SHORT-PAPER

PASCAL: A Theory-Informed Visual Interface for Property Graph Schema Visualization

KASIDIS CHANTHATROJWONG, Nanyang Technological University, Singapore City, Singapore

SOURAV S BHOWMICK, Nanyang Technological University, Singapore City, Singapore

BYRON KOON KAU CHOI, Hong Kong Baptist University, Hong Kong, Hong Kong

Open Access Support provided by:

Hong Kong Baptist University

Nanyang Technological University

Published: 22 June 2025

Citation in BibTeX format

SIGMOD/PODS '25: International Conference on Management of Data
June 22 - 27, 2025
Berlin, Germany

Conference Sponsors:
SIGMOD

https://doi.org/10.1145/3722212.3725123
ISBN: 9798400715648

PASCAL: A Theory-Informed Visual Interface for Property Graph Schema Visualization

Kasidis Chanthatrojwong
Nanyang Technological University
Singapore, Singapore
kasidis001@e.ntu.edu.sg

Sourav S. Bhowmick
Nanyang Technological University
Singapore, Singapore
assourav@ntu.edu.sg

Byron Choi
Hong Kong Baptist University
Hong Kong SAR, China
choi@hkbu.edu.hk

Abstract

Property graph schemas are essential for organizing property graph data, serving both prescriptive and descriptive roles. This has led to the recent development of property graph schema languages such as PG-Schema. While understanding of these languages requires familiarity with complex syntax, this poses usability challenges, particularly for domain experts who are not programmers. Current visual abstractions, such as the *labeled schema graph* (LSG), simplify representation but suffers from visual clutter and limited feature support. In this demonstration, we present a novel *theory-informed* visual interface called PASCAL for *multi-modal* visualization of property graph schemas to address these limitations. It realizes a novel visual abstraction called *labeled iconized composite schema* (LICS), whose design is informed by theories and principles from HCI, cognitive psychology, and visualization. Under the hood, it leverages the novel *Map-Paint* paradigm for creating and maintaining the visual components of LICS.

CCS Concepts

• **Human-centered computing** → *Visualization systems and tools*;
• **Information systems** → *Query languages*.

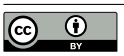
Keywords

Visual interface, property graph schemas, schema visualization, theory-informed design

ACM Reference Format:
Kasidis Chanthatrojwong, Sourav S. Bhowmick, and Byron Choi. 2025. PASCAL: A Theory-Informed Visual Interface for Property Graph Schema Visualization. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725123

1 Introduction

A property graph schema language is crucial for organizing property graph data, which is increasingly used to model relationships between real-world entities. Understanding it, however, requires familiarity with the syntax and semantics of schema languages like PG-Schema [1], which poses a usability challenge since many end users are domain experts but not programmers. Despite this, research on schema languages has mostly concentrated on improving expressiveness rather than usability.



SIGMOD-Companion '25, Berlin, Germany

ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725123

A popular approach to address the usability challenges of schema languages is through visual abstraction. At first glance, it may seem that the popular *labeled schema graph* (LSG) abstraction [1] can serve the visualization need, where node types/labels are depicted using basic shapes (like circles or rectangles), and edge types/labels and inheritance are shown with directed labeled edges. Properties and constraints can be displayed as text within these shapes or accessed on demand.

Despite its simplicity, the LSG abstraction has two significant limitations. First, using text labels for nodes and edges can lead to visual congestion and clutter, undermining the theories and principles of visualization and cognitive psychology (*e.g.,* Gestalt principles [9], visual complexity theories [6]). Additionally, hiding annotations by default limits users' understanding of the property graph schema. Second, the LSG only visually represents a limited subset of schema features, such as node and edge types and inheritance, while other features like constraints, *label expressions*, and the concept of *arbitrary labels/properties* [1] lack corresponding visual encoding. This makes it challenging for users to achieve a comprehensive visualization of a property graph schema.

In this demonstration, we present a novel visual interface for property graph schemas (*e.g.,* PG-Schema) coined PASCAL (Property grAph SChema VisuALizer) to address the aforementioned challenges. It is grounded on a novel *theory-informed*, *extensible* visual abstraction called *labeled iconized composite schema* (LICS) [2] for visually encoding property graph schemas. Specifically, LICS is aimed at enhancing the representation of various schema features such as *node* and *edge types*, *inheritance*, *constraints*, and *label expressions*. Unlike the traditional LSG abstraction, LICS employs composite geometric *shapes* and *icons* to effectively depict these elements, guided by theories and principles from human-computer interaction (HCI), visualization, and cognitive psychology. An example of the LICS abstraction is depicted in Figure 1 (right).

Under the hood, LICS realizes a novel Map-Paint paradigm. At a high level, the *Map Engine* assigns different schema components from a schema language to their respective *shape buckets* based on the visual encoding applied. Following this, the *Paint Engine* "paints" these buckets according to established theories and principles.

2 System Overview

Figure 1 (left) depicts the architecture of PASCAL. It currently supports the PG-Schema [1], a comprehensive and flexible property graph schema language. We now elaborate on the key components in these layers. For details, please refer to [2].

The PASCAL Visual Interface. Figure 1 (right) depicts the PASCAL GUI. It consists of four panels. *Panel 1* is the navigation panel that allows users to load a plaintext property graph schema and switch between two views: *Visualizer View* and *Text View*. The

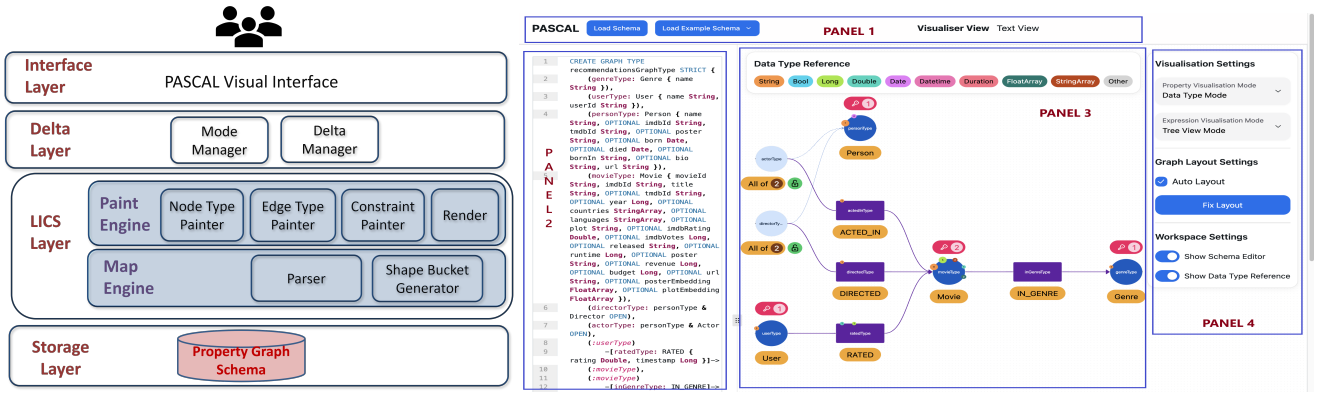


Figure 1: PASCAL: architecture (left); visual query interface (right).

Visualizer View contains the *Schema Editor* and the *Schema Canvas* whereas the *Text View* only involves the former. *Panel 2* is the *Schema Editor* that shows a property graph schema in a specific schema language (*e.g.,* PG-Schema) and allows a user to edit it, if necessary. *Panel 3* is the *Schema Canvas* which visualizes the schema in *Panel 2* using the novel LICS abstraction discussed later. The top of the panel exposes the *Data Type Reference* subpanel. It contains the color codes of different data types used in a property graph schema. *Panel 4* enables a user to set various *modes* for interactive schema visualization.

The LICS Abstraction. Under the hood, *Panel 3* invokes the *Map* and *Paint Engines* to create and maintain the LICS view of the schema. Intuitively, the *labeled iconized composite schema (LICS)* is a directed graph that consists of two types of visual shapes: *elementary* and *derived*. These shapes visually represent key features of a property graph schema. The elementary shapes include *circles*, *arrows*, and *pills*, while the derived shapes consist of *composite circles*, *composite pills*, and *composite arrows*. In a LICS, the nodes represent Node Types, which can be either *base* or *inherited*. These nodes are color-coded circles (blue for base and light blue for inherited). Similarly, the Edge Types—*base* and *inherited*—are illustrated using purple and light purple rectangles, respectively. Specifically, a circle (or rectangle) indicates a Node Type (or Edge Type) without any properties, while a composite circle (or rectangle) represents a Node Type (or Edge Type) that has one or more properties.

The endpoints of Edge Types and inheritances are represented using a solid or dotted, color-coded (purple or blue) *arrow* and a *composite arrow*. The edge source and target are visualized using purple edge at the left and right of Edge Type. A blue dotted straight line represents Node inheritance edges, while a purple dotted Bézier curve signifies Edge inheritance edges. Additionally, a rectangle is overlaid on the arrow of the Edge Type to differentiate it from inheritance edges.

The visual representation of properties for a Node or Edge Type draws inspiration from visualizing predicates in property query graphs as described in [3]. These are depicted as color-coded circles and they can be viewed in two modes: *Data Type* and *Property*[1]. The circles that represent properties are termed *property circles*, while those indicating data types are called *data type circles*. In both modes, the circles are systematically placed along the circumference or perimeter of a Node or Edge Type, ensuring they do not overlap the left and right handles. The color coding of these circles corresponds directly to the colors associated with the properties' data types. Additionally, the area of a data type circle expands in proportion to the number of properties linked to that data type. Each data type circle is labeled with the number of associated properties. For instance, in Figure 2 (left), the STRING data type circle is labeled with 4, indicating that there are four properties of type STRING linked to the PersonType. Its area is also larger than that of other data type circles. Users can hover over a data type circle to see the complete list of properties (Figure 2 (middle)). If a property is not mandatory, an OPTIONAL pill will be shown alongside the color-coded *Data Type* pill.

The label for a Node or Edge Type is visually represented by a yellow pill (an elementary shape) placed near the corresponding (composite) circle or rectangle. Since a property graph schema may have *multiple* labels expressed through label expressions, these labels can be displayed in two modes: *Expression Mode* and *Tree View Mode*. In the *Expression Mode*, the complete label expression is presented textually within the pill, which may be difficult for non-expert users to interpret due to unfamiliarity with label expressions. To address this challenge, multiple labels can also be visualized in the *Tree View Mode*, where the expression is represented as a tree structure along with textual labels (Figure 2 (right)).

A node can optionally be associated with one or more *icons* contained within a pill or circle (*i.e.,* derived shape) to visually represent information related to key constraints, as well as arbitrary labels or properties (*e.g.,* OPEN Label, OPEN Property [1]). A key constraint is indicated by a red pill with a key icon at the top of the Node Type. A green circle with an unlock icon next to the label and below the type name represent arbitrary labels and properties, respectively. Likewise, an Edge Type may feature a red pill with a connect icon to the right of the Node Type to signify a participation constraint. Figure 1 (right) shows examples of icons for the key constraints and arbitrary labels. When users hover their cursor over a key or participation constraint, a pop-up appears, displaying a natural language (NL) description of the constraint.

We now briefly describe the theories and principles that guide the design of LICS. The reader may refer to [2] for details.

[1]In addition, PASCAL supports a *Hidden Mode* that enables a user to hide all properties on-demand.

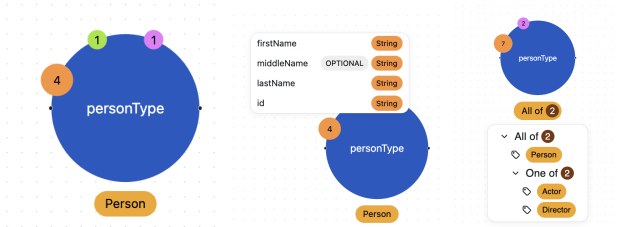


Figure 2: Node Type properties in LICS: (left) *Data Type* mode; (middle) Properties; (right) *Tree View* of label expression.

Theories and principles from cognitive psychology. The design of LICS is informed by *Gestalt principles* [9], *visual complexity theories* [6], and *icon theories* [7, 10]. The *Law of Proximity* is demonstrated by placing labels and constraints near their corresponding Node or Relationship Type. Additionally, data type and property circles are arranged along the edges of the Node Type or Relationship Type, while different data types are color-coded and grouped at the top of the *Schema Canvas* panel. The *Law of Similarity* is evident in the design through consistent use of colors and shapes: Node Types are blue circles, Relationship Types are purple rectangles, and relationship endpoints are purple arrows. Labels are yellow pills, constraints are red pills, and data type and property circles are color-coded circles. Node inheritances appear as blue dotted arrows, relationship inheritances as purple dotted arrows, and arbitrary labels and properties as green circles with unlock icons. The *Law of Closure* is embraced by using circular and rectangular shapes for Node and Relationship Types. The *Law of Continuity* is reflected in the arrangement of data type and property circles along the edges of these types, with Bézier curves connecting arrows. The *Law of Symmetry* is demonstrated by evenly spacing data type and property circles around the circumference and perimeters of the Node Type and Relationship Type. Lastly, the *Law of Common Fate* is adhered as labels, properties, and arbitrary keywords move along with their associated Node or Relationship Types and relationship endpoints and inheritances are all left-to-right by default.

The overall design of the LICS also embraces theories of *visual complexity*[2] by carefully balancing several features. It minimizes the *quantity of information* and the *variety of visual forms* by limiting the number of property and data type circles, as well as color codes, to effectively communicate relevant information. Detailed properties, relationships, and constraints are made available on-demand, while all components of the same type share consistent shapes (circles, rectangles, or pills). Colors are used strategically for visual correlation, focusing on data type rather than semantics to reduce color variety. Size differences indicate the number of properties for specific data types. The layout of property and data type circles is symmetrical to enhance *spatial organization*, and visual congestion is minimized through adequate spacing, detail concealment, and color contrast with the background.

Icon theories [7, 10] are exploited in the design of the unlock, key, and connect icons. They are easily identifiable and represent commonly understood functions, thereby exemplifying the *recognition over recall* principle. *Consistence and standard* is maintained by using three icons consistently in all schemas. *Simplicity and clarity* is realized as these three symbols are not visually complex. Finally, *affordance theory* and semantic understanding are illustrated through the unlock icon, which conveys open, the key icon, which signifies constraints like primary keys, and the connect icon, which represents relationship-related constraints.

Theories and principles from visualization. The LICS abstraction is also based on the criteria of *expressiveness* and *effectiveness* in visualization [5]. Elements such as circles, arrows, icons, and colors precisely convey the necessary information, adhering to the *expressiveness* criterion. The *effectiveness* criterion is met by utilizing color, position, and shapes that are ranked high or medium in perceptual tasks for nominal data. The *Principle of Importance Ordering* is applied by employing the most suitable perceptual tasks to encode more significant information. Specifically, area and nominal values are used to indicate variations in the number of properties represented in data type circles, as higher-ranked perceptual tasks (*e.g.,* length, angle, slope) are not effective for this purpose.

Principles from HCI. Shneiderman's eight golden rules [10] are exploited for the GUI design. In particular, the *strive for consistency rule* is achieved through the uniform representation of all visual components in LICS. The *Schema Editor* presents a list of error messages that indicate the cause of parsing errors, adhering to the *informative feedback* rule. Users of PASCAL take the lead in actions during schema visualization, reflecting the *support internal locus of control* principle. Lastly, the *short-term memory reduction* rule is implemented by including a *Data Type Reference Panel*, which helps users avoid having to memorize the color codes for each data type.

The Map Engine. Intuitively, this module first parses the property graph schema $\mathbb{S}$ in *Panel 2* using the *Parser* submodule. It then maps the Node Types, Edge Types, inheritance, and constraints in $\mathbb{S}$ to their corresponding *shape buckets* through the *Shape Bucket Generator* submodule. Specifically, the shape buckets are associated with the circle, arrow, and icon shapes in the LICS abstraction. Since all Node Types are represented using a circle in LICS, it is mapped to the shape bucket representing circles. Similarly, Edge Type and inheritance, and constraints are stored in shape buckets representing (dotted) arrow and icons, respectively. Importantly, the *Map Engine* does not create extra shape buckets for shapes related to labels (pills) and properties (circle); these are linked to specific Node or Edge Types and are deferred to the *Paint Engine* to generate the elementary and derived shapes.

The Paint Engine. It is responsible for "painting" the LICS abstraction of $\mathbb{S}$ on the PASCAL GUI. It operates on the buckets generated by the *Map Engine* and invokes a set of *LICS operators*–join, up, iconize, position, translate, and delete–to this end, which is then rendered on the GUI by the *Render* submodule. The join operation, adapted from [3], "joins" data type or property circles with Node Type or Edge Type shapes to form a derived shape. This is done by systematically positioning them—guided by Gestalt principles and theories of visual complexity—around the circumference (or perimeter) of the Node Type (or Edge Type). The up operation enlarges data type circles according to the Principle of Importance Ordering and is triggered by the join operator. The iconize operation creates icons for constraints, following icon design theories. The position operation positions shapes in the GUI, adhering to the

[2]Visual complexity of a GUI can be defined as the combination of different features such as *quantity of information, variety of visual form, spatial organization*, and *perceivability of details* [6]

Gestalt principles and visual complexity theories. The translate operation produces NL descriptions of constraints or labels, while the delete operation removes components from the LICS abstraction.

For a Node Type n_i the *Node Type Painter* submodule generates the circle shape and set its name and color. Next, it retrieves the properties associated with n_i and generates the corresponding color-coded data type or property circles depending on the *property mode*. Then, it joins these circles with the circle representing n_i by invoking the join operation, which may invoke the up operation to increase the area of relevant data type circles. Next, it generates the color-coded pill shape of the label/label expression of n_i and positions it near the circle. It exploits the Principle of Common Fate by ensuring that the pill moves along with the circle. Lastly, if n_i is associated with arbitrary label/property (*e.g.,* OPEN keywords) then for each such usage it generates the corresponding icon by invoking the iconize operation that utilizes an *icon map* and position it based on the aforementioned theories and principles. The generation of the visual representations of Edge Types and inheritance is similar and is performed by the *Edge Type Painter* submodule.

For each constraint in the shape bucket for constraints, the *Constraint Painter* submodule determines the associated elements and then determines whether it is associated with a key or participation constraint. If it is a key constraint, the corresponding key icon is selected to create the icon shape using the iconize operation. Otherwise, it is a participation constraint and hence the connect icon shape is created. Next, the constraint is translated into a natural language representation using a rule-based approach by invoking the translate operation and added to the icon as an attribute. Lastly, the visual representation is updated by positioning the icon appropriately by invoking the position operation.

The Mode Manager. *Panel 4* enables users to interactively visualize a LICS in different settings orchestrated by the *Mode Manager*. The *Visualisation Settings* allows one to toggle between the *Data Type*, *Property*, and *Hidden* Modes (the default mode is *Data Type*). The *Expression Visualisation Mode* enables users to choose one of the two modes for displaying labels (default is the *Tree View Mode*). The translate operation is invoked in this mode and it uses a rule-based approach to translate a label expression to its NL-based tree view. The *Graph Layout Settings* allows a user to set the layout of the LICS. By default, the *auto layout* feature is enabled which automatically adjusts the coordinates of various components.

The Delta Manager. The *Mode Manager* sends the changes to the settings initiated by a user to this module for updating the LICS abstraction in *Panel 3*. It invokes the delete operation to remove affected objects (*e.g.,* data type circles) and then executes the relevant part(s) of the *Paint Engine* to recreate and render new visual objects (*e.g.,* property circles) in *Panel 3*. The *Delta Manager* is also responsible for updating the LICS abstraction in response to changes made by users to the plaintext schema in *Panel 2*.

3 Related Systems & Novelty

Theory-informed design is rare in HCI [8]. Recently, SIERRA [4] demonstrated a theory-informed GUI to visualize conjunctive property graph queries, focusing on developing visual encoding schemes for query nodes, links, and predicates. However, property graph schemas have different and more complex characteristics that necessitate effective and user-friendly visualization [2].

4 Demonstration Overview

PASCAL is implemented with React. Our demonstration will be loaded with a few real property graphs schemas (*e.g., Recommendation*, *Northwind*, *Neoflix* from Neo4j) with different levels of complexities. We shall use PG-Schema as the representative property graph schema language for these schemas. A **video** to illustrate the main features of PASCAL using an example is available at https://youtu.be/6iLhEQv_z4U. The key objectives of the demonstration are the followings.

Multi-modal visualization of property graph schemas. One will be able to select a schema using the PASCAL GUI (Figure 1) and visualize it. To do this, they can click the *Load Schema* button in *Panel 1*, upload a schema file, and then click save to display it in *Panel 2*. If the schema is valid, its LICS abstraction will be displayed immediately in *Panel 3* by leveraging the *Map* and *Paint Engines*. In the *Schema Editor* (*Panel 2*), users can add or modify Node Types, Relationship Types, inheritance, and constraints. Any changes made to the PG-Schema will trigger the components in LICS (*Panel 3*) to be re-rendered through the *Delta Manager*. The audience can also select various modes of visualization as described earlier that will trigger the *Mode Manager* and *Delta Manager* to perform relevant changes to the LICS view in *Panel 3*.

Theory-informed design of PASCAL. A key objective of the demonstration is to allow the audience to experience the theory-informed design of PASCAL. By editing a PG-Schema in the *Schema Editor*, participants will see how LICS operators generate and maintain various schema components that incorporate the aforementioned theories and principles. For instance, they will engage in an interactive, hands-on experience that demonstrates how adding or removing properties in the PG-Schema impacts the sizes of data type circles and automatically adjusts their positioning to enhance spatial organization and minimize visual clutter. Participants will also learn how to quickly identify the different data types associated with these circles by consulting the *Data Type Reference Panel*, thereby reducing the load on their limited working memory as they do not need to remember the color codes.

5 Acknowledgment

Sourav S Bhowmick is partially supported by Ministry of Education (MOE) AcRF Tier 1 Grant No. RG19/24.

References

[1] R. Angles, et al. PG-Schema: Schemas for Property Graphs. *Proc. ACM Manag. Data*, 1(2): 198:1-198:25, 2023.
[2] K. Chanthatrojwong, S. S. Bhowmick, B. Choi. LICS: Towards Theory-Informed Effective Visual Abstraction of Property Graph Schemas. *In ACM SIGMOD*, 2025.
[3] J. Ma, et al. Theories and Principles Matter: Towards Visually Appealing and Effective Abstraction of Property Graph Queries. *In SIGMOD*, 2023.
[4] J. Ma, et al. SIERRA: A Counterfactual Thinking-based Visual Interface for Property Graph Query Construction. *In SIGMOD*, 2024.
[5] J. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Trans. on Graphics*, 5(2), 1986.
[6] A. Miniukovich, S. Sulpizio, A. De Angeli. Visual complexity of graphical user interfaces. *In AVI*, 2018.
[7] David Gittins. Icon-Based Human-Computer Interaction. *Int. J. Man Mach. Stud.*, 24(6): 519-543, 1986.
[8] A. Oulasvirta, K. Hornbaek. Counterfactual Thinking: What Theories Do in Design. *Int. Journal of Human-Computer Interaction,* 38(1), 2022.
[9] A. S. Reber. Gestalt psychology. *The Penguin Dictionary of Psychology*, Viking, ISBN 9780670801367, 1985.
[10] B. Shneiderman, C. Plaisant. Designing the user interface: Strategies for effective human-computer interaction. 5th Ed., Addison-Wesley, 2010.