Schema language for both RDF and LPGs

Dr. Ora Lassila

Principal Technologist, Neptune graph database team, AWS

Co-chair, W3C RDF-star working group

aws

Speaker introduction

Current Principal Technologist, Amazon Neptune graph database team
Co-chair, W3C RDF-star (RDF 1.2) working group

Past State Street, HERE, PegaSystems, Nokia Research Center, MIT/W3C, Carnegie Mellon University, Helsinki University of Technology, …

Co-author of the original RDF specification
Co-author of the seminal article on the Semantic Web
Grand Prize winner, USENIX Obfuscated C Code Contest

Education Ph.D AI & CS, Helsinki University of Technology
M.Sc CS & Telecom, Helsinki University of Technology

aws

RDF, LPGs, and the state of the “graph industry”

Graphs are increasingly popular

BUT

The “rift” between RDF and LPGs still hurts mainstream adoption

We need to better align the two graph models

- understand the fundamental differences better
- explore various ways how “graph interoperability” can be achieved
- evolve both models in ways that improves the alignment

Project “OneGraph” aims to do exactly this

RDF, LPGs, and the state of the “graph industry”

Graphs are increasingly popular

BUT

The “rift” between RDF and LPGs still hurts mainstream adoption

We need to better align the two graph models

- understand the fundamental differences better
- explore various ways how “graph interoperability” can be achieved
- evolve both models in ways that improves the alignment

Project “OneGraph” aims to do exactly this

aws

Background for today's presentation

Historically, LPGs have not had an actual schema language

- PG-Schema is an emerging proposal from LDBC

RDF, on the other hand, has too many schema languages 😱

- RDFS, OWL, SHACL… this can be confusing
- even R2RML is a language that at least implies a schema

Possible goals:

- one schema language for RDF (ontology + validation constraints)
- one schema language that covers both RDF and LPGs → **OneGraph**
- generally, understand related issues better

Background: PG-Schema

Defined by the LDBC LEX WG (SIGMOD 2023 Best Paper Award)

Simple, allows the definition of

- node types (i.e., “classes”), with property specifications
- edge types (again with property specifications)
- graph types (as collections of the above)

Types are defined as combinations of labels (+ properties)

- nodes in an LPG do not reference types, these are internal to PG-Schema
- labels could be interpreted as primitive base classes
- (alternatively, we could use some kind of DL-style classification)

PG-Schema: an example

```
CREATE GRAPH TYPE socialNetwork LOOSE {
  (person: thing & Person { OPTIONAL gender STRING })
  (thing: Thing { name STRING }
  (: person)-[friendsWith: Friend]->(: person)
}
```


aws

PG-Schema: an example


```
CREATE GRAPH TYPE socialNetwork LOOSE {
   (person: thing & Person { OPTIONAL gender STRING })
   (thing: Thing { name STRING }
   (: person)-[friendsWith: Friend]->(: person)
}
```

Graph definitions may be partial (“loose”)

Types may be partial (“open”)

Properties are allowed to be mandatory or optional

aws

PG-Schema vs. RDFS + SHACL

PG-Schema

Schema: separate from the graph

- no introspection
- no external identifiers

Types: combinations of labels associated with nodes

- no label hierarchy, only for internal types in the schema

RDFS + SHACL

Schema: part of the graph

- introspection, reasoning
- external identifiers are needed

Types: classes are referenced by instances (nodes)

- classes can establish type hierarchies

Both support descriptive and prescriptive use cases

aws

PG-Schema for RDF?

Think of this as a thought experiment

Immediate purpose:

- establish class-superclass hierarchies → ontologies
- validate RDF graphs

Game plan:

1. ditch labels (rather, “treat them as classes”)
2. make type identifiers external
3. translate the schema to a combination of SHACL and RDFS

Simple changes to syntax: namespaces, qualified names, URIs

No changes to RDF or SHACL semantics!

aws

PG-Schema for RDF: an example

```
PREFIX ex: <http://example.com/schema#>
CREATE GRAPH TYPE ex:socialNetwork LOOSE {
  (ex:Person : ex:Thing
    { OPTIONAL ex:gender xsd:string })
  (ex:Thing { ex:name xsd:string }
  (: ex:Person)-[ex:friendsWith]->(: ex:Person)
}
```

PG-Schema for RDF: an example


```
PREFIX ex: <http://example.com/schema#>
CREATE GRAPH TYPE ex:socialNetwork LOOSE {
  (ex:Person : ex:Thing
    { OPTIONAL ex:gender xsd:string })
  (ex:Thing { ex:name xsd:string }
  (: ex:Person)-[ex:friendsWith]->(: ex:Person)
}
```


aws

PG-Schema for RDF: a possible translation

Node type → class (with possible superclasses) + node shape

- types declared “OPEN” → SHACL shapes with `sh:closed false`

Node property → datatype property + sh:property constraint

- SHACL cardinality constraints can be used for optional/mandatory

Edge type → object property + sh:property constraint

- endpoint types become domains and ranges
- no real edge properties (at least until RDF-star has a say)

Graph type → ontology

aws

Translation to RDFS and SHACL

```
CREATE GRAPH TYPE ex:G LOOSE {
  (ex:A : ex:B
    { ex:p xsd:string })
}
```

SHACL

```
ex:A a sh:NodeShape ;
  sh:targetClass ex:A ;
  sh:closed true ;
  sh:property [
    sh:path ex:p ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:maxCount 1
  ] ;
  sh:ignoredProperties (
    rdf:type
  ) .
```

RDFS

```
ex:G a owl:Ontology .

ex:A a rdfs:Class ;
  rdfs:subClassOf ex:B .

ex:p a owl:DatatypeProperty ;
  rdfs:domain ex:A ;
  rdfs:range xsd:string .
```


aws

Translation to RDFS and SHACL

```
CREATE GRAPH TYPE ex:G LOOSE {
  (ex:A : ex:B
    { ex:p xsd:string })
}
```

SHACL

```
ex:A a sh:NodeShape ;
  sh:targetClass ex:A ;
  sh:closed true ;
  sh:property [
    sh:path ex:p ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:maxCount 1
  ] ;
  sh:ignoredProperties (
    rdf:type
  ) .
```

RDFS

```
ex:G a owl:Ontology .

ex:A a rdfs:Class ;
  rdfs:subClassOf ex:B .

ex:p a owl:DatatypeProperty ;
  rdfs:domain ex:A ;
  rdfs:range xsd:string .
```


aws

Translation to RDFS and SHACL

```
CREATE GRAPH TYPE ex:G LOOSE {
  (ex:A : ex:B
    { OPTIONAL ex:p xsd:string })
}
```

SHACL

```
ex:A a sh:NodeShape ;
  sh:targetClass ex:A ;
  sh:closed true ;
  sh:property [
    sh:path ex:p ;
    sh:datatype xsd:string ;
    sh:minCount 0 ;
    sh:maxCount 1
  ] ;
  sh:ignoredProperties (
    rdf:type
  ) .
```

RDFS

```
ex:G a owl:Ontology .

ex:A a rdfs:Class ;
  rdfs:subClassOf ex:B .

ex:p a owl:DatatypeProperty ;
  rdfs:domain ex:A ;
  rdfs:range xsd:string .
```


aws

Translation to RDFS and SHACL

```
CREATE GRAPH TYPE ex:G LOOSE {
  (ex:A : ex:B {} )
  (: ex:A)-[ex:q]->(: ex:A)
}
```

SHACL

```
ex:A a sh:NodeShape ;
  sh:targetClass ex:A ;
  sh:closed true ;
  sh:property [
    sh:path ex:q ;
    sh:class ex:A
  ] ;
  sh:ignoredProperties (
    rdf:type
  ) .
```

RDFS

```
ex:G a owl:Ontology .

ex:A a rdfs:Class ;
  rdfs:subClassOf ex:B .

ex:q a owl:ObjectProperty ;
  rdfs:domain ex:A ;
  rdfs:range ex:A .
```


aws

Translation to RDFS and SHACL

```
CREATE GRAPH TYPE ex:G LOOSE {
  (ex:A : ex:B { OPEN } )
  (: ex:A)-[ex:q]->(: ex:A)
}
```

R S

```
ex:G a owl:Ontology .

ex:A a rdfs:Class ;
  rdfs:subClassOf ex:B .

ex:q a owl:ObjectProperty ;
  rdfs:domain ex:A ;
  rdfs:range ex:A .
```

SHACL

```
ex:A a sh:NodeShape ;
  sh:targetClass ex:A ;
  sh:closed false ;
  sh:property [
    sh:path ex:q ;
    sh:class ex:A
  ] ;
  sh:ignoredProperties (
    rdf:type
  ) .
```


aws

Translating edge properties?

RDFS and SHACL have no vocabulary for establishing the use of edge properties in ontologies

- (we call them ”statements about statements”)
- W3C RDF-star WG will address this issue

Possible translation: SPARQL-based SHACL shape

- (using SPARQL-star)
- this could also be achieved using the classical RDF reification

Translation (notional, since RDF-star is not complete yet)

```
CREATE GRAPH TYPE ex:G LOOSE {
  (: ex:A)
  -[ex:q { ex:r xsd:int }]
  ->(: ex:A)}
```

SHACL

```
[] a sh:NodeShape ;
  sh:targetSubjectOf ex:q ;
  sh:sparql [
    a sh:SPARQLConstraint ;
    sh:ask """
      ASK {
        $target a ex:A .
        <<$target ex:q ?o>> {| ex:r ?value |} .
        FILTER (isLiteral(?value)
                && datatype(?value) = xsd:int)
      }
    """
  ] .
```

What is the domain of ex:r?

aws

Features of PG-Schema that cannot be covered in SHACL

Domain constraints for edge properties

“Strict” mode schemas (closed graph definitions)

- SHACL has no ability to identify parts of graphs not covered by any shape

Implementation (in Python)

Parser:

- direct translation of PG-Schema BNF to pyparsing
- productions generate AST elements
- productions for type identifiers, labels, property keys, and scalar types are replaced with Turtle BNF for URIs and qualified names

Translator:

- translator classes: simple dependency inversion for AST elements – they all have either a translate() method (for type definitions) or an evaluate() method (for identifiers, type names, etc.)
- translation is a single recursive traversal of the AST

Other benefits (possibly beyond RDF)?

LPGs: demonstrate how to use schema references and self-describing data

OneGraph: unification of RDF and LPG

- e.g., use openCypher over RDF (syntactically similar to PG-Schema)

Experiment with RDF:

- e.g., allow arbitrary strings as predicates (“class-local namespace”)

Future developments

Annotations, annotation properties

- minimally, rdfs:label

More general cardinality constraints

- RDF allows cardinalities > 1 for datatype properties
- PG-Schema does not (currently) have cardinality constraints for edges
- (SHACL is well equipped to handle all this)

Formal definition of the translation process (to show equivalence)

PG-Keys → SPARQL-based SHACL shapes?

Questions?

Contact:

- ora@amazon.com

Code available at:

- github.com/aws-samples/amazon-neptune-samples/tree/master/pg-schema-for-rdf

Thanks:

- Michael Schmidt (AWS)
- Holger Knublauch (TopQuadrant)
- Juan Sequeda (data.world)

aws