Towards a Systematic Approach to Graph Data Modeling: Scenario-based Design and Experiences

Mengjia ZHAO*, Yan LIU†, Peng ZHOU‡
School of Software Engineering, Tongji University
Shanghai, China
Email: *1434319@tongji.edu.cn, †yanliu.sse@tongji.edu.cn, ‡1435855@tongji.edu.cn

***Abstract*—Graph database is recently being adopted by data analytic systems as an appealing alternative to relational database for the management of large-scale inherent graph-like data. A great challenge of leveraging graph database technologies is to model a problem domain into graph. However, in the absence of considering application requirements or goals, current graph data modeling approaches seem to be invalid. This paper presents an exploration of a systematic approach for graph data modeling—SuMo. Starting from real world scenarios, requirements are transformed into a domain model, which acts as an intermediate model in SuMo, captures modeling features of that domain. SuMo defines a set of rules for the subsequent transformation of this domain model to produce a graph model. We applied SuMo to the modeling of a data-intensive analytic system using real datasets as an illustrating example to clarify our main idea. SuMo is empirically evaluated in terms of query performance, the experimental results indicate promising feasibility and efficiency. The major contribution of our work is a preliminary graph data modeling approach based on scenarios.**

Keywords-graph data modeling; data analytic system; scenario-based modeling; model-driven design

I. INTRODUCTION

Graph database has recently gained popularity rapidly because of a need to effectively manage large-scale inherent graph-like data [1]. Social network, biology, semantic web and health-care are typical application domains that contain such kind of data [2][3][4]. It has been observed that graph database is usually preferable to relational database in managing data of these domains since the latter hardly captures the inherent graph structure [5]. Motivated by scalability and performance needs of current applications, graph database is increasingly adopted by data analytic systems.

The analysis of relationships becomes important when dealing with highly connected data. A graph consists of a finite set of nodes, and a finite set of edges defining relationships between these nodes. In graph database, data is stored as graph and is accessed by queries as graph traversal operations. Compared with the stores of relational database, relationships are treated as first-class citizens in graph database [6], expressed in a more straightforward way. Moreover, queries involving complex and inefficient join operations are transformed into graph traversals. The execution time of graph traversal is proportional only to the size of traversed part [6].

DOI reference number: 10.18293/SEKE2016-119

Graph data modeling is the process in which an arbitrary application domain described as a connected graph of nodes and edges, it is not a context-free process, but a purposive abstraction related with application requirements. As it happens with relational database, the design of relational model can start from ER model, such kind of "springboard" is also conducive to the design of graph model.

Guidelines and principles on graph data modeling can be found from various sources, including books, technical reports, graph database online community and practitioners' blogs. There has also been an involvement of academic papers in the study of graph data modeling, many works focus on converting existing data from relational to graph model automatically; a number of researchers aim at generic graph data modeling approaches. However, most of these works, simply demonstrated some rudimentary strategies of graph data modeling, rather than connecting to practical application requirements. As a result, practitioners can only find some scattered modeling guidelines.

Currently, graph data modeling is still based on best practices and probably unproved guidelines, which are usually relevant to specific systems [5]. This paper aims to propose a systematic graph data modeling approach at a "proof of concept" stage. The primary challenge that come is, how to analyze requirements of application domain to support graph data modeling? Translating these requirements to produce a model brings more challenges.

In this paper, we are exploring a systematic graph data modeling approach, using scenarios to start out the modeling process. We suggest an adapted domain modeling approach with strategies to abstract key concepts from scenarios. A set of rules are defined for the transformation of domain model to graph model. Our aim is to match application requirements and facilitate the design of graph-based analytic systems. We also provide experiments, in terms of query performance, showing the advantages of our approach with respect to a naïve approach. The main contribution of this paper is a sketch of a systematic graph data modeling approach.

The rest of this paper is organized as follows, Section II presents related work. The proposed approach is presented in Section III. A case study is demonstrated in Section IV. We design and conduct a group of experiments to evaluate our approach, as well as discuss the results in Section V. The conclusion and future works are in Section VI.

II. Related Work

Graph database practitioners and enthusiasts have built graph data models in their respective domains [7]. After familiarization of the application domain, they create a basic skeleton of graph model directly and intuitively. Generally, these models need further refinements. Many best practices have emerged in domains of real-time recommendations, fraud detection, social network, etc [8]. Guidelines for graph data modeling can be concluded from these use cases. However, a systematic solution can not be generalized or summarized.

Relational database has been around for many decades and is the choice for most traditional data-intensive storage and retrieval systems. To meet new demands of current applications, a lot of existing systems choose to migrate data layer from relational database to a graph-based storage system. In [9], a methodology was proposed to convert a relational model to a graph model by exploiting the schema and constraints. [10] and [11] focus on mapping relational to graph model without semantically loss, use *Primary Keys* and *Foreign Keys* to create edges between nodes.

A number of researchers have tried to propose generic graph data modeling approaches. In [5], a model-driven methodology was proposed for the design of graph database. It starts from Entity-Relationship (ER) model, translates this conceptual model into graph model following a specific strategy, aims to minimize the number of needed access operations in retrieving data. In our opinion, some purposeful works, like identifying application goals or requirements, are not involved in [5].

Related work so far can not fully support the modeling process of an arbitrary graph-based analytic system from the very start. Our approach tends to concern more about requirements. We will present our work in the following sections to explore a systematic, requirement driven approach for graph data modeling.

III. SuMo: a scenario-based approach for graph data modeling

We name our approach "SuMo" referring to graph data modeling using scenarios. Fig. 1 presents an overview of SuMo. The modeling processes can be organized in the following two phases:

A. Phase 1: Scenario-based domain modeling

The main difference between what we do in this phase and general domain modeling is that, we are "graph-oriented". That is to say, every activity we do is purposeful, we set up a graph in mind at the beginning stage. *Discover* and *Invention*, which are used often in OOD, will abstract domain concepts or attributes at early stage. Specifically, constraints derived from scenarios will check the domain entities and their relationships. In the refinement of domain concepts, *Integration* and *Split-off*, together with the former two strategies might be used.

1) Discovery: Discovery is a basic strategy to recognize the abstractions used by domain experts. If the scenarios mention it a lot, the abstraction is usually important.

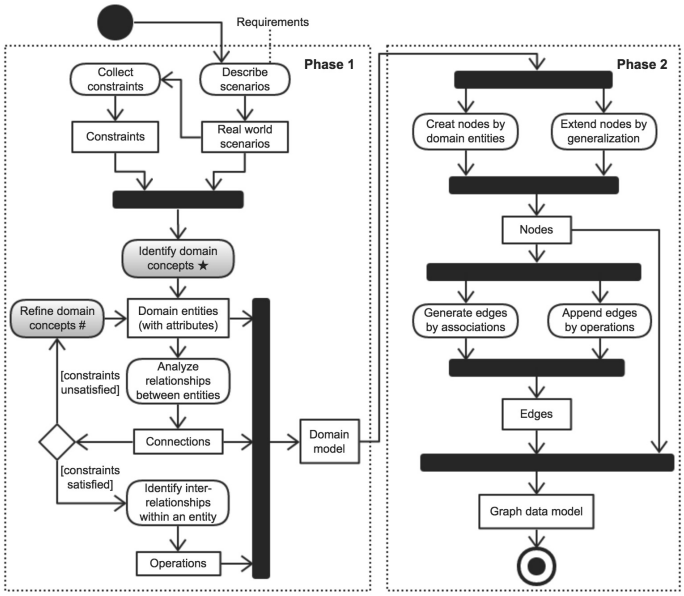

Figure 1. An overview of SuMo

2) Invention: Through invention, we create artifacts (new concepts or attributes) that are useful for the application domain. One of the cases is when dealing with the so-called "n-ary" relationships.

3) Integration: Remember, domain modeling is directed against graph database applications. Integration is used to merge those less important concepts into related ones.

4) Split-off: In contrast, some concepts, which are usually treated as attributes in general domain modeling, might be important "indexes" of that domain. According to scenarios and constraints, we will split off those important attributes to become new entities.

Besides, we add a specific activity in the adapted domain modeling: using operations to identify inter-relationships within a domain entity. These operations define the relationships between each object of a conceptual class, which is important to graph model since each node in graph is an object not a class. By ensuring the correctness of the domain model we are implicitly improving the graph data model.

B. Phase 2: Model transformation

In this phase, domain entities, attributes, connections and operations are mapped into different elements of graph model accordingly. We set up the following rules:

1) Creating nodes by domain classes: Each object of a domain class will be transformed into a node, attributes of this object will be properties of the corresponding node. Fig. 2 presents an example of creating nodes by domain classes.

2) Extending nodes by generalization: Generalization indicates that the subclass is considered to be a specialized form of its superclass. Corresponding node of subclass will be tagged with the type of superclass, the properties of this node will be extended accordingly. Fig. 2 presents an example of extending nodes by generalization.

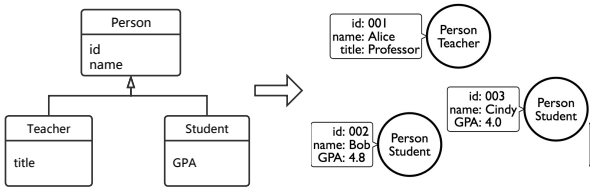

Figure 2. An example of creating and extending nodes

3) Generating edges by associations: An association is a relationship of two (or more) domain classes that describes links between their object instances. Aggregation and composition are specialized forms of association. We will name the relationship semantically in the transformation of aggregation and composition. Importantly, multiplicity of associations will be retained in graph model. Fig. 3 presents an example of generating edges by associations.

4) Appending edges by operations: Operations in SuMo is used to describe relationships between objects of a domain class, will be transformed into edges connecting same type of nodes. These edges help to build a strongly connected graph, reduce the length of paths between same type of nodes and the number of access operations when walking in these nodes. Fig. 4 presents an example of appending edges by operations.

IV. Case study

We applied SuMo to the modeling of a Criminal Network Analysis (CNA) system. In this section, we choose a scenario of "route tracking" as an example. The primary goal of "route tracking" is to support detecting criminal organizations. We will present how SuMo is used to generate a graph model.

A. Phase 1: Scenario-based domain modeling

The original scenario of "route tracking" is adapted to the following one for the reason of confidentiality, all the critical information for modeling is retained.

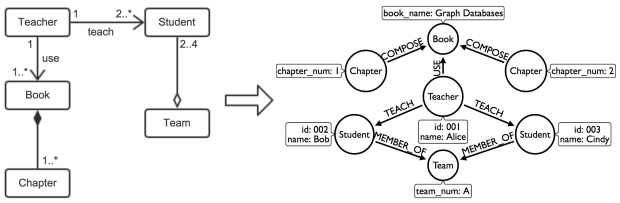

Figure 3. An example of generating edges by associations

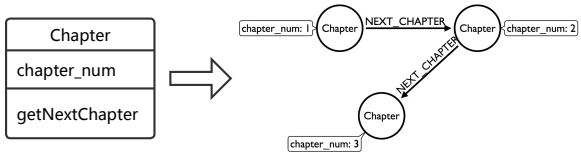

Figure 4. An example of appending edges by operations

Peter and his team are users of our system, they are tracking one of the suspicious people whose name is Bob. In order to find out other people in Bob's criminal organization, Peter tracks Bob's travel route or multi-step route. Based on working experience, Peter supposes that Bob's fellows in crime mainly lie in those people who take same flight/train or have same route with Bob on the same day or within a few days.

Based on the above scenario, we collected the following constraints that should be considered in domain modeling:

- Traveling by flight or by train are homologous.
- One city may have several airports and railway stations, but travel route only concerns about the city.
- Multi-step route indicates that, the departure city of *PersonA's* next travel is the destination city of his or her last travel, such as *"city1-city2, city2-city3"*.
- Date is an important index in route tracking.

To demonstrate the abstraction of domain concepts, let's follow two activities (marked by $\star$ and $\sharp$) in Fig. 1, details of capturing domain concepts using modeling strategies proposed in Section III-A are presented in Table I. The domain model we generated is as Fig. 5 shows. We identified two operations of `Travel`: `getNextTravel` searches the next travel this person took, `getNextTrace` seeks the next travel of this person satisfies that: the departure city of one's next travel is the destination city of his or her last travel.

B. Phase 2: Model transformation

Based on the domain model in Fig. 5, we use the rules defined in Section III-B to generate our graph model, we call it *S-Model* for short. Fig. 6 illustrates an instance of *S-Model*. Some properties of node `Day` are omitted due to the limited space. Every person will have a travel chain in chronological order, multi-step routes are expressed in a certain graph pattern.

TABLE I. Demonstration of how to abstract domain concepts

Activity	Domain concepts/attributes	Strategy used
Marked by $\star$	`Person`	Discovery
	`Flight`	Discovery
	`Train`	Discovery
	`Flight_Travel`	Invention
	`Train_Travel`	Invention
Marked by $\sharp$	`departure_city`	Invention (integrate into `Flight_Travel`, `Train_Travel`)
	`destination_city`	Invention (integrate into `Flight_Travel`, `Train_Travel`)
	`travel_route`	Invention (integrate into `Flight_Travel`, `Train_Travel`)
	`Travel`	Invention
	`Day`	Split-off

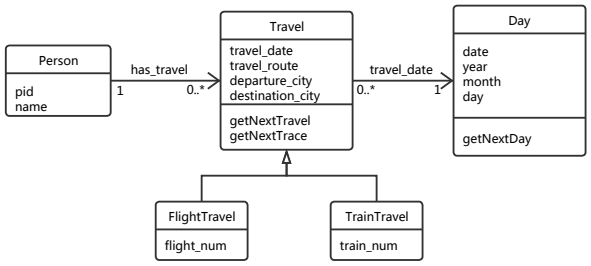

Figure 5. Domain model of route tracking

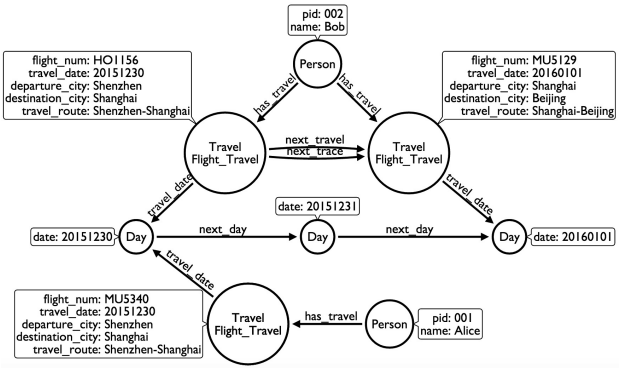

Figure 6. An instance of *S-Model*

For comparison, we build another graph model (*N-Model*) following a naïve approach (tuples are mapped to nodes and *Foreign Keys* are to edges) commonly used in mapping relational model to graph model.

V. Evaluation and Results

We now present the experiments to evaluate SuMo in terms of query performance comparing with the naïve approach. All the experiments were executed in a machine with an Intel Core i5 processor, 2.8 GHz and 8 Gigabytes of RAM memory. We used Neo4j Community Edition 2.3.0 and set JVM heap to 4096K. All the queries are written in Cypher, a declarative graph query language provided by Neo4j. The datasets illustrated in Section IV were used to conduct our experiments, for the reason of confidentiality, part of the real data were allowed to be used (427,127 travel records).

Based on the scenario described in Section IV, we grouped 7 query sets including 4 simple query sets (set1-set4) and 3 rather complex query sets (set5-set7). Each set has 5 different queries that are homogeneous with respect to function and complexity. We chose input randomly, executed each query in all sets 10 times, measured the average execution time.

A. Simple queries

Fig. 7 presents the performance of simple queries (set1-set4). The *S-Model* performs consistently better than the *N-Model* for all of the queries, significantly outperforming in set3 and set4 that concern about `travel_route`. This is due to *S-Model* scans less subgraphs than the *N-Model* which spends more time in mapping airports or train stations to cities.

B. Complex queries

Considering complex queries (set5-set7), we found they can hardly be achieved only by Cypher in *N-Model* because these

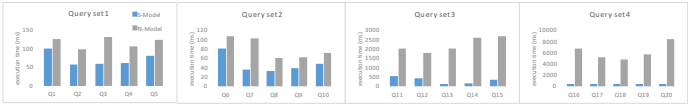

Figure 7. Performance of simple queries

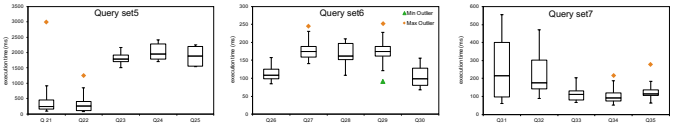

Figure 8. Performance of set5-set7 in *S-Model*

queries seek multi-step routes. Generally, we need some off-line processing to get them done in relational model. *N-Model* is directly mapped from relational model, the difficulty in dealing with these rather complex queries is similar. In *S-Model*, these queries can be accomplished in normal graph traversals. The performance of set5-set7 is shown in Fig. 8. The outliers fall in an acceptable range.

VI. Conclusion and Future Work

In this paper, we presented an exploration of a systematic graph data modeling approach based on scenarios. Our preliminary approach contains an adapted modeling phase involving specific strategies for capturing domain concepts. Rules for converting domain model to graph model are defined in the phase of model transformation. The experimental results obtained are promising regarding SuMo's potential to be used in real development. In future work, we will consider generating properties of edges in the model transformation phase. We also intend to refine the domain modeling phase by providing templates of scenarios.

References

[1] R. Angles, "A comparison of current graph database models," in *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE Computer Society, 2012, pp. 171–177.

[2] C. Cattuto, M. Quaggiotto, A. Panisson, and A. Averbuch, "Time-varying social networks in a graph database: a neo4j use case," in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, p. 11.

[3] A. Gonzalez-Beltran, E. Maguire, P. Georgiou, S.-A. Sansone, and P. Rocca-Serra, "Bio-graphiin: a graph-based, integrative and semantically-enabled repository for life science experimental data," *EMBnet. journal*, vol. 19, no. B, pp. pp–46, 2013.

[4] Y. Park, M. Shankar, B.-H. Park, and J. Ghosh, "Graph databases for large-scale healthcare systems: A framework for efficient data management and data services," in *2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2014, pp. 12–19.

[5] R. De Virgilio, A. Maccioni, and R. Torlone, "Model-driven design of graph databases," *Conceptual Modeling*, pp. 172–185, 2014.

[6] I. Robinson, J. Webber, and E. Eifrem, "Graph databases," 2013.

[7] "Neo4j GraphGist," https://github.com/neo4j-contrib/graphgist/wiki, accessed: Nov. 21, 2015.

[8] "Neo4j Top 7 Use Cases," http://neo4j.com/use-cases/, accessed: Jan. 5, 2016.

[9] R. De Virgilio, A. Maccioni, and R. Torlone, "Converting relational to graph databases," in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, p. 1.

[10] D. W. Wardani and J. Kiing, "Semantic mapping relational to graph model," in *2014 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*. IEEE, 2014, pp. 160–165.

[11] D. W. Wardani and J. Kung, "Semantic mapping relational to a directed property hypergraph model," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*. IEEE, 2015, pp. 152–159.