Vision Paper

Breaking Down the Data-Metadata Barrier for Effective Property Graph Data Management

Sepehr Sadoughi
TU Eindhoven
Eindhoven, The Netherlands
s.sadoughi@tue.nl

Nikolay Yakovets
TU Eindhoven
Eindhoven, The Netherlands
hush@tue.nl

George Fletcher
TU Eindhoven
Eindhoven, The Netherlands
g.h.l.fletcher@tue.nl

ABSTRACT

The ISO standard Property Graph model has become increasingly popular for representing complex, interconnected data. However, it lacks native support for querying metadata and reification, which limits its abilities to deal with the demands of modern applications. We introduce the vision of Meta-Property Graph, a backwards compatible extension of the property graph model addressing these limitations. Our approach enables first-class treatment of labels and properties as queryable objects and supports reification of substructures in a graph. We propose MetaGPML, a backwards compatible extension of the Graph Pattern Matching Language forming the core of the ISO standard GQL, to query these enhanced graphs. We demonstrate how these foundations pave the way for advanced data analytics and governance tasks that are challenging or impossible with current property graph systems.

1 INTRODUCTION

Modern data engineering applications increasingly demand flexibility and agility in handling data and metadata more than ever before. During exploratory analytics, structure emerges gradually without a predefined schema. Heterogeneity in what is a data value versus what is an attribute name is inherent during discovery, profiling, and exploration. In data integration scenarios, such as building knowledge graphs, data and schema heterogeneity is inevitable: what appears as a data value in one source might be a node label or property name in another, while a subgraph in one source might correspond to a single node elsewhere. Modern data management solutions must therefore fully support heterogeneity and fluid boundaries between data and metadata.

Metadata is commonly understood in two distinct ways. The first is *attribute metadata* - characteristics associated with data objects, such as column names in relational databases or node labels in graphs. For instance, when storing contact information, attributes like name and email are metadata. The second form is *reification*- representing an aggregation of complex relationships or structures into a new entity or data object, making it easier to manage and analyze. In the context of entity-relationship modeling, reification is often used to transform relationship sets into entity sets, allowing for more effective data modeling and manipulation. For example, in an e-commerce system, reifying the relationship between customers and orders creates an order history entity that can be analyzed and queried.[1]

The ISO standard Property Graph (PGs) model has gained popularity in graph data management and is widely adopted, e.g., in graph DB systems such as Neo4j, Tigergraph, and Amazon Neptune, as well as relational DB systems such as DuckDB which implement the ISO extensions to SQL for PG querying. In a property graph nodes and edges are labeled and also have associated sets of property name/value pairs (e.g., a node labeled `Person` with property `Birthdate` having data value 11-11-2001; here `Person` and `Birthdate` are attribute metadata). While PGs offer a model closely aligned with domain representations, the model (1) strictly separates metadata from data and (2) lacks support for reification. In contrast, the W3C's RDF graph model [18] treats everything as data, including node and edge attributes, and natively supports reification. This makes RDF particularly useful for applications requiring metadata validation, verification, and governance. There is an opportunity to bring these powerful features to PGs while preserving their core design principles and inherent strengths in handling data heterogeneity.

In this paper, we present a vision for overcoming barriers to flexible management of data-metadata heterogeneity in PG data management applications. Towards this, we introduce Meta-Property Graph (MPG), a fully backwards compatible extension of the PG model that addresses limitations in representing and querying metadata. Our approach enables first-class treatment of labels and properties as queryable data objects, as well as reification of subgraphs. On this foundation, we further propose MetaGPML, a fully backwards compatible extension of the Graph Pattern Matching Language (GPML), the core language at the heart of the ISO standard GQL for PG querying. We give a complete formal specification of MPG and MetaGPML, providing the foundations for our vision. Furthermore, we demonstrate how these contributions facilitate advanced data analytics, integration, and governance tasks that are challenging or impossible with current PG systems.

2 RELATED WORK

The PG model represents a design point on the continuum between the Relational (fixed metadata) and RDF (no fixed metadata) data models. Of course, all points along this design continuum are equally important, finding their applications and use-cases. And indeed, data cleaning, wrangling, integration, and exchange are very often about moving (meta)data along the continuum. What has been missing is an appropriate design for PGs to fully meet the modeling and querying needs of data management scenarios such as these (see Chapter 2 of [5] for a survey of design approaches for PG modeling).

In the research literature, dealing with the challenges of data-metadata heterogeneity was studied in the context of relational data integration and data integration on the web, leading to solutions for relational and XML data-metadata mapping and exchange (e.g., [4, 10, 15]) and seminal work such as the SchemaSQL and FISQL data-metadata query languages for relational databases [16, 22]. Our work approaches these challenges in the new context of the ISO standards for graph data management.

[1]Other more complex types of metadata, such as reflective or active data [7, 9, 19], are beyond the scope of our current discussion.

In another direction, there is a growing body of work on mappings between graph data models [1, 6, 14, 20, 21] and generalizing both PG and RDF [3, 13, 17]. Our vision is orthogonal and complementary to these investigations, aiming at extending the PG model and the standardized GQL and SQL/PGQ query languages with seamless data-metadata functionalities.

3 A GUIDED TOUR OF WORKING WITH MPGS

Figure 1 illustrates a sample MPG database of publications, indexing databases, and persons. Unlike standard PGs, our model treats four types of data objects as first-class citizens: edges (E_G), nodes (N_G), properties (P_G), and labels (L_G). Table 1 introduces the pattern notation used in our examples, which will be formally defined in Section 5.

Table 1: Data objects pattern notation

Nodes	
`(x:l)`	node variable (nv) x with label l
`(x:l).z`	nv x with label l and property variable z
`(x::π)`	nv x with a pattern π in its reified substructure
Edges	
`-[x:l]->`	edge variable x with label l
`-[x:?y]->`	edge variable x with label set variable y
`-[x].z->`	edge variable x with property variable z
Properties	
`{x}`	property variable x
`KEY(x)`, `VAL(x)`	key and value of property variable x
Labels	
`|x|`	label set variable x
`c ELEMENTOF x`	check if label c exists in label set variable x

3.1 Working with Data, Labels, and Properties

Meta-Property Graph allows direct querying of label sets as first-class data objects, independent of their associated nodes or edges. This enables powerful analytics over class structures in the database. For example, query Q_1 retrieves all label sets containing `Publication` to discover co-occurring tags:

Q_1: Which label sets contain the label `Publication`?

```
MATCH |l|
WHERE "Publication" ELEMENTOF l
RETURN l AS "Publication_Co_Tags"
```

Result table of Q_1

Publication_Co_Tags
{"Publication", "Journal"}
{"Publication", "Conference"}

With MetaGPML we can also query properties as independent data objects. Query Q_2 demonstrates this by retrieving the values of all `Name` properties, regardless of whether the property is attached to a node or an edge.

Q_2: What are the values of `Name` properties?

```
MATCH {p}
WHERE KEY(p) = "Name"
RETURN VAL(p) AS "Names"
```

Result table of Q_2

Names
Lee
Scopus
Rose
PubMed

MetaGPML's ability to bind properties and label sets to variables enables fluid movement between data and metadata. Q_3 demonstrates this by matching relationships between publications and indexing databases and treating metadata of `LABEL(y)` as data in the return clause. Q_4 shows how treating metadata as queryable data helps identify potential reviewers by matching researchers' fields with publication properties—data and metadata comparison, which is a task difficult with standard PG queries.

Q_3: Which relationships does each publication have with each indexing databases?

```
MATCH (x:Publication)-[:?y]->(z:
    Indexing_DB)
RETURN x.Title AS "Title", LABEL(y) AS
z.Name
```

Query results are shown below, where `LABEL` resolves label sets bound to `y`, while `z.Name` values form column headers:

Result bindings of Q_3

{ (Title ↦ "Nature Studies", Scopus ↦ {"Archived"}),
(Title ↦ "Nature Studies", PubMed ↦ {"Indexed"}),
(Title ↦ "Biology Advancements", PubMed ↦ {"Indexed"}) }

Q_4: Finding reviewers based on research fields

```
MATCH (x:Person), (y:Publication).z
WHERE x.ResearchField = KEY(z)
RETURN x.Name AS "Reviewer candidate",
       y.Name AS "Publication venue",
       KEY(z) AS "Research field"
```

Result table of Q_4

Reviewer candidate	Publication venue	Research field
Lee	Nature Studies	Biology
Lee	Biology advancement	Biology
Rose	Nature Studies	Ecology

The graph structure treats publication fields as metadata, allowing us to match a publication's research fields with potential reviewers' expertise through their `ResearchField` property.

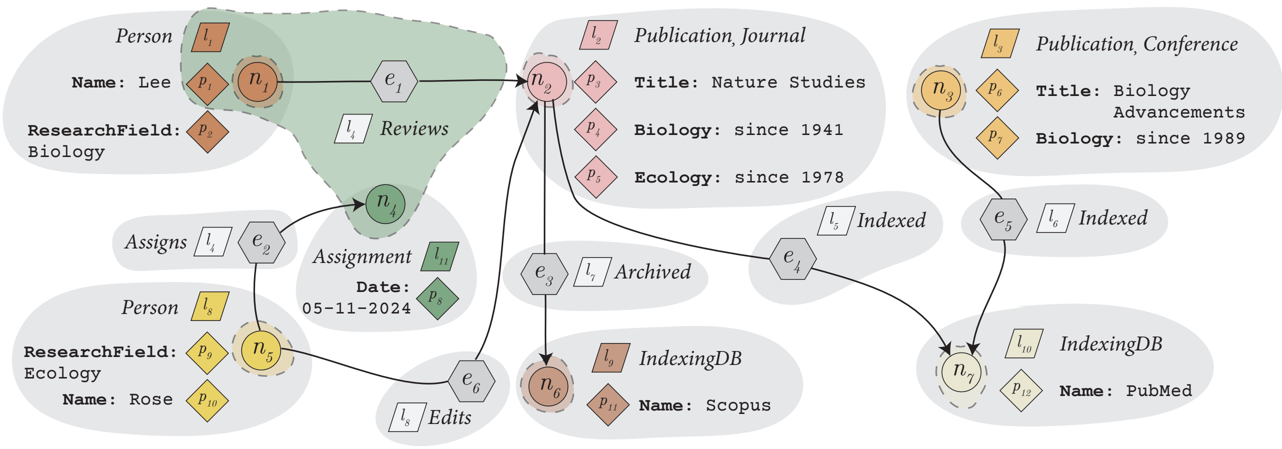

Figure 1: Example meta-property graph G

3.2 Working with Reification

MPG and MetaGPML enable reification as the second type of metadata on property graphs. A sub-structure of an MPG can be reified as a node, thereby making it a first-class citizen. This enables annotating part of the graph by assigning properties and labels to a node that reifies that part. It is worth mentioning that the part of the graph that is being reified may or may not be a proper graph, which means that a node can reify some data objects without their assigned relationships in the graph, e.g., a set of specific properties or relationships, can be reified without the nodes they are associated with.

In Figure 1, we represent the statement "Rose assigned Lee as a reviewer on 5th November 2024" by reifying the review relationship into a node. Query Q_5 retrieves the assigning editor's name:

Q_5: Who assigned Lee as a reviewer and when?

```
MATCH   (x:Person)-[:assigns]->
        (y::(z:Person)-[:reviews]->())
WHERE   z.Name = "Lee"
RETURN   z.Name AS "reviewer name",
         y.Date AS "Date",
         x.Name AS "Assigning editor"
```

Note that in the `MATCH` clause we have a graph pattern embedded in a node pattern, to denote a query to be executed on the sub-structure reified by the node which is bound to variable `y`.

Result table of Q_5

Reviewer name	Date	Assigning editor
Lee	05-11-2024	Rose

While it may look that MPG is a hypergraph model it's important to note that hypergraphs focus on complex relationships, while reification focuses on complex objects. In MPG, edges still connect exactly two nodes. Nodes can act as meta-nodes, with other data objects (nodes, edges, properties, label sets) assigned for reification.

4 META-PROPERTY GRAPH MODEL

We next formalize Meta-Property Graphs. Let $\mathcal{I}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{V}$ be pairwise disjoint sets of object identifiers, labels, property keys, and property values, respectively.

Definition 4.1. *A meta-property graph is a directed and undirected vertex- and edge-labeled graph $G = (N, E, P, L, \lambda, \mu, \sigma, \upsilon, \eta, \rho)$, where:*

- *$N, E, P, L \subseteq \mathcal{I}$ are finite, pairwise disjoint sets,*
- *$\mu : L \rightarrow 2^{\mathcal{L}}$ assigns a finite set of labels to each label set identifier,*
- *$\lambda : N \cup E \rightarrow L$ is a bijective labeling function assigning a label set identifier to each node and edge,*
- *$\upsilon : P \rightarrow \mathcal{K} \times \mathcal{V}$ assigns key-value pairs to properties,*
- *$\sigma : N \cup E \rightarrow \mathbb{C}$ assigns compatible property sets to nodes and edges, such that for each pair of distinct objects $o_1, o_2 \in N \cup E$, it holds that $\sigma(o_1) \cap \sigma(o_2) = \emptyset$ and $\bigcup_{o \in N \cup E} \sigma(o) = P$, i.e., every and each property $p \in P$ is assigned to exactly one node or edge.*
- *$\eta = (\eta_s, \eta_t, \eta_u)$ where:*
  - *$\eta_s, \eta_t : E_d \rightarrow N$ assign source and target nodes to directed edges,*
  - *$\eta_u : E_u \rightarrow \{\{u, v\} \mid u, v \in N\}$ assigns node pairs to undirected edges,*
- *$\rho : N \rightarrow 2^{N \cup E \cup P \cup L}$ assigns finite sets of objects to nodes, such that for each $n \in N$, it holds that $n \notin \rho^*(n)$, where $\rho^*(n)$ is the closure of $\rho(n)$,[2] ensuring that the sub-structure associated with each node is well-founded.*

where $E = E_d \cup E_u$, and the set of compatible property sets $\mathbb{C} = \{C \subseteq P \mid \forall p_1, p_2 \in C, p_1 \neq p_2 \Rightarrow Key(p_1) \neq Key(p_2)\}$, $Key(p) = \pi_1(\upsilon(p))$, $Val(p) = \pi_2(\upsilon(p))$.

Design decisions. The constraints we have imposed in our design of MPGs are minimalistic, ensuring only that all properties are uniquely assigned to edges and nodes, and that reification is well-founded. For instance, there is no constraint preventing a node from having an outgoing edge to a subgraph containing one of the node's own properties. This flexibility allows for

[2] Formally, $\rho^0(n) = \rho(n)$, $\rho^i(n) = \rho^{i-1}(n) \cup \bigcup_{n' \in N \cap \rho^{i-1}(n)} \rho(n')$, and $\rho^*(n) = \bigcup_{i=0}^{\infty} \rho^i(n)$.

representing scenarios such as "Mary legally changed her first name" or "Mary audits all names in the database (including her own)". Similarly, there is no constraint requiring that for each edge in $\rho(n)$, the source and target of the edge must also be in $\rho(n)$, which, for instance, can be particularly valuable in applications with privacy preservation concerns. As an example, this permits representations like "Mary is confident that Bob edited a book, but not confident about which book", where only the node representing Bob and Bob's outgoing edge labeled 'edited' are included, without the target of this edge. Of course, such additional constraints can be added (or existing constraints can be removed) as appropriate to the specific application domain. It is worth noting that MPGs support meta-properties, i.e., properties on properties. For instance, we can represent "John's birthdate was entered on 23 March 2020" since ρ can consist of just a singleton containing a property. For simplicity, we have omitted special treatment of meta-properties in our main presentation. We next define the graph substructure associated with a node.

Definition 4.2. *(Sub-Structure) Let $G = (N, E, P, L, \lambda, \mu, \sigma, \upsilon, \eta, \rho)$ be a meta-property graph and $n \in N$. The sub-structure of G induced by n is $G_n = (N_n, E_n, P_n, L_n, \lambda_n, \mu_n, \sigma_n, \upsilon_n, \eta_n, \rho_n)$ where:*

- $N_n = N \cap \rho(n), E_n = E \cap \rho(n), L_n = L \cap \rho(n), P_n = P \cap \rho(n)$
- $\upsilon_n = \upsilon|_{P_n}, \mu_n = \mu|_{L_n}$ *(domain restrictions)*
- $\lambda_n(o) = \lambda(o)$ *for* $o \in N_n \cup E_n$ *if* $\lambda(o) \in \rho(n)$*, undefined otherwise*
- $\sigma_n(o) = \sigma(o) \cap \rho(n)$ *for* $o \in N_n \cup E_n$
- $E_{dn}, E_{un} \subseteq E_n$ *are directed and undirected edge subsets*
- $\eta_n = (\eta_{sn}, \eta_{tn}, \eta_{un})$ *where:*
  - *For* $e \in E_{dn}$*:*
    - $\eta_{sn}(e) = \eta_s(e)$ *if* $\eta_s(e) \in \rho(n)$*, undefined otherwise*
    - $\eta_{tn}(e) = \eta_t(e)$ *if* $\eta_t(e) \in \rho(n)$*, undefined otherwise*
  - $\eta_{un}(e) = \eta_u(e) \cap \rho(n)$ *for* $e \in E_{un}$
- $\rho_n(n') = \rho(n) \cap \rho(n')$ *for* $n' \in N_n$*, satisfying the same well-foundedness property as in Definition 4.1.*

Intuitively, the sub-structure G_n is a "view" of the meta-property graph G focused on node n. It includes only nodes, edges, properties, and labels explicitly linked to n through ρ, maintaining their relationships within this subset. This enables analysis of complex nested structures while preserving their context in the larger graph.

5 META-PROPERTY GRAPH PATTERN MATCHING LANGUAGE

We next formalize our Graph Pattern Matching Language for Metaproperty Graphs (MetaGPML) as an extension to the Graph Pattern Matching Language (GPML) [8, 11, 12]. GPML is the core element of both SQL/PGQ and GQL languages standardized by ISO. By extending GPML we ensure backward compatibility in the sense that every GPML query is a MetaGPML query.

5.1 Syntax

Let $\mathcal{X}$ be a countable infinite set of variables. We introduce descriptors δ_{nodes} and δ_{edges} in MetaGPML, consistent with GPML [12]. For $\mathtt{x}, \mathtt{y}, \mathtt{z} \in \mathcal{X}$, $\ell \in \mathcal{L}$, $k \in \mathcal{K}$, and $v \in \mathcal{V}$, we have

Descriptors:

$$\begin{aligned} \delta_{nodes} &:= \delta_{edges} \mid \mathtt{::}\pi \mid \mathtt{:}\ell\mathtt{::}\pi \mid \mathtt{x::}\pi \mid \mathtt{x:}\ell\mathtt{::}\pi \\ &\quad\mid \mathtt{:?y::}\pi \mid \mathtt{x:?y::}\pi \\ \delta_{edges} &:= \mathtt{x} \mid \mathtt{:}\ell \mid \mathtt{:?y} \mid \mathtt{x:}\ell \mid \mathtt{x:?y} \end{aligned} \tag{1}$$

The descriptors in (1) will be used in the following patterns to assign variables and constants to different data objects. A pattern π in MetaGPML and operations on it can be formulated as follows:

Patterns:

$$\begin{aligned} \pi := \;& \mathtt{(}\delta_{nodes}\mathtt{)} \mid \mathtt{()} && \text{(node pattern)} \\ & \mid \mathtt{(}\delta_{nodes}\mathtt{).x} \mid \mathtt{().x} \\ & \mid \mathtt{-[}\delta_{edges}\mathtt{]->} \mid \mathtt{<-[}\delta_{edges}\mathtt{]-} && \text{(edge pattern)} \\ & \mid \mathtt{-[}\delta_{edges}\mathtt{]-} \\ & \mid \mathtt{-[]->} \mid \mathtt{<-[]-} \mid \mathtt{-[]-} \\ & \mid \mathtt{-[}\delta_{edges}\mathtt{].x->} \\ & \mid \mathtt{<-[}\delta_{edges}\mathtt{].x-} \\ & \mid \mathtt{-[}\delta_{edges}\mathtt{].x-} \\ & \mid \mathtt{\{x\}} \mid \mathtt{\{\}} && \text{(property pattern)} \\ & \mid \mathtt{|x|} \mid \mathtt{||} && \text{(label pattern)} \\ & \mid \pi\pi && \text{(concatenation)} \\ & \mid \pi\mathtt{+}\pi && \text{(union)} \\ & \mid \pi \;\mathtt{WHERE}\; \Phi && \text{(conditioning)} \\ \Pi := \;& \pi \mid \Pi, \Pi && \text{(graph pattern)} \end{aligned} \tag{2}$$

We define $\mathcal{E}$ as the *expression* for *condition* Φ that can be used in a pattern after the `WHERE` keyword as follows, considering $c \in \mathcal{K} \cup \mathcal{V} \cup \mathcal{L}$.

Expressions and conditions:

$$\begin{aligned} \mathcal{E} := \;& \mathtt{x} \mid \mathtt{x.}k \mid v \mid k \mid \ell && \text{(expressions)} \\ & \mid \mathtt{KEY(x)} \mid \mathtt{VAL(x)} \mid \mathtt{LABEL(x)} \\ \Phi := \;& \mathcal{E} = \mathcal{E} \mid \mathcal{E} < \mathcal{E} \mid \mathtt{x} : \ell && \text{(conditions)} \\ & \mid \mathtt{SUBSETEQ(x,\ y)} \\ & \mid c \;\mathtt{ELEMENTOF\ y} \\ & \mid \Phi \;\mathtt{AND}\; \Phi \mid \Phi \;\mathtt{OR}\; \Phi \mid \mathtt{NOT}\; \Phi \end{aligned} \tag{3}$$

It is important to note that `KEY(x)` and `VAL(x)` operate on properties as first-class data objects, similar to nodes and edges. The same applies to `LABEL(x)`, which operates on label sets. These functions should not be confused with similar functions in other PG query languages like OpenCypher, which only simulate these capabilities at the query language level within the conventional PG data model. Finally, based on (2) and (3), we formulate the definition of *clauses* and *queries*.

Clauses and queries:

$$\begin{aligned} \mathcal{C} &:= \mathtt{MATCH}\; \Pi \\ &\quad\mid \mathtt{FILTER}\; \Phi \\ \mathcal{Q} &:= \mathcal{C}\mathcal{Q} \\ &\quad\mid \mathtt{RETURN}\; \mathcal{E}_1 \;\mathtt{AS}\; x_1, \ldots, \mathcal{E}_n \;\mathtt{AS}\; x_n \end{aligned} \tag{4}$$

MetaGPML syntax builds on GQL, using familiar patterns: nodes in parentheses and edges in brackets, with labels prefixed by colons. The double-colon operator (::) enables querying of reified subgraphs. Variables are distinguished from constants using question mark prefixes. Set operations on labels and properties use predicates like `ELEMENTOF` and `SUBSETEQ`.

5.2 Semantics

Next, we present the semantics of our language as a clear, unambiguous foundation for further study and implementation. To avoid unnecessary complexity, we presented the most vital semantics to illustrate the concept to understand how MetaGPML works.

Definition 5.1. *(Bindings) A binding $\beta : \mathcal{X} \to \mathbb{V}$ assigns variables $x \in \mathcal{X}$ to values $v \in \mathbb{V}$, where $\mathbb{V} = \mathcal{I} \cup \mathcal{L} \cup \mathcal{K} \cup \mathcal{V}$. We denote this as $(x_1 \mapsto v_1, ..., x_n \mapsto v_n)$, where $x_1, ..., x_n$ are the variables in the domain of β ($Dom(\beta)$) and $v_1, ..., v_n$ are their corresponding values. We denote the empty binding as $()$.*

Definition 5.2. *(Compatibility of Bindings and their Join) Two bindings β_1, β_2 are compatible, denoted by $\beta_1 \sim \beta_2$, if they agree on their shared variables. Specifically, for every $x \in Dom(\beta_1) \cap Dom(\beta_2)$, it holds that $\beta_1(x) = \beta_2(x)$.*

When $\beta_1 \sim \beta_2$, their join $\beta_1 \bowtie \beta_2$ is defined as follows: $Dom(\beta_1 \bowtie \beta_2) = Dom(\beta_1) \cup Dom(\beta_2)$. For any variable x, $(\beta_1 \bowtie \beta_2)(x) = \beta_1(x)$ if $x \in Dom(\beta_1) \setminus Dom(\beta_2)$, and $(\beta_1 \bowtie \beta_2)(x) = \beta_2(x)$ if $x \in Dom(\beta_2)$.

A pattern matching semantic $[\![\pi]\!]_G$ in meta-property graph is a set of bindings β that assign variables to values.

Node pattern matching:

$$[\![()]\!]_G = \{() \mid n \in N\} \quad (5)$$
$$[\![(\mathtt{x})]\!]_G = \{(\mathtt{x} \mapsto n) \mid n \in N\} \quad (6)$$
$$[\![(\mathtt{x}:\ell)]\!]_G = \{(\mathtt{x} \mapsto n) \mid n \in N, \ell \in \mu(\lambda(n))\} \quad (7)$$
$$[\![(\mathtt{x}:?\mathtt{y})]\!]_G = \{(\mathtt{x} \mapsto n, \mathtt{y} \mapsto l) \mid n \in N, l = \lambda(n)\} \quad (8)$$
$$[\![(\mathtt{x}).\mathtt{z}]\!]_G = \{(\mathtt{x} \mapsto n, \mathtt{z} \mapsto p) \mid n \in N, p \in \sigma(n)\} \quad (9)$$

Equation 5 matches any node $n \in N$ in graph G. Equation 6 assigns a node to variable x. Equation 7 matches nodes with specific labels, where $\mu(\lambda(n))$ determines the node's labels. Equation 8 assigns a node's label set to variable y. Finally Equation 9 assigns property p of node n to variable z.

Meta-nodes are a key concept in MetaGPML, defined as nodes to which diverse elements are assigned based on the concept of sub-structure. This feature enables reification within the Meta-Property Graph, allowing for a more nuanced representation of complex relationships and structures.

Equation 10 demonstrates the matching of nodes with labels to patterns within their associated sub-structure:

Meta-node pattern matching:

$$[\![(\mathtt{x}:\ell::\pi)]\!]_G = [\![(\mathtt{x}:\ell)]\!]_G \bowtie [\![\pi]\!]_{G_n} \quad (10)$$

The meta-node semantic definition involves a full join operation between two patterns divided by :: in the node pattern descriptor. Here, $[\![(\mathtt{x}:\ell)]\!]_G$ represents a simple node pattern matching, while $[\![\pi]\!]_{G_n}$ is a pattern matching within the G_n which is a sub-structure reified by node n that assigned to variable x. We define the semantics of edges in our model, supporting both directed and undirected edges.

Edge pattern matching:

$$[\![\mathtt{-[]->}]\!]_G = \{() \mid e \in E_d\} \quad (11)$$
$$[\![\mathtt{-[x]->}]\!]_G = \{(\mathtt{x} \mapsto e) \mid e \in E_d\} \quad (12)$$
$$[\![\mathtt{-[:}\ell\mathtt{]->}]\!]_G = \{() \mid e \in E_d, \ell \in \mu(\lambda(e))\} \quad (13)$$
$$[\![\mathtt{-[x:}\ell\mathtt{]-}]\!]_G = \{(\mathtt{x} \mapsto e) \mid e \in E_u, \ell \in \mu(\lambda(e))\} \quad (14)$$
$$[\![\mathtt{-[x:?y]->}]\!]_G = \{(\mathtt{x} \mapsto e, \mathtt{y} \mapsto l) \mid e \in E_d, l = \lambda(e)\} \quad (15)$$
$$[\![\mathtt{-[x].z->}]\!]_G = \{(\mathtt{x} \mapsto e, \mathtt{z} \mapsto p) \mid e \in E_d, p \in \sigma(e)\} \quad (16)$$

Equations 11-14 match various edge patterns, including directed and undirected edges with or without labels. Equation 15 allows assigning the edge's label set to variable y, while 16 enables assigning the edge's properties to variable z.

The meta-property graph model enables defining specific semantics for property and label objects, enhancing query capabilities on these elements.

Property and label objects:

$$[\![\{\mathtt{x}\}]\!]_G = \{(\mathtt{x} \mapsto p) \mid p \in P\} \quad (17)$$
$$[\![|\mathtt{x}|]\!]_G = \{(\mathtt{x} \mapsto l) \mid l \in L\} \quad (18)$$

Pattern concatenation, union, and conditioning:

$$[\![\pi_1\pi_2]\!]_G = \{\beta_1 \bowtie \beta_2 \mid \beta_i \in [\![\pi_i]\!]_G, i = 1,2 \text{ and } \beta_1 \sim \beta_2\} \quad (19)$$
$$[\![\pi_1 + \pi_2]\!]_G = \{\beta \cup \beta' \mid \beta \in [\![\pi_1]\!]_G \cup [\![\pi_2]\!]_G\} \quad (20)$$
$$[\![\pi \text{ WHERE } \Phi]\!]_G = \{\beta \in [\![\pi]\!]_G \mid [\![\Phi]\!]_G^\beta = True\} \quad (21)$$

In (20), β' maps any variable in $[\![\pi_1 + \pi_2]\!]_G$ not in β's domain to $Null$.

Graph patterns:

$$[\![\Pi_1, \Pi_2]\!]_G = \{\beta_1 \bowtie \beta_2 \mid \beta_i \in [\![\Pi_i]\!]_G, i = 1,2 \text{ and } \beta_1 \sim \beta_2\} \quad (22)$$

The semantics $[\![\mathcal{E}]\!]_G^\beta$ of an expression $\mathcal{E}$ is computed with respect to binding β over graph G. For a variable x in β's domain, $[\![\mathtt{x}]\!]_G^\beta = \beta(x)$.

Node and property values:

$$[\![\mathtt{x}.k]\!]_G^\beta = \begin{cases} Val(p) & \text{if } \beta(\mathtt{x}) \in N \cup E, p \in \sigma(\beta(\mathtt{x})), Key(p) = k \\ Null & \text{otherwise} \end{cases} \quad (23)$$

Property and Label set operations:

$$[\![\mathsf{KEY}(\mathtt{x})]\!]_G^\beta = Key([\![\mathtt{x}]\!]_G^\beta) \text{ for } [\![\mathtt{x}]\!]_G^\beta \in P \quad (24)$$
$$[\![\mathsf{VAL}(\mathtt{x})]\!]_G^\beta = Val([\![\mathtt{x}]\!]_G^\beta) \text{ for } [\![\mathtt{x}]\!]_G^\beta \in P \quad (25)$$
$$[\![\mathsf{LABEL}(\mathtt{x})]\!]_G^\beta = \mu([\![\mathtt{x}]\!]_G^\beta) \text{ for } [\![\mathtt{x}]\!]_G^\beta \in L \quad (26)$$

The semantics $[\![\Phi]\!]_G^\beta$ of a condition Φ is an element in $\{True, False, Null\}$, computed with respect to β as follows.

Conditions:

$$[\![\mathcal{E}_1 = \mathcal{E}_2]\!]^\beta_G = \begin{cases} Null & \text{if } [\![\mathcal{E}_1]\!]^\beta_G = Null \text{ or } [\![\mathcal{E}_2]\!]^\beta_G = Null \\ True & \text{if } [\![\mathcal{E}_1]\!]^\beta_G = [\![\mathcal{E}_2]\!]^\beta_G \\ False & \text{otherwise} \end{cases} \tag{27}$$

$$[\![\mathcal{E}_1 < \mathcal{E}_2]\!]^\beta_G = \begin{cases} Null & \text{if } [\![\mathcal{E}_1]\!]^\beta_G = Null \text{ or } [\![\mathcal{E}_2]\!]^\beta_G = Null \\ True & \text{if } [\![\mathcal{E}_1]\!]^\beta_G < [\![\mathcal{E}_2]\!]^\beta_G \\ False & \text{otherwise} \end{cases} \tag{28}$$

$$[\![\mathtt{x} : \ell]\!]^\beta_G = \begin{cases} True & \text{if } [\![\mathtt{x}]\!]^\beta_G \in N \cup E \text{ and } \ell \in \mu(\lambda([\![\mathtt{x}]\!]^\beta_G)) \\ False & \text{if } [\![\mathtt{x}]\!]^\beta_G \in N \cup E \text{ and } \ell \notin \mu(\lambda([\![\mathtt{x}]\!]^\beta_G)) \end{cases} \tag{29}$$

Logical operations:

$$[\![\Phi_1 \text{ AND } \Phi_2]\!]^\beta_G = [\![\Phi_1]\!]^\beta_G \wedge [\![\Phi_2]\!]^\beta_G \tag{30}$$

$$[\![\Phi_1 \text{ OR } \Phi_2]\!]^\beta_G = [\![\Phi_1]\!]^\beta_G \vee [\![\Phi_2]\!]^\beta_G \tag{31}$$

$$[\![\text{NOT } \Phi]\!]^\beta_G = \neg [\![\Phi]\!]^\beta_G \tag{32}$$

Set operations:

$$[\![c \text{ ELEMENTOF } \mathtt{y}]\!]^\beta_G = \begin{cases} True & c \in \mu([\![\mathtt{y}]\!]^\beta_G) \\ False & \text{otherwise} \end{cases} \tag{33}$$

$$[\![\mathtt{SUBSETEQ(x,\ y)}]\!]^\beta_G = \begin{cases} True & \mu([\![\mathtt{x}]\!]^\beta_G) \subseteq \mu([\![\mathtt{y}]\!]^\beta_G) \\ False & \text{otherwise} \end{cases} \tag{34}$$

Definition 5.3. *MetaGPML evaluates queries and clauses using working tables, ensuring consistency with GQL standards. A table T comprises bindings with shared domains.*

Finally, we define the semantics of clause $\mathcal{C}$ and query $\mathcal{Q}$ as functions operating on table T for graph G.

Clauses and queries:

$$[\![\text{MATCH } \Pi]\!]_G(T) = \bigcup_{\beta \in T} \{\beta \bowtie \beta' \mid \beta' \in [\![\Pi]\!]_G, \beta \sim \beta'\} \tag{35}$$

$$[\![\text{FILTER } \Phi]\!]_G(T) = \{\beta \in T \mid [\![\Phi]\!]^\beta_G = True\} \tag{36}$$

$$[\![\mathcal{C}\ \mathcal{Q}]\!]_G(T) = [\![\mathcal{Q}]\!]_G([\![\mathcal{C}]\!]_G(T)) \tag{37}$$

$$[\![\text{RETURN } \mathcal{E}_1 \text{ AS } x_1, \ldots, \mathcal{E}_n \text{ AS } x_n]\!]_G(T) = \bigcup_{\beta \in T} \{(x_1 \mapsto [\![\mathcal{E}_1]\!]^\beta_G, \ldots, x_n \mapsto [\![\mathcal{E}_n]\!]^\beta_G)\} \tag{38}$$

The semantics of MetaGPML extend GPML to enable querying meta-property graphs through: matching nodes and edges with labels/properties, handling meta-nodes with subgraphs, and treating property and label objects as first-class citizens. The language supports pattern operations, set predicates, and standard clauses for metadata-aware analytics.

6 FUTURE RESEARCH VISION FOR THE MPG

In this paper, we highlighted the critical need to break down barriers between data and metadata in property graph management. As the first concrete steps towards this vision, we introduced a fully specified data model and query language for meta-property graphs, enabling seamless modeling and interoperation of data and metadata. While this work lays the foundation for flexible property graph management for contemporary applications, further research is needed to fully realize this vision.

(1) Physical implementation and technical challenges. A key challenge is implementing the MPG data model efficiently. Since label sets and properties are treated as data objects with identifiers, they may need dedicated storage and indexing strategies. Alternatively, approaches like concatenated IDs could maintain existing storage structures but may impact query evaluation. Research is needed on physical representations and indexing strategies that optimize MPG performance.

(2) Meta-Property Graphs in practice. It's important to further investigate how Meta-Property Graph can enhance knowledge engineering and management in practice. Understanding how metadata awareness and sub-structure reification can contribute to improving tasks like auditing and human-in-the-loop validation of knowledge graphs or data cleaning, wrangling, integration, and exchange is crucial. Furthermore, studies should be conducted on how the capabilities that MPG introduce, such as subgraph annotation and querying different forms of metadata, can enhance knowledge reasoning and facilitate advanced analysis within knowledge graphs. Additionally, developing effective educational approaches and training resources for students and professionals working with MPGs and MetaGPML requires further study.

(3) Improvements and integration. Meta-Property Graph and MetaGPML can be enhanced through: (1) extending MetaGPML with additional functions to leverage better metadata awareness and also including other currently existing abilities such as paths and repetition which we did not include in this proposed vision of MetaGPML for the sake of simplicity, (2) developing schema and constraint languages for MPG building on PG-SCHEMA [2], and (3) incorporating other forms of metadata such as reflection to expand the metadata awareness in MPG.

REFERENCES

[1] Ghadeer Abuoda, Daniele Dell'Aglio, Arthur Keen, and Katja Hose. 2022. Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches – extended version. In *QuWeDa Workshop on Storing, Querying and Benchmarking Knowledge Graphs.*

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2023. PG-Schema: Schemas for Property Graphs. *Proc. ACM Manag. Data* 1, 2 (2023), 198:1–198:25.

[3] Renzo Angles, Aidan Hogan, Ora Lassila, Carlos Rojas, Daniel Schwabe, Pedro Szekely, and Domagoj Vrgoč. 2022. Multilayer Graphs: A Unified Data Model for Graph Databases. In *Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA).* Association for Computing Machinery, Article 11, 6 pages.

[4] Angela Bonifati, Elaine Qing Chang, Terence Ho, Laks V. S. Lakshmanan, Rachel Pottinger, and Yongik Chung. 2010. Schema mapping and query translation in heterogeneous P2P XML databases. *VLDB J.* 19, 2 (2010), 231–256.

[5] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying graphs.* Morgan & Claypool.

[6] Julian Bruyat, Pierre-Antoine Champin, Lionel Médini, and Frédérique Laforest. 2021. PREC: semantic translation of property graphs. *1st workshop on Squaring the Circles on Graphs* (2021). https://doi.org/arXiv:2110.12996v1

[7] Jan Van den Bussche, Dirk Van Gucht, and Stijn Vansummeren. 2007. A crash course on database queries. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, 143–154.

[8] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. 2022. Graph Pattern Matching in GQL and SQL/PGQ. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*. Association for Computing Machinery, 2246–2258. https://doi.org/10.1145/3514221.3526057

[9] George Fletcher. 2014. On reflection in Linked Data management. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA*. IEEE Computer Society, 269–271.

[10] George Fletcher and Catharine Wyss. 2009. Towards a General Framework for Effective Solutions to the Data Mapping Problem. *Journal on Data Semantics XIV* (2009), 37–73.

[11] Nadime Francis, Amelie Gheerbrant, Paolo Guagliardo, Libkin Leonid, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoč. 2023. A Researcher's Digest of GQL. In *26th International Conference on Database Theory (ICDT 2023)*, Vol. 255. https://doi.org/10.4230/LIPIcs.ICDT.2023.1

[12] Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoc. 2023. GPC: A Pattern Calculus for Property Graphs. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '23)*. Association for Computing Machinery, 241–250. https://doi.org/10.1145/3584372.3588662

[13] Ewout Gelling, George Fletcher, and Michael Schmidt. 2024. Statement Graphs: Unifying the Graph Data Model Landscape. In *Database Systems for Advanced Applications - 29th International Conference, DASFAA 2024*. 364–376.

[14] Olaf Hartig. 2014. Reconciliation of RDF* and Property Graphs. *arXiv:1406.3399v3* (2014). https://doi.org/10.48550/ARXIV.1409.3288

[15] Mauricio A. Hernández, Paolo Papotti, and Wang Chiew Tan. 2008. Data exchange with data-metadata translations. *Proc. VLDB Endow.* 1, 1 (2008), 260–273.

[16] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. 2001. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Transactions on Database Systems (TODS)* 26, 4 (2001), 476–519. https://doi.org/10.1145/503099.503102

[17] O. Lassila, M. Schmidt, B. Bebee, D. Bechberger, W. Broekema, A. Khandelwal, K. Lawrence, R. Sharda, and B. Thompson. 2023. The OneGraph Vision: Challenges of Breaking the Graph Model Lock-In. *Semantic Web Journal* 14 (2023). Issue 1.

[18] Ora Lassila and Ralph R. Swick. 1999. Resource Description Framework (RDF) Model and Syntax Specification. World Wide Web Consortium (W3C) Recommendation. https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[19] Divesh Srivastava and Yannis Velegrakis. 2007. Intensional associations between data and metadata. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 401–412.

[20] Radu Stoica, George Fletcher, and Juan F. Sequeda. 2019. On Directly Mapping Relational Databases to Property Graphs. In *AMW*.

[21] Dominik Tomaszuk, Renzo Angles, and Harsh Thakkar. 2020. PGO: Describing Property Graphs in RDF. *IEEE Access* 8 (2020), 118355–118369. https://doi.org/10.1109/access.2020.3002018

[22] Catharine M. Wyss and Edward L. Robertson. 2005. Relational languages for metadata integration. *ACM Transactions on Database Systems* 30, 2 (2005). https://doi.org/10.1145/1071610.1071618