g:0.3.5 g2g'
```

Check if it works:

```
$ g2g --help
```

1.2.2 Execution

Download example turtle file:

```
$ wget https://raw.githubusercontent.com/g2glab/g2g/master/examples/mini-05/mini-05.
↪ttl
```

`mini-05.ttl`

```
@prefix : <http://example.org/> .
:person1 a :Person .
:person2 a :Person .
[] a :Follow ;
   :follower :person1 ;
   :followed :person2 ;
   :since 2017 .
```

Download example g2g file:

```
$ wget https://raw.githubusercontent.com/g2glab/g2g/master/examples/mini-05/mini-05.
↪g2g
```

`mini-05.g2g`

```
PREFIX : <http://example.org/>

(p:person)
    ?p a :Person .

(p1:person)-[:follows {since:s}]->(p2:person)
    ?f :follower ?p1 ;
       :followed ?p2 ;
       :since ?s .
```

Run (mapping against RDF data file):

```
$ g2g mini-05.g2g mini-05.ttl
```

Check the output file:

```
$ more output/mini-05/mini-05.pg
```

`mini-05.pg`

```
"http://example.org/person1" :person
"http://example.org/person2" :person
"http://example.org/person1" -> "http://example.org/person2" :follows since:2017
```

For further details please refer to:

- G2GML: Description of the mapping language.
- G2G Mapper: Usage of the command line tool.

1.3 Related tools

- PG Tools: Tools to manage resulting property graph files.

CHAPTER 2

G2GML

The Graph To Graph Mapping Language (G2GML) is a language for mapping **RDF graphs** to **property graphs**.

2.1 Overview

- The mapping is described with a combination of **RDF graph patterns** and **property graph patterns**.
- RDF graph patterns are written in WHERE clause syntax of **SPARQL**, while the property graph patterns are written in MATCH clause syntax of **Cypher**.

`Structure of G2GML`

```
<prefixes>

<property graph patterns>           <-- Cypher MATCH clause syntax
    <semantic graph patterns>       <-- SPARQL WHERE clause syntax

...
```

2.2 Basic Examples

- RDF resource > PG node
- RDF datatype property > PG node property
- RDF object property > PG edge

2.2.1 RDF resource > PG node

Input: `mini-01.ttl`

```
@prefix : <http://example.org/> .
:person1 a :Person .
```

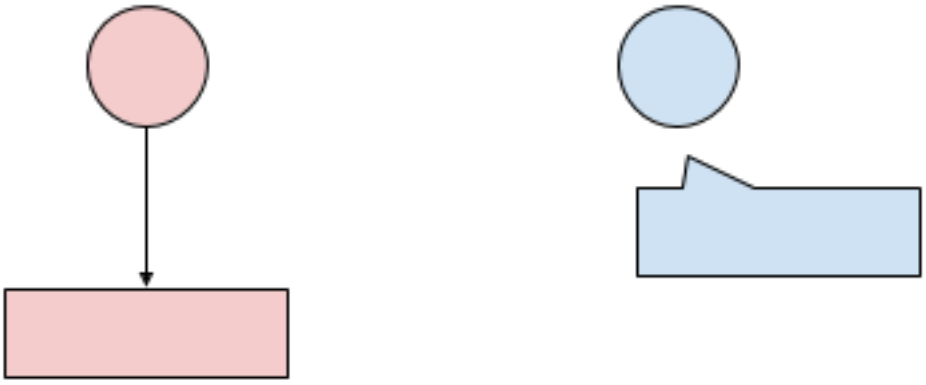
Mapping: `mini-01.g2g`

```
PREFIX : <http://example.org/>
(p:person)                          <-- PG node is defined
    ?p a :Person .
```

Output: `mini-01.pg`

```
"http://example.org/person1" :person
```

2.2.2 RDF datatype property > PG node property



Input: `mini-02.ttl`

```
@prefix : <http://example.org/> .
:person1 a :Person .
:person1 :age 30 .
```

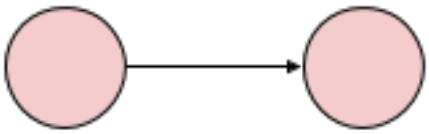
Mapping: `mini-02.g2g`

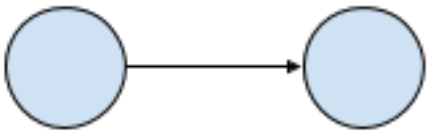
```
PREFIX : <http://example.org/>
(p:person {age:a})                  <-- PG node property is defined
    ?p a :Person .
    ?p :age ?a .
```

Output: `mini-02.pg`

```
"http://example.org/person1" :person age:30
```

2.2.3 RDF object property > PG edge





Input: `mini-03.ttl`

```
@prefix : <http://example.org/> .
:person1 a :Person .
:person2 a :Person .
:person1 :follows :person2 .
```

Mapping: `mini-03.g2g`

```
PREFIX : <http://example.org/>
(p:person)
    ?p a :Person .
(p1:person)-[:follows]->(p2:person)       <-- PG edge is defined
    ?p1 :follows ?p2 .
```

Output: `mini-03.pg`

```
"http://example.org/person1" :person
"http://example.org/person2" :person
"http://example.org/person1" -> "http://example.org/person2" :follows
```

2.2.4 RDF resource > PG edge

Input: `mini-04.ttl`

```
@prefix : <http://example.org/> .
:person1 a :Person .
:person2 a :Person .
[] a :Follow ;
   :follower :person1 ;
   :followed :person2 .
```

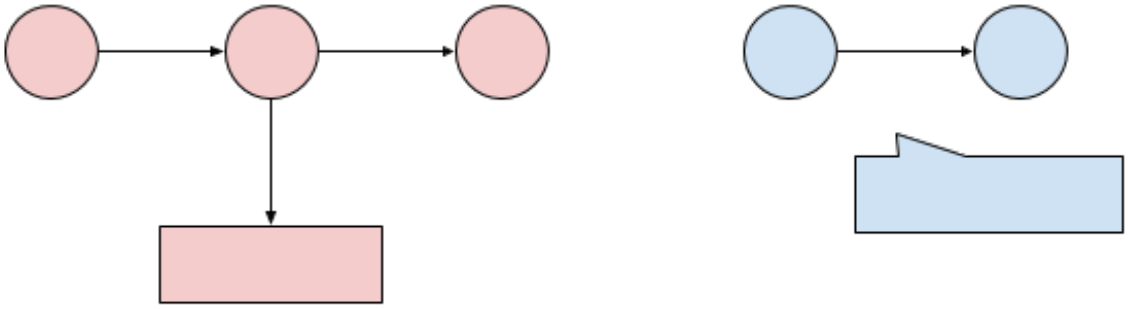
Mapping: `mini-04.g2g`

```
PREFIX : <http://example.org/>
(p:person)
    ?p a :Person .
(p1:person)-[:follows]->(p2:person)       <-- PG edge is defined
    ?f :follower ?p1 ;
       :followed ?p2 .
```

Output: `mini-04.pg`

```
"http://example.org/person1" :person
"http://example.org/person2" :person
"http://example.org/person1" "http://example.org/person2" :follows
```

2.2.5 RDF datatype property > PG edge property



Input: `mini-05.ttl`

```
@prefix : <http://example.org/> .
:person1 a :Person .
:person2 a :Person .
[] a :Follow ;
   :follower :person1 ;
   :followed :person2 ;
   :since 2017 .
```

Mapping: `mini-05.g2g`

```
PREFIX : <http://example.org/>
(p:person)
    ?p a :Person .
(p1:person)-[:follows {since:s}]->(p2:person)    <-- PG edge is defined
    ?f :follower ?p1 ;
       :followed ?p2 ;
       :since ?s .
```

Output: `mini-05.pg`

```
"http://example.org/person1" :person
"http://example.org/person2" :person
"http://example.org/person1" "http://example.org/person2" :follows since:2017
```

2.3 Actual Example

`musician.g2g`

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <http://schema.org/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpedia-prop: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

# Node mappings
(mus:musician {vis_label:nam, born:dat, hometown:twn, page_length:len})
    ?mus rdf:type foaf:Person, dbpedia-owl:MusicalArtist .
    ?mus rdfs:label ?nam .
    FILTER(lang(?nam) = "en") .
    OPTIONAL { ?mus dbpedia-prop:born ?dat }
    OPTIONAL { ?mus dbpedia-owl:hometown / rdfs:label ?twn. FILTER(lang(?twn) = "en").
↪ }
    OPTIONAL { ?mus dbpedia-owl:wikiPageLength ?len }

# Edge mappings
(mus1:musician)-[:same_group {label:nam, hometown:twn, page_length:len}]->
↪(mus2:musician)
    ?grp a schema:MusicGroup.
    { ?grp dbpedia-owl:bandMember ?mus1 , ?mus2. } UNION
    { ?grp dbpedia-owl:formerBandMember ?mus1 , ?mus2. }
    FILTER(?mus1 != ?mus2)
    OPTIONAL { ?grp rdfs:label ?nam. FILTER(lang(?nam) = "en")}
    OPTIONAL { ?grp dbpedia-owl:hometown / rdfs:label ?twn. FILTER(lang(?twn) = "en").
↪ }
```

(continues on next page)

(continued from previous page)

```
    OPTIONAL { ?grp dbpedia-owl:wikiPageLength ?len }

(mus1:musician)-[:influenced]->(mus2:musician)
    ?mus1 dbpedia-owl:influenced ?mus2 .
```

CHAPTER 3

G2G Mapper

3.1 Installation

If **Docker** is installed on your machine, run the following:

```
$ alias g2g='docker run --rm -v $PWD:/work g2glab/g2g:0.3.7 g2g'
$ g2g --version
0.3.7
```

Otherwise, install **Git** and **Node**, then run the following:

```
$ git clone -b v0.3.7 https://github.com/g2glab/g2g.git
$ cd g2g
$ npm install
$ npm link
$ g2g --version
0.3.7
```

If you use local file mode without Docker, install **Apache Jena ARQ** and make sure that `arq` command can be executed.

3.1.1 Testing installation

If you want to check whether your installation works correct, run the following:

(With Docker)

```
$ docker run --rm -v $PWD:/work g2glab/g2g:0.3.7 bash -c "cd /opt/g2g && npm test"
```

(Without Docker)

```
$ npm test
```

3.2 Usage

```
Usage:

   g2g [options] <g2gml_file> <data_source>

Options:

  -V, --version              shows the version number
  -f, --format [format]      format of results <rq|pg|pgx|neo|dot|aws|all (default:␣
↪pg)>
  -o, --output_dir [prefix]  output directory (default: output/<input_prefix>)
  -h, --help                 output usage information
```

3.3 Endpoint Mode

Download example g2g file:

```
$ wget https://raw.githubusercontent.com/g2glab/g2g/master/examples/musician/musician.
↪g2g
```

musician.g2g

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
...

# Node mappings
(mus:musician {vis_label:nam, born:dat, hometown:twn, page_length:len})
    ?mus rdf:type foaf:Person, dbpedia-owl:MusicalArtist .
...

# Edge mappings
(mus1:musician)-[:same_group {label:nam, hometown:twn, page_length:len}]->
↪(mus2:musician)
    ?grp a schema:MusicGroup ;
...
```

Run (mapping against SPARQL endpoint):

```
$ g2g musician.g2g http://dbpedia.org/sparql
```

Check the output file:

```
$ more output/musician/musician.pg
```

musician.pg

```
"http://dbpedia.org/resource/Martin_Glover"     :musician       vis_label:"Martin␣
↪Glover"
"http://dbpedia.org/resource/Per_Wiberg"        :musician       vis_label:"Per Wiberg
↪"  hometown:Stockholm
"http://dbpedia.org/resource/Tex_Perkins"       :musician       vis_label:"Tex Perkins
↪"
```

(continues on next page)

(continued from previous page)

```
"http://dbpedia.org/resource/Michelle_DaRosa"   :musician       vis_label:"Michelle␣
↪DaRosa"
"http://dbpedia.org/resource/Raúl_Sánchez_(musician)"   :musician       vis_label:
↪"Raúl Sánchez (musician)"     hometown:"Valencia, Spain"
...
"http://dbpedia.org/resource/Jin_Tielin"        ->        "http://dbpedia.org/
↪resource/Zu_Hai"        :influenced
"http://dbpedia.org/resource/George_Lam"        ->        "http://dbpedia.org/
↪resource/Eason_Chan"        :influenced
"http://dbpedia.org/resource/Aaron_Kwok"        ->        "http://dbpedia.org/
↪resource/Alien_Huang"        :influenced
"http://dbpedia.org/resource/Samuel_Hui"        ->        "http://dbpedia.org/
↪resource/Albert_Au"        :influenced
"http://dbpedia.org/resource/George_Lam"        ->        "http://dbpedia.org/
↪resource/Albert_Au"        :influenced
...
"http://dbpedia.org/resource/Ville_Valo"        ->      "http://dbpedia.org/resource/
↪Linde_Lindström"   :same_group     label:"HIM (Finnish band)"      hometown:Helsinki
"http://dbpedia.org/resource/Jerry_Donahue"     ->      "http://dbpedia.org/resource/
↪Sally_Barker"      :same_group     label:Fotheringay
"http://dbpedia.org/resource/Line_Horntveth"    ->      "http://dbpedia.org/resource/
↪Øystein_Moen"      :same_group     label:"Jaga Jazzist"    hometown:Norway
"http://dbpedia.org/resource/Adam_von_Buhler"   ->      "http://dbpedia.org/resource/
↪Kasson_Crooker"    :same_group     label:"Splashdown (band)"       hometown:"United␣
↪States"
"http://dbpedia.org/resource/Jeong_Jinwoon"     ->      "http://dbpedia.org/resource/
↪Lee_Chang-min_(singer)"    :same_group     label:"2AM (band)"      hometown:"South␣
↪Korea"
...
```

3.4 Local File Mode

Download example turtle file:

```
$ wget https://raw.githubusercontent.com/g2glab/g2g/master/examples/mini-05/mini-05.
↪ttl
```

mini-05.ttl

```
@prefix : <http://example.org/> .
:person1 a :Person .
:person2 a :Person .
[] a :Follow ;
   :follower :person1 ;
   :followed :person2 ;
   :since 2017 .
```

Download example g2g file:

```
$ wget https://raw.githubusercontent.com/g2glab/g2g/master/examples/mini-05/mini-05.
↪g2g
```

mini-05.g2g

```
PREFIX : <http://example.org/>

(p:person)
    ?p a :Person .

(p1:person)-[:follows {since:s}]->(p2:person)
    ?f :follower ?p1 ;
       :followed ?p2 ;
       :since ?s .
```

Run (mapping against RDF data file):

```
$ g2g mini-05.g2g mini-05.ttl
```

Check the output file:

```
$ more output/mini-05/mini-05.pg
```

`mini-05.pg`

```
"http://example.org/person1"        :person
"http://example.org/person2"        :person
"http://example.org/person1"        ->        "http://example.org/person2
↪"        :follows        since:2017
```

3.5 Internal Behaviour

1. Interprets G2GML and generates SPARQL queries to retrieve data
2. Issues SPARQL queries against public endpoints or given RDF data
3. Obtains the query results and transforms it into PG format
4. (optional) Translates PG data into specific formats for graph databases

3.6 Output Formats

3.6.1 PG

- Use `-f pg` or no `-f` option (default)
- Number of output files: 1 (sample.pg)
- PG tools can transform PG into other common formats.

3.6.2 JSON-PG

- Use `-f json`
- Number of output files: 1 (sample.json)

3.6.3 Neo4j

- Use `-f neo`
- Number of output files: 2 (sample.neo.nodes, sample.neo.edges)

3.6.4 Oracle Labs PGX

- Use `-f pgx`
- Number of output files: 3 (sample.pgx.nodes (opv), sample.pgx.edges (ope), sample.pgx.json (config))

3.6.5 Amazon Neptune

- Use `-f aws`
- Number of output files: 2 (sample.aws.nodes, sample.aws.edges)

3.6.6 Graphviz

- Use `-f dot`
- Number of output files: 1 (sample.dot)

CHAPTER 4

Mapping RDF Graphs to Property Graphs

https://arxiv.org/abs/1812.01801

4.1 Abstract

Increasing amounts of scientific and social data are published in the **Resource Description Framework (RDF)**. Although the RDF data can be queried using the SPARQL language, even the SPARQL-based operation has a limitation in implementing traversal or analytical algorithms. Recently, a variety of graph database implementations dedicated to analyses on the **property graph model** have emerged. However, the RDF model and the property graph model are not interoperable. Here, we developed a framework based on the **Graph to Graph Mapping Language (G2GML)** for mapping RDF graphs to property graphs to make the most of accumulated RDF data. Using this framework, graph data described in the RDF model can be converted to the property graph model and can be loaded to several graph database engines for further analysis. Future works include implementing and utilizing graph algorithms to make the most of the accumulated data in various analytical engines.

4.2 Introduction

Increasing amounts of scientific and social data are described as graphs. As a format of graph data, the **Resource Description Framework (RDF)** is widely used. Although RDF data can be queried using the SPARQL language in a flexible way, SPARQL is not dedicated to traversal of graphs and has a limitation in implementing graph analysis algorithms.

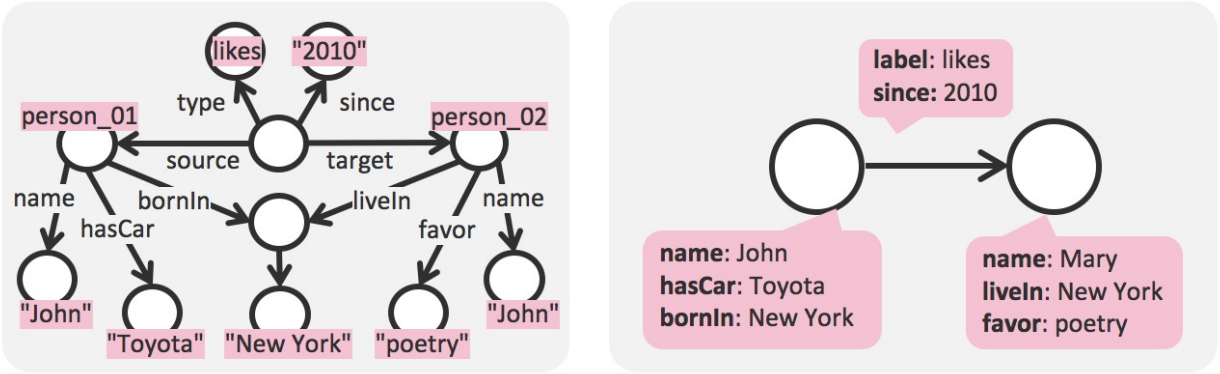
In the context of graph analysis, the **property graph model** is becoming popular; various graph database engines, including Neo4j, Oracle Labs PGX, and Amazon Neptune, adopt this model. These graph database engines support algorithms for traversal or analyzing graphs. However, currently not many datasets are consistently described in the property graph model, so the application of these powerful engines are limited.

Considering this situation, it is valuable to develop a method to transform RDF data into property graphs. However, the transformation is not straightforward due to the differences in the data model. In RDF graphs, all information is expressed as the triple (node-edge-node), whereas in property graphs, arbitrary information can be contained in each

of the nodes and edges as key-value form (**Figure 1**). Although previous works addressed this issue by formalizing transformations, users cannot define their specific mappings intended for each use case.

Here, we developed a framework based on the **Graph to Graph Mapping Language (G2GML)** for mapping RDF graphs to property graphs. Using this framework, accumulated graph data described in the RDF model can be converted to the property graph model and can be loaded to several graph database engines.

Figure 1. RDF Graph and property graph



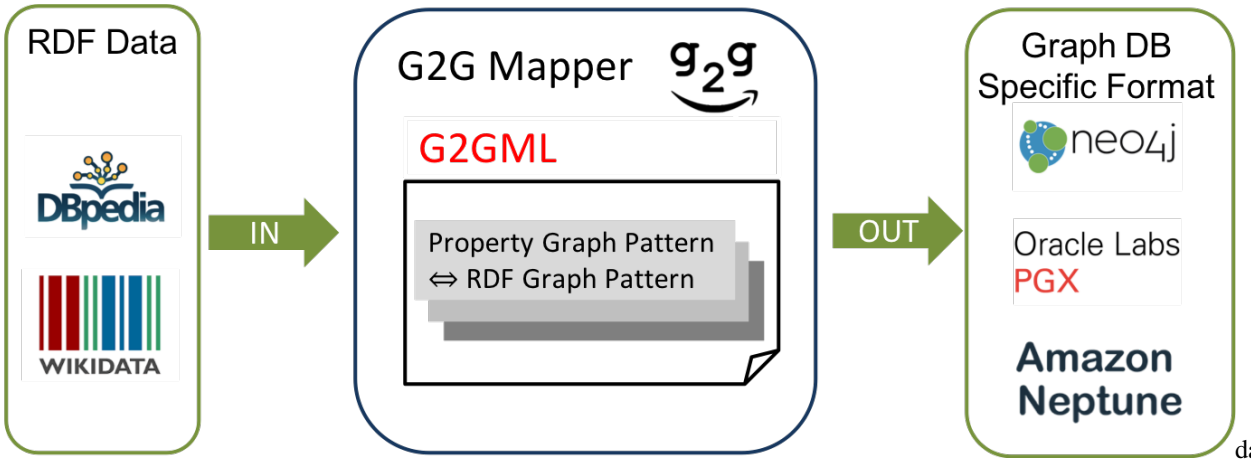
screen shot 2018-11-20 at 19 10 57

4.3 Method

Figure 2 shows the overview of proposed framework. In the proposed framework, users write mappings from RDF graphs to property graphs in G2GML. This mapping can be processed by an implementation called **G2G Mapper**, which is implemented by authors (available on https://github.com/g2gml). This tool retrieves RDF data from SPARQL endpoints and converts them to property graph data in several different formats specified by popular graph databases.

G2GML is a declarative language which consists of pairs of RDF graph patterns and property graph patterns. An intuitive meaning of a G2GML is a mapping between RDF subgraphs that matches the described patterns and described components of the property graph.

Figure 2. Overview of G2GML mapping



dataflow

Figure 3. Usage of G2G mapper

```
$ g2g [options] <g2gml_file> <data_source>
$ g2g -f pgx examples/musician/musician.g2g http://ja.dbpedia.org/sparql
```

4.4 Example

Figure 4 shows an example of G2GML mapping, which converts RDF data retrieved from DBpedia into property graph data. When we focus on relationships that one musician and another are in the same group, the information can be summarized into the property graph data as shown in this figure.

Figure 4. Mapping of RDF Data

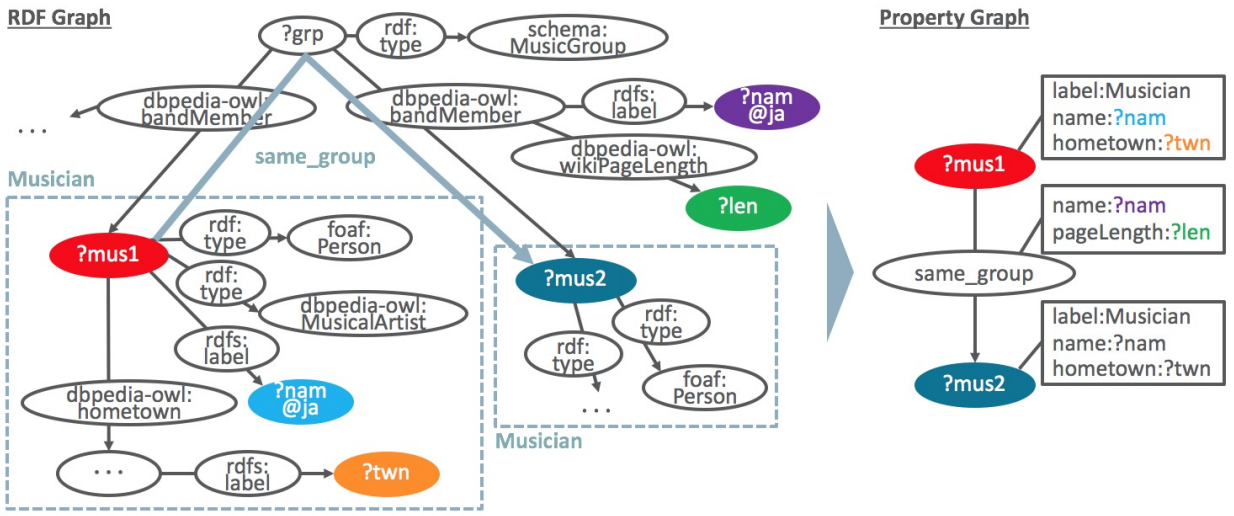


Figure 4

For this conversion, the actual G2GML is described as in **Figure 5**. It starts with URI prefixes used to write mappings, and then, each mapping consists of one unindented line of a property graph pattern and indented lines of an RDF graph pattern. A property graph pattern is written in a syntax like **Cypher** (the query language of Neo4j), whereas an RDF graph pattern is written as a pattern in **SPARQL**. Variables in each pattern are mapped by those names. This example contains one node mapping for Musician entity and one edge mapping for same group relationship only. In G2GML, edge mappings are defined based on the conditions of node mappings, which means that edges are generated in property graph iff both nodes' patterns and edges' patterns are matched in RDF graph. Also, **mus**, **nam**, **dat**, **twn** and **len** are used as variables to extract resources and literals from RDF graph. In the resulting property graph, resources can be mapped to nodes, while literals can be mapped to values of properties.

Figure 5. G2GML mapping definition

```
# Prefixes
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prop: <http://dbpedia.org/property/>
PREFIX schema: <http://schema.org/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
# Node mapping
(mus:musician {vis_label:nam, born:dat, hometown:twn})                    # PG Pattern
  ?mus rdf:type foaf:Person, dbpedia-owl:MusicalArtist .                  # RDF Pattern
  ?mus rdfs:label ?nam .
  OPTIONAL { ?mus prop:born ?dat }
```

(continues on next page)

(continued from previous page)

```
  OPTIONAL { ?mus dbpedia-owl:hometown / rdfs:label ?twn }
# Edge mapping
(mus1:musician)-[:same_group {label:nam, length:len}]->(mus2:musician)  # PG Pattern
  ?grp a schema:MusicGroup ;                                            # RDF Pattern
  dbpedia-owl:bandMember ?mus1 , ?mus2 .
  FILTER(?mus1 != ?mus2)
  OPTIONAL { ?grp rdfs:label ?nam. FILTER(lang(?nam) = "ja")}
  OPTIONAL { ?grp dbpedia-owl:wikiPageLength ?len }
```

Finally, **Figure 6** shows the SPARQL query to retrieve the pairs of musicians who are in the same group. After G2GML mapping above, we can load the generated property graph data into graph databases, such as Oracle Labs PGX, and the query can be written in PGQL (the query language of PGX).

Figure 6. SPARQL and PGQL

```
# SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <http://schema.org/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
SELECT DISTINCT
  ?nam1 ?nam2
WHERE {
  ?mus1 rdf:type foaf:Person , dbpedia-owl:MusicalArtist .
  ?mus2 rdf:type foaf:Person , dbpedia-owl:MusicalArtist .
  ?mus1 rdfs:label ?nam1 . FILTER(lang(?nam1) = "ja") .
  ?mus1 rdfs:label ?nam2 . FILTER(lang(?nam2) = "ja") .
  ?grp a schema:MusicGroup ;
        dbpedia-owl:bandMember ?mus1 , ?mus2 .
    FILTER(?mus1 != ?mus2)
}

# PGQL
SELECT DISTINCT m1.name, m2.name WHERE (m1)-[same_group]-(m2)
```

4.5 Conclusion

In this work, we defined **G2GML** for mapping RDF graphs to property graphs and implemented a converter based on the G2GML. We also showed an example usage of G2GML. Future works include further analysis of the converted graph data on the database engines adopting the property graph model.

4.6 For more information

- Project Home - https://github.com/g2glab
- G2G Sandbox - http://g2g.fun
- For Citation - https://arxiv.org/abs/1812.01801