Semantic Web 14 (2023) 125–134
DOI 10.3233/SW-223273
IOS Press

The OneGraph vision: Challenges of breaking the graph model lock-in[1]

Ora Lassila [a,*], Michael Schmidt [a], Olaf Hartig [a,b], Brad Bebee [a], Dave Bechberger [a], Willem Broekema [a], Ankesh Khandelwal [a], Kelvin Lawrence [a], Carlos Manuel Lopez Enriquez [a], Ronak Sharda [a] and Bryan Thompson [a]

[a] *Amazon Neptune Team, Amazon Web Services, Seattle, WA, USA*
[b] *Department of Computer and Information Science (IDA), Linköping University, Sweden*

Editors: Pascal Hitzler, Kansas State University, USA; Krzysztof Janowicz, University of Vienna, Austria and University of California, Santa Barbara, USA

Abstract. Amazon Neptune is a graph database service that supports two graph models: W3C's Resource Description Framework (RDF) and Labeled Property Graphs (LPG). Customers choose one or the other model. This choice determines which data modeling features can be used and – perhaps more importantly – which query languages are available. The choice between the two technology stacks is difficult and time consuming. It requires consideration of data modeling aspects, query language features, their adequacy for current and future use cases, as well as developer knowledge. Even in cases where customers evaluate the pros and cons and make a conscious choice that fits their use case, over time we often see requirements from new use cases emerge that could be addressed more easily with a different data model or query language. It is therefore highly desirable that the choice of the query language can be made without consideration of what graph model is chosen and can be easily revised or complemented at a later point. To this end, we advocate and explore the idea of OneGraph ("1G" for short), a single, unified graph data model that embraces both RDF and LPGs. The goal of 1G is to achieve interoperability at both data level, by supporting the co-existence of RDF and LPG in the same database, as well as query level, by enabling queries and updates over the unified data model with a query language of choice. In this paper, we sketch our vision and investigate technical challenges towards a unification of the two graph data models.

Keywords: Interoperability, RDF, labeled property graphs, graph queries

1. Introduction

The Amazon Neptune graph database service supports both the data model of the Resource Description Framework (RDF) [5] with its query language SPARQL [10] and Labeled Property Graphs (LPG) [22,24][2] with the query languages Gremlin [23] and openCypher [9]. As of today, customers are not able to "cross-use" these technologies (e.g., a user cannot query RDF data using Gremlin). This limitation, as well as the differences in domain modeling approaches between the two graph models, cause confusion among prospective Neptune users. Especially, users who are new to graphs struggle frequently with the choice between the two models. We have even

[1] An earlier version of this paper, titled *"Graph? Yes! Which one? Help!"* was presented at the SCG2021 workshop at Semantics'21 in September 2021, and is available at https://arxiv.org/abs/2110.13348.

[*] Corresponding author. E-mail: ora@amazon.com.

[2] Note that we use the term "LPG" rather loosely, since there are differences even between different LPG implementations.

seen customers make unfortunate graph model choices with no easy way to reverse direction later, causing additional development effort such as introducing functionality into the application that could have been handled by the database. Examples along these lines include missing container/subgraph management support in today's LPG landscape where SPARQL provides built-in functionality via named graphs. Similarly, in use cases where path extraction and variable length path (VLP) queries[3] are of essence, LPG languages (e.g., Gremlin, openCypher) provide more flexibility than SPARQL.

Occasionally, we also receive requests for interoperability (i.e., cross-use) from experienced users who are proficient with RDF or LPG, or both, and from customers with use cases that are sometimes better served via RDF and at other times better served via LPG. The latter often happens when the initial requirements fall either into the data integration space (e.g., data alignment, master data management, data exchange) where RDF with standardized exchange formats and built-in ontology management capabilities has unique strengths, or in the analytics space (e.g., graph analytics, graph traversals) where the vertex/edge centric LPG abstraction with traversal-based query languages is often a more natural fit. In both scenarios, especially when graph technology is extended across departments, new use cases may suggest new and/or different modeling requirements. Concretely, it is likely that the users of a data integration use case will later ask for analytics and the users of an analytics use case will later ask for data integration.

Surveying the two data models in more detail, RDF offers a formal model that supports global identifiers (IRIs), well-defined graph merging, a natural way to break a graph into subgraphs, and query federation. This makes RDF graphs well-suited to use cases where multiple independent data sources are used (see, for example, the Linked Open Data project[4]). RDF also supports the reuse of schemas in the form of vocabularies for ontology definition as well as logical reasoning. Not surprisingly, we often see information architects prefer the features of the RDF model because of a good fit with data integration use cases. LPGs, on the other hand, are perceived as more agile because the data modeling effort is minimal. The flexibility to attach properties to either vertices or edges makes LPGs more natural in many use cases, whereas modeling alternatives in RDF that achieve the same goal are perceived as "workarounds." To overcome this limitation, the RDF-star/SPARQL-star approach proposes syntax and semantics extensions to attach and to query statements about edges [11]. Yet, for path queries, the expressivity of SPARQL and SPARQL-star is still limited. In contrast, the expressivity of LPG query languages already facilitates graph analytics and graph traversals with path extraction and VLP queries. Moreover, LPG languages are more in line with familiar programming languages and software developers find them more natural and better integrated with their programming language of choice. Also, developers coming from the SQL world appreciate that a vertex in an LPG seems like a row in a relational database. Unfortunately, many of these developers are then challenged when they need to expand a single graph to multiple data sources or when they have to deal with external data. Note that the choice of LPG can also happen when RDF is dismissed out of hand because it is viewed as complex and "academic." Regardless of the reasons, we believe that forcing customers to choose between graph models slows the adoption of graph technologies because it creates confusion and segmentation in the graph database space.

What if users could choose between different query languages, independent of what graph model they have decided to use? From interactions with Neptune customers we have noticed that there are often strong preferences for a particular query language, and there are also situations where one query language is simply better suited because of its particular features. For instance, expressiveness of graph traversals and path queries in Gremlin vs. `SERVICE` federation in SPARQL [20]. We argue that one's choice of a query language should not dictate that one has to forego benefits of a graph model said query language does not support.

Therefore, in this paper we examine the idea of *graph interoperability*. That is, removing the obstacles that prevent us from using SPARQL over LPGs, Gremlin or openCypher over RDF, etc. The goal is not merely to be able to cross-use query languages, but to do it in a manner where the user does not have to be careful about how the interoperability is achieved. In other words, we are interested in a data model with a unified semantics that generalizes the specifics of the RDF and LPG data models. In addition to flexibility in choosing the query language, the idea is that this data model would also allow graph users to combine and interlink data sets maintained in both

[3]Some RDF stores like Stardog and GraphDB offer propriertary constructs for path search queries. While this overcomes some SPARQL limitations, it harms portability as these constructs are not part of the SPARQL specification.

[4]https://lod-cloud.net/

RDF and LPG formats. More generally, the idea of providing a unified data model abstracts away the need for customers to choose a data format ahead of time, therefore removing a major obstacle to graph database adoption.

In this context it should be noted that less interesting is some kind of "qualified interoperability" where the cross-use of query languages would require one to understand the implementation of the underlying graph models. In other words, we want to stay away from, say, *implementing*[5] RDF using an LPG, mapping an RDF graph to an LPG [4,6,21], or describing LPGs in RDF [26], particularly if the chosen approach requires the users to be knowledgeable of multiple data models instead of a single one (due to limitations, information loss, etc.).

The discussion of LPGs in this paper is limited to the form of LPGs used by Gremlin and openCypher. Other graph query languages and implementations may come with additional challenges to consider. When discussing RDF, we consider features of RDF-star as well.

2. Preliminaries

As a basis for the discussion in this paper, we introduce a data-model-independent notation that has an interpretation in both the RDF and the LPG world and, thus, enables us to present examples using a data-model-independent syntax. Specifically, in our examples we represent graph data via so-called *statements* of the following form:

$$\texttt{src} \xrightarrow{\texttt{label}} \texttt{value} : sid$$

In the context of RDF, such statements shall be interpreted as triples with `src` as the subject, `label` as the predicate, and `value` as the object. For LPGs, the interpretation of a statement depends on the type of its `src` and `value` components. If `src` and `value` are both vertices, then the statement represents an edge with the given `label`; if `src` is a vertex and `value` is a literal (e.g., a string or numeric), the statement shall represent a property with key `label` and value `value`. Additionally, each statement in our notation is associated with a *statement identifier* (*SID*), *sid*. We will discuss use cases, intent, and characteristics of such identifiers throughout examples.

While we do believe that the concept of elements with *identity* is crucial to overcome existing gaps between RDF and LPGs, we emphasize that the statement notation with explicit SIDs used throughout the paper should not be understood as an attempt to propose this notation as a new graph data model. A complete formalization that bridges all the gaps between the triples-based view on graphs taken by RDF and the vertex/edge-centric paradigm underlying LPGs may well differ from, or go beyond, the (deliberately simple) statement notation chosen within this paper. Also, we would like to stress that this notation does not imply a necessity (neither does it suggest a preference) for a triples-based or a quads-based physical indexing scheme for graph data. The sole purpose of the statement notation is to provide a syntax that allows for a semi-formal presentation of examples, to illustrate characteristics of the two data models and the gaps between them.

As a concrete example of the notation, consider the following three statements:

$$\texttt{Alice} \xrightarrow{\texttt{knows}} \texttt{Bob} : sid_1 \qquad \texttt{Alice} \xrightarrow{\texttt{name}} \texttt{"Alice"} : sid_2 \qquad \texttt{Bob} \xrightarrow{\texttt{name}} \texttt{"Bob"} : sid_3$$

Informally, these statements represent a graph that captures a `knows` relationship between nodes, one for `Alice` (with name *"Alice"*) and one for `Bob` (with name *"Bob"*). The idea behind the SIDs is that they can now be used in either the `src` or the `value` position of statements, to refer to the statements that they identify (say, to make statements about these statements). For instance, to make a statement about the `knows` relationship between `Alice` and `Bob` we do so by adding a statement that references the statement through its SID sid_2 in the `src` position:

$$sid_1 \xrightarrow{\texttt{since}} 2020 : sid_4$$

[5] e.g., https://www.tigergraph.com/blogs/about-tigergraph/graph-gurus-episode-1-building-an-enterprise-knowledge-graph-from-rdf-data/.

Conceptually, we distinguish between *simple statements*, as statements that do not contain SIDs in the first three positions, and *meta-statements*, represented as statements with SIDs in the `src` or the `value` position.[6] In the previous example, the first three statements are simple ones and the last one is a meta-statement.

With the statement notation at hand, we can now discuss interoperability questions by discussing possible "views" of RDF, RDF-star, and LPG over a set of statements, where a view is defined by a (possible) mapping from our notation to these data models. We sketch this idea by providing possible interpretations of our toy graph, for RDF, RDF-star, and LPG, respectively:

```
# Plain RDF without reification (no support for edge properties)
:Alice :knows :Bob .
:Alice :name "Alice" .
:Bob :name "Bob" .

# RDF-star: edge properties = statements about quoted triples
:Alice :knows :Bob .
<<:Alice :knows :Bob>> :since 2020 .
:Alice :name "Alice" .
:Bob :name "Bob" .
```

With LPG, meta-statements about edges are viewed as edge properties, so our LPG interpretation may consist of the following vertices and edges:

v_1 with $id =$ `Alice` and properties $\{$`name => "Alice"`$\}$

v_2 with $id =$ `Bob` and properties $\{$`name => "Bob"`$\}$

e_1 from v_1 to v_2 with label `knows` and properties {`since` => 2020}

There exist, of course, other possible interpretations of the statement notation for each of the existing graph data formats, and we deliberately omit a detailed formalization in order to leave room for discussions around options and trade-offs throughout the paper. For instance, in the example above we chose the meta-statement with SID sid_4 to be "invisible" in the plain RDF representation. Alternatively, we could have represented this meta-statement using a set of triples that leverage the RDF reification vocabulary or any other custom scheme for RDF reification such as n-ary relations [8] or singleton properties [18].

3. Interoperability challenges and possible solutions

In this section, we discuss challenges to interoperability broadly, ranging from semantics to potential implementation issues. Note that the issue of *schema languages* is out of scope for this paper.

3.1. Challenge #1: Edge properties, multiple edge instances, and reification

One of the most fundamental (structural) differences perceived between RDF and LPGs is that RDF does not offer built-in support for edge properties (except as "statements about statements" via reification, which is generally understood to be cumbersome and inefficient). We sketched this aspect already when talking about possible interpretations for edge properties in RDF in the context of the example in Section 2. From our experience, the convenience of having edge properties as a built-in construct is one of the core strengths of LPGs over RDF.

The RDF-star effort is adding edge properties to RDF, but not in a way that is fully aligned with LPGs. That is, in RDF-star, a quoted triple (s, p, o) – as used in the subject or object position of other triples – is understood to be determined uniquely by its three elements, s, p and o. As a consequence, there cannot exist multiple identical instances of a quoted triple, each with its own identity. As such, all statements involving said (s, p, o) are understood

[6] We do not see a strong use case for supporting SIDs in the `label` position.

to reference the *same* (quoted) triple. In contrast, the original RDF reification mechanism [16, Section 4.3] is more expressive: each statement model has its own identity. In LPGs, each edge is considered a unique object with its own identity, and it is perfectly possible to have two edges that are identical in terms of both their incident vertices and their label. The following example illustrates a case of such multiple edge instances using our statement notation:

$$\texttt{Alice} \xrightarrow{\texttt{knows}} \texttt{Bob} : sid_1 \qquad \texttt{Alice} \xrightarrow{\texttt{knows}} \texttt{Bob} : sid_2$$

Furthermore, assume there are four meta-statements about these two simple statements:

$$sid_1 \xrightarrow{\texttt{statedBy}} \texttt{NYTimes} : sid_3 \qquad sid_2 \xrightarrow{\texttt{statedBy}} \texttt{TheGuardian} : sid_4$$

$$sid_1 \xrightarrow{\texttt{since}} 2020 : sid_5 \qquad sid_2 \xrightarrow{\texttt{since}} 2021 : sid_6$$

Informally speaking, the four meta-statements about the simple statements say that Alice knows Bob (a) since 2020 as per the NY Times, and (b) since 2021 as per The Guardian. This example has a natural representation in LPGs (namely, two distinguished edges, each carrying the respective two edge properties), whereas there is no built-in mechanism in RDF-star to capture this distinction. This means that, with respect to edge properties (and some other issues notwithstanding), we can take an RDF(-star) graph and map it onto an LPG, but there are cases where the converse is not true, at least not without falling back on more complex modeling approaches (such as introducing intermediate nodes or using named graphs to attach custom statement identifiers).

Of course, one could easily argue that our interpretation of the above model is wrong, because there is no logical dependency between the $since$ and $statedBy$ statements, and that one should in fact add more "nesting", as follows (the $statedBy$ statements are now about the $since$ statements – we have retained the original SIDs for clarity):

$$sid_1 \xrightarrow{\texttt{since}} 2020 : sid_5 \qquad sid_2 \xrightarrow{\texttt{since}} 2021 : sid_6$$

$$sid_5 \xrightarrow{\texttt{statedBy}} \texttt{NYTimes} : sid_3 \qquad sid_6 \xrightarrow{\texttt{statedBy}} \texttt{TheGuardian} : sid_4$$

but this is not the typical pattern employed with LPGs. It also makes query-writing more difficult.

Prohibiting multiple edge instances would seriously hamper how LPGs are used. There is a clear use case behind the distinction of the two edges in the example above. So, it would make sense to capture this expressiveness in a potential 1G model for graph database interoperability. The question then becomes what an RDF-star view (in which the scenario cannot be expressed "naturally," with built-in mechanisms) over such data would look like. One option is to collapse multiple edge instances (in our example, the statements with sid_1 and sid_2) into a single edge, effectively exposing a single edge with four edge properties when looking at the given example from an RDF-star perspective. This may still allow for some useful SPARQL-star queries (e.g., a query such as "give me all simple statements that have been stated by `:NYTimes`"), but in the general case information would be lost. Conceptually, this could be seen as a "dimensional reduction" when mapping from the potential 1G model into RDF-star, which is accounting for the fact that RDF-star is not expressive enough to capture the distinction between these two edges.

There are other open questions related to reification, apart from the edge identity challenge discussed so far. Those include aspects such as multi-level reification (which, in contrast to the scenario above, can be expressed with RDF-star but not with LPGs), the ability to reference statements in the `value` position rather than the `src` position (which is also conceptually possible in RDF-star but not with LPGs), and questions such as whether the reification of a statement via its SID necessarily implies the presence of the underlying statement as a triple in RDF or not. While those may sound like technical details, we believe that overcoming these "reification gaps" is at the very heart of graph interoperability.[7]

[7] We have submitted some use case examples to W3C that we believe should be considered in the eventual RDF-star specification [14].

3.2. Challenge #2: Triples vs. graph abstraction

At the very basic level, RDF graphs are defined as sets of triples, whereas LPGs are defined as (optionally labeled) vertices with properties that can be connected via labeled edges. The statement notation used in this paper gives us a straightforward mapping to RDF triples: in the absence of reification scenarios discussed in the previous section (i.e., in cases without meta-statements that have SIDs in their `src` or `value` positions), there is a natural one-to-one mapping between the `src`, the `label`, and the `value` of a statement to the subject, predicate, and object positions of RDF triples. However, the mapping from statements to LPG elements is more challenging. For instance, certain flavors of LPGs allow for "stand-alone" vertices without labels and properties, i.e., a vertex itself that neither carries any information nor is connected to any other vertex.[8] In fact, such stand-alone vertices can even appear in query results of LPG query languages – e.g., in Gremlin via `g.V()` and in openCypher via `MATCH (n)` – but there is no natural way to encode them in the statement notation that we use to sketch examples. Consequently, without extensions, this notation is not powerful enough to capture this specific aspect of LPGs. This raises the interesting question how a formal 1G model could look like in order to bridge this gap between the triple-centric abstraction of RDF and the vertex/edge-centric abstraction of LPGs. Should it be based on triples with extensions to capture LPG-related specifics or based on a vertex/edge-centric formalization with a mapping to statements, or are there even alternatives "in between" that provide a natural way to define mappings into both RDF and LPGs?

Several other interesting questions related to the notions of vertices, edges, vertex properties, and edge properties arise. For instance, Gremlin supports queries to enumerate all vertices using the expression `g.V()`, whereas edges can be retrieved via `g.E()`, and properties can then be obtained from these elements by dedicated path steps. A related key challenge in defining a semantics for LPG query languages over a potential 1G model then becomes how to define and distinguish the concepts of vertices and edges. Concretely, the question is how to align the concepts of vertices, edges, vertex properties, and edge properties with triples. By taking our statement notation as an example, the basic idea could be to map identifiers from the `src` and the `value` position of a statement to vertices, and use the SIDs as edge or property identifiers (depending on whether a statement carries an identifier, such as `Alice` or `Bob`, or a literal, such as a string `"Alice"` or `"Bob"`, in its `value` position, respectively). Another example is the LPG notion of a *vertex label*, which is typically used to classify a vertex. In this sense, this notion is very similar to that of using triples with the predicate `rdf:type` in RDF. However, vertex labels may also be used for other purposes, in which case mapping them to `rdf:type`-labeled statements may not be a desirable interpretation.

3.3. Challenge #3: Datatype alignment

A potential 1G model requires a unified type system over RDF and LPGs. RDF builds upon the XML Schema definition of datatypes and utilizes primitive XML Schema datatypes such as strings, numbers, and dates [2]. Since RDF is defined along an Open World paradigm, its datatypes tend to be more extensible and flexible than the types of values in LPGs. One specific aspect where RDF differs from LPG type systems is the absence of validation. For instance, nothing prevents users from adding ill-typed values such as `"this is not an integer"^^xsd:integer`. Another example is RDF's support for language tags as a built-in mechanism for localization. On the other hand, composite types such as lists, bags, and sequences are not available as "primary" literal types but need to be modeled explicitly using RDF containers [3, Section 5.1.1]. Based on such containers, RDF supports bags but not sets, because the graph structure offers no easy means to enforce the semantics of sets while also permitting the merge of RDF graphs.

The type system for LPGs, on the other hand, is not defined formally (to the best of our knowledge). Semantics of datatypes in LPGs are opaque and are typically "delegated" to the underlying implementation language, making it potentially hard to unify graph representation. Generally, we have a menagerie of datatypes to reconcile, and needless "baggage" because of the reliance on implementation languages. The challenge for a unifying graph data model is to define a *meta type system* that captures and aligns these different types, and gives them a concrete semantics. In contrast to RDF, however, LPGs typically support different composite datatypes (e.g., lists and sets)

[8] Note that RDF, effectively, *does not have vertices* that could exist in such a "stand-alone" sense. Instead, there is an infinite space of identifiers, any of which could be used as vertices in edges. This is not merely a philosophical notion, as we now see.

as built-in types. In other words, while RDF uses the graph structure to model composite types, in LPGs a property value itself may be an instance of a composite type. This feature reflects the general notion of semi-structured JSON documents as property values, and those composite types are also an integral part of LPG query languages.

One possible approach to align the type systems of RDF and LPGs more closely – with a focus on composite types – would be to leverage the notion of user-defined literals of RDF, which is an extension mechanism that can be used to syntactically represent arbitrary (simple and complex) types. As an alternative to the structural approach taken by RDF, a list could be represented as a literal whose lexical form is, say, `"[1, 2, 3]"^^:OneGraphList`, where `:OneGraphList` is a reserved datatype IRI defining the literal value to be a list with well-defined syntax that can be used interchangeably in RDF and LPGs.[9] This approach renders these datatypes opaque with respect to existing RDF semantics. For processing (say, with SPARQL) we could define specific access and folding/unfolding operations that allow users to access and expose the internal structure. In some ways, these operations could be considered an extension of the "literal value" notion that is part of the RDF specification [5, Section 3.3].

3.4. Challenge #4: Graph partitioning

RDF defines the notion of named graphs [5, Section 4], which are often used to support subgraph management use cases. Named graphs are sometimes thought of as an extension of the triple model to a quad model with the addition of a (sub)graph identifier. Some users have chosen to treat named graphs as containers (sometimes of a single triple) to make "statements about statements" (or sets of statements) in lieu of using the reification mechanism. This approach is, however, outside the formal semantics of RDF, since named graphs do not have any semantic theory associated with them in the RDF specification.

The absence of an explicit subgraph container mechanism in LPGs raises the question whether it is possible to map the concept of named graphs to existing LPG constructs. To sketch one possible approach, we extend our statement mechanism by a dedicated graph membership relation, where `inGraph` is a reserved `label`. With this idea, we could express named graph membership as follows:

$$s \xrightarrow{p} o : sid_1 \qquad sid_1 \xrightarrow{\texttt{inGraph}} g : sid_2$$

The motivation behind this approach is to restore symmetry to the data model instead of privileging named graphs as somehow special, treating named graphs as an application of the SID, much like *aspects* in [15]. Note, however, that such a representation does not dictate how a database system physically organizes the graph (i.e., a physical storage scheme may account for the special role of named graphs and encode them in a more compact form).

Looking at named graphs from this perspective reveals another idea: in Section 3.1 we used exactly the same pattern of SIDs in the `src` position to represent edge properties, which suggests that we could expose named graphs in LPGs as a (distinguished) edge or meta property. A positive aspect of this idea would be that it circumvents the need to explictly extend LPGs and LPG query languages to access and interpret named graph membership information. Note, however, that by this idea, named graph containers would be attached to edges and properties, not vertices. It remains debatable whether the notion of named graphs in an LPG should extend to vertices themselves.

3.5. Challenge #5: Graph merging and external identifiers

RDF has a definition for graph merging [12, Section 4.1], which is one of the distinguished benefits of RDF and, conversely, one of the weakest aspects of LPGs. Whenever multiple data sources are used, particularly from multiple organizations, graph merging is a key functionality. IRIs as global external identifiers are an essential part of this mechanism. Of course, it is possible to use IRIs as identifiers in LPGs, but there is more to graph merging: edge properties and particularly multiple edge instances (cf. Section 3.1) complicate any merging semantics. Specifically, under which conditions may two "similar" edges in two graphs that are to be merged be considered the "same" edge?

[9] The syntax makes no assertion about how the data are represented internally.

Allowing both RDF and LPG data to be represented in a single unifying model requires the co-existence of global identifiers (i.e., IRIs coming from RDF data) and local identifiers (i.e., vertex and edge identifiers from LPGs). User-defined default namespaces could then be used to expose local identifiers as if they were IRIs, whenever they are queried via SPARQL; conversely, IRIs originating from RDF data could be shortened according to existing namespace prefixes, to make querying and syntactic result representation for LPG query languages more user friendly.

While such a distinction would make it possible to load data via both RDF and LPG data formats into the same logical graph, it would result in a mere co-existence, without any (initial) overlap in vertex identifiers, labels, etc. Our vision for a unifying graph data model, however, goes beyond just co-existence of local and global identifiers: users may want to unify graph elements (such as vertices and edges) originating from RDF and LPGs. Assume, for instance, a user who maintains internal data about countries, which is available as an LPG in which the countries are identified via a simple country code string. Suppose the user wants to augment this data with information coming from an RDF dump of the Geonames[10] dataset. When initially loading the data sets, the LPG country identifier strings and the IRIs for the respective countries coming from Geonames would be disjoint. In order to enable users to align such identifiers, we would need to provide user-configurable rules that guide the merging process (in the example case, collating the country IRIs from the Geonames RDF data with the country code strings from the LPG). Such a mechanism could also be used to customize RDF graph merging when blank nodes are present (e.g., when two graphs derived from the same source are merged, one may want to treat blank node identifiers as indeed identifying the same blank nodes, which is something that is allowed under RDF semantics). Identifiers could also be generated using a templating mechanism such as the one employed in R2RML [7, Section 7.3].

3.6. Challenge #6: Lack of a formal foundation

Unlike SPARQL and SQL, existing LPG query languages – by and large – lack strict formal semantics (in the form of, say, a query algebra), which makes it hard to assess semantic compatibility of queries in different languages. Similarly, unlike for RDF, there does not exists a formal semantics for LPGs (or only exist *post hoc*, as in [17,25]). For LPG query languages, semantics is typically defined informally, either via documentation and examples or via an implementation. The Gremlin implementation in Neptune, for instance, is based in significant part on our interpretation of the informal Tinkerpop specification[11] – which comes with some ambiguous details – and on its reference implementation. As a simple example, consider ordering guarantees: if a Gremlin query contains an `order()` step in a non-terminal position, it is unclear whether (and how) subsequent steps should maintain that order. For instance, for the query `g.V(1,2,3,4).order().by('age').out()`, should the out-neighbors of the four vertices with IDs 1, 2, 3, and 4 be reported in the order implied by the age property of the four vertices? Similar questions arise in the context of dynamic typing, execution order guarantees, and runtime exceptions thrown through the query language. Also for openCypher, there is ambiguity. While openCypher does provide an official spec, many details remain unclear. Even for SPARQL, there is some ambiguity in its specification [13,19].

3.7. Challenge #7: Update semantics

In order to be able to subsume both RDF(-star) and LPGs, a unifying graph model needs to be as expressive as the "most expressive" model, in each of the considered dimensions. As we have illustrated in previous examples, certain extensions that are defined for the more expressive model may not have a natural representation in the less expressive model, thus introducing "dimensions" that are invisible when looking at the data from the less expressive model's perspective (e.g., recall the example of multiple edge instances as discussed in Section 3.1).

While semantics of read queries can be defined unambiguously by mapping a unified data model to a lower-dimensional level, the situation becomes more complex for queries that manipulate the data. Consider, for instance, the example graph sketched in Section 3.1 and assume a SPARQL update statement (not SPARQL-star) to drop the triple (`Bob`, `knows`, `Alice`). From a SPARQL query perspective, this triple would show up once, but the question is whether dropping it should, behind the scenes, delete both simple statements? And if so, should deletion include

[10] https://www.geonames.org/

[11] https://tinkerpop.apache.org/gremlin.html

the deletion of the attached edge properties (which exposes a risk for accidental over-deletion)? Would adding an edge for every subject with IRI `:Bob` to a new person introduce one or two edges? Also, by using SPARQL-star to add a new edge property to the example graph, which of the two edges should the property be attached to?

The common root cause for the ambiguity of all these scenarios is that we request updates over a simplified, dimensionally reduced view of the data. While the discussion of all these questions goes beyond the scope of this paper, the semantics of such operations needs to be carefully designed in order to avoid unwanted side effects, as to obtain a "natural" behavior in scenarios where data is queried and manipulated via different query languages.

4. Concluding remarks and the way forward

A recent survey of organizations working on, or considering to adopt, knowledge graphs found that *interoperability and standards* have a very high priority among survey respondents, and *data integration* was seen as the dominant use case [1]. These findings could be interpreted to suggest a need for RDF/LPG compatibility and unification. While it may seem that making RDF and LPGs fully compatible is not possible (as per the official RDF specifications and the emerging RDF-star work), we believe there is a way forward. Minimally, we must address the challenges of edge identity (multiple similar edges), graph merging, and well-defined update semantics across languages. One way to make progress would be to define some kind of "compatibility subset" to cover enough ground so that most RDF and LPG applications work with no or minimal modifications. Lack of interoperability slows the overall adoption of graph technologies and, thus, should be of high priority to be addressed by the graph data community.

In this paper we have sketched the 1G vision of a unifying model as a source of ideas about where to go next. The open questions we have posed imply lots of interesting research, the outcomes of which will have significant practical relevance. We look forward to working with the broader community to explore solutions to these topics.

References

[1] M. Atkin, T. Deely and F. Scharffe, 2021 Knowledge Graph Industry Survey, 2021. doi:10.5281/zenodo.4950097.

[2] P.V. Biron and A. Malhotra, XML Schema Part 2: Datatypes (Second Edition), W3C Recommendation, World Wide Web Consortium, 2004, https://www.w3.org/TR/xmlschema-2/.

[3] D. Brickley and R.V. Guha, RDF Schema 1.1, W3C Recommendation, World Wide Web Consortium, 2014, https://www.w3.org/TR/rdf-schema/.

[4] H. Chiba, R. Yamanaka and S. Matsumoto, G2GML: Graph to graph mapping language for bridging RDF and property graphs, in: *Proceedings of the 19th International Semantic Web Conference (ISWC)*, 2020. doi:10.1007/978-3-030-62466-8_11.

[5] R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J.J. Carroll and B. McBride, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, World Wide Web Consortium, 2014, http://www.w3.org/TR/rdf11-concepts/.

[6] S. Das, J. Srinivasan, M. Perry, E.I. Chong and J. Banerjee, A tale of two graphs: Property graphs as RDF in oracle, in: *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014*, Athens, Greece, March 24–28, 2014, 2014. doi:10.5441/002/edbt.2014.82.

[7] S. Das, S. Sundara and R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation, World Wide Web Consortium, 2012.

[8] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez and D. Vrandecic, Introducing Wikidata to the linked data web, in: *Proceedings of the 13th International Semantic Web Conference (ISWC)*, 2014. doi:10.1007/978-3-319-11964-9_4.

[9] A. Green, M. Junghanns, M. Kießling, T. Lindaaker, S. Plantikow and P. Selmer, *openCypher: New Directions in Property Graph Querying, in: Proceedings of the 21st International Conference on Extending Database Technology (EDBT)*, 2018. doi:10.5441/002/edbt.2018.62.

[10] S. Harris and A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation, World Wide Web Consortium, 2013, https://www.w3.org/TR/sparql11-query/.

[11] O. Hartig, P.-A. Champin, G. Kellogg and A. Seaborne, RDF-star and SPARQL-star, W3C Community Group Report, World Wide Web Consortium, 2021, https://www.w3.org/2021/12/rdf-star.html.

[12] P.J. Hayes and P.F. Patel-Schneider, RDF 1.1 Semantics, W3C Recommendation, World Wide Web Consortium, 2014, http://www.w3.org/TR/rdf11-mt/.

[13] D. Hernández, C. Gutierrez and R. Angles, Correlation and Substitution in SPARQL, 2016, arXiv preprint arXiv:1606.01441. doi:10.48550/arXiv.1606.01441.

[14] O. Lassila, RDF-star use cases from Amazon Neptune, 2021, https://lists.w3.org/Archives/Public/public-rdf-star/2021Dec/att-0001/rdf-star-neptune-use-cases-20211202.pdf.

[15] O. Lassila, M. Mannermaa, I. Oliver and M. Sabbouh, Aspect-Oriented Data, Accepted submission to the W3C Workshop on RDF Next Steps, World Wide Web Consortium, 2010.
[16] F. Manola, E. Miller and R.D.F. Primer, RDF Primer, W3C Recommendation, World Wide Web Consortium, 2004, https://www.w3.org/TR/rdf-primer/.
[17] J. Marton, G. Szárnyas and D. Varró, Formalising openCypher graph queries in relational algebra, in: *European Conference on Advances in Databases and Information Systems*, 2017. doi:10.1007/978-3-319-66917-5_13.
[18] V. Nguyen, O. Bodenreider and A.P. Sheth, Don't like RDF reification?: Making statements about statements using singleton property, in: *Proceedings of the 23rd International World Wide Web Conference (WWW)*, 2014. doi:10.1145/2566486.2567973.
[19] P.F. Patel-Schneider and D. Martin, EXISTStential aspects of SPARQL, in: *International Semantic Web Conference (Posters & Demos)*, Vol. 1690, CEUR-WS, 2016.
[20] E. Prud'hommeaux and C. Buil-Aranda, SPARQL 1.1 Federated Query, W3C Recommendation, World Wide Web Consortium, 2013.
[21] S. Purohit, N. Van and G. Chin, Semantic property graph for scalable knowledge graph analytics, in: *Proceedings of the 2021 IEEE International Conference on Big Data (Big Data)*, 2021. doi:10.1109/BigData52589.2021.9671547.
[22] I. Robinson, J. Webber and E. Eifrém, *Graph Databases: New Opportunities for Connected Data*, 2nd edn, O'Reilly Media, Inc., 2015.
[23] M.A. Rodriguez, The gremlin graph traversal machine and language (invited talk), in: *Proceedings of the 15th Symposium on Database Programming Languages (DBPL 2015)*, 2015, pp. 1–10. doi:10.1145/2815072.2815073.
[24] M.A. Rodriguez and P. Neubauer, Constructions from dots and lines, *Bulletin of the American Society for Information Science and Technology* **36**(6) (2010), 35–41. doi:10.1002/bult.2010.1720360610.
[25] J. Shinavier and R. Wisnesky, Algebraic Property Graphs, 2019, arXiv preprint arXiv:1909.04881. doi:10.48550/arXiv.1909.04881.
[26] D. Tomaszuk, R. Angles and H. Thakkar, PGO: Describing Property Graphs in RDF, *IEEE Access* **8** (2020). doi:10.1109/ACCESS.2020.3002018.