A generic modelling to capture the temporal evolution in graphs

Landy Andriamampianina[*,**], Franck Ravat[*]
Jiefu Song[*,**], Nathalie Vallés-Parlangeau[*]

[*] IRIT - Université Toulouse 1 Capitole, 2 Rue du Doyen Gabriel Marty
F-31042 Toulouse Cedex 09
{*firstname.lastname*}@ut-capitole.fr,
[**] Activus Group, 1 Chemin du Pigeonnier de la Cépière,
31100 Toulouse
{*firstname.lastname*}@activus-group.fr

Abstract. A temporal graph is based on two aspects: it is a graph that models the interconnection of data and that includes time dimension to model the temporal evolution of data. Temporal graphs are commonly used in different domains to capture the different evolution types that occur in a graph over time. However, existing modelling solutions of temporal graph do not allow to manage all temporal evolution types in the real world. In this paper, we propose a generic model that is able to manage the temporal evolution at different levels: at the schema level to capture the temporal evolution in the graph data structure, and at the instance level to capture the temporal evolution of data contained in entities and their relationships as well as the temporal evolution of the graph topology. We complete our proposed modelling solution with a set of translation rules compatible with the property graph model. The feasibility of our proposal is illustrated through an implementation of our model in the Neo4j database system.

1 Introduction

Graphs have been widely used to model complex relationships among interconnected entities in real applications (Holme and Saramäki, 2012). For example, in social networks, graphs model relationships between users. Graph modelling is not only exploiting data contained in entities and their relationships but also the graph topology [1], which is also a meaningful source of information. For instance, the graph topology allows to discover the friend of a friend in a social network.

It is also relevant to exploit the temporal evolution of a graph. For example, modelling the activity of users in a social network requires to consider the evolution of the network through time (Ahmed et al., 2010). Accordingly, the *temporal graph* model has been created not only to exploit data contained in entities and their relationships as well as graph topology but also

1. Topology is the way in which the nodes and edges are arranged within a graph.

to exploit different types of temporal evolution in the graph: the way that entities, relationships and graph topology change over time (Zaki et al., 2016).

However, the temporal evolution of the graph data structure is not included in existing temporal graph models. We describe the graph data structure as the organization of entities and their relationships into types which consists in the specification of the attributes set of each type. The temporal evolution of the graph data structure is a new source of information. As an example, tracking the temporal evolution of the attributes set of an entity type allows to capture the change in its meaning over time.

In a nutshell, current works on temporal graph model only focus on some temporal evolution types. Yet, some business domains require to study the complete aspect of evolution i.e. the temporal evolution of data contained in entities and their relationships, the temporal evolution of the graph topology and the temporal evolution of the graph data structure. In this case, it is necessary to have a generic model capable to manage these different types of evolution.

In this paper, we therefore address the problem of modelling temporal graph to integrate any temporal evolution types. In response to this problem, we first analyze how current works model the temporal evolution of graphs (Section 2). Second, we demonstrate the interest of capturing different evolution types in a running example (Section 3). Then, we present our contribution consisting in a temporal graph modelling which is generic enough to capture the different evolution types we discuss in the previous paragraph. To do so, we consider different abstraction levels in our modelling: (a) the *schema level* to capture the temporal evolution in the graph data structure, and (b) the *instance level* to capture the temporal evolution in data contained in entities and their relationships as well as the temporal evolution of the graph topology (Section 4). Finally, we study the feasibility of our proposal by implementing our model in the Neo4j database system (Section 5).

2 State of the Art

A graph is a collection of nodes with directed or undirected edges connecting them. Several graph based modelling proposed in the literature combine graph theory and classical data models concepts (entity-relationship model or Unified Modelling Language), through the concepts of *entities* represented by nodes, *relationships* represented by edges, *entity and relationship types* and their *attributes* (Ebert and Franzke, 1994) (Ghrab et al., 2016) (Gómez et al., 2019). Some works consider a schema level (entity and relationships types) and an instance level (entity and relationship) in their modelling but the distinction between the two is not always clear (Angles and Gutierrez, 2008).

According to Vernet et al. (2018), "a temporal graph varies over time and is defined over a time horizon. The nodes and edges can be modified, as well as all other parameters that the graph can have according to a model". Sequence of graph snapshots [2] have been commonly used in literature to model the temporal evolution in a graph (Borgwardt et al., 2006) (Chan et al., 2008) (Fard et al., 2012) (Khurana and Deshpande, 2013). Each snapshot consists of a copy of the entire graph at each time instance of its timeline. This results in a large number of snapshots of little use when many small changes take place frequently (Maduako et al., 2019).

2. denoted $G_1, G_2, ..., G_T$ where G_i is an image of the entire graph at the time i and $[1; T]$ is the lifetime of the graph.

The temporal evolution of a graph refers to the changes on its parameters over time, namely (a) *its topology* (the way nodes are connected by edges) and (b) *data* (data values of node and edge attributes). The temporal evolution of the graph topology is discussed in the literature through two types of evolution: *edge evolution* and *node evolution* (Zaki et al., 2016). *Edge evolution* depicts the addition or deletion of edges at each time instance of the graph lifetime. As an example, consider a road traffic network where each edge represents a road segment with smooth traffic and each node a location. As traffic keeps changing over time, road segments are added or deleted over time (Yang et al., 2014). *Node evolution* describes the addition or deletion of nodes at each time instance of the graph lifetime. For instance, the interactions between people in a social network over time are represented by the addition of a couple of nodes connected by an edge at each time they communicate (Kostakos, 2009).

The temporal evolution of graph data is discussed in the literature through two types of evolution: *edge attributes evolution* and *node attributes evolution* (Zaki et al., 2016). Each evolution type depicts changes through time in the values of the attributes of a node or an edge. For instance, Desmier et al. (2012) analyze data values of node attributes to observe the evolution through time of authors' publication number in a co-authorship network. Zhao et al. (2020) examine data values of edge attributes in a transportation network to observe the changes in the values of bus routes attributes (the departure time, the corresponding travel time, the cost of the ticket, and the distance between stations) over time.

To sum up, the state-of-the-art approaches based on snapshots do not provide a straightforward solution to capture changes in a graph, since they require observing differences between two snapshots of the graph. This implies some drawbacks (Maduako et al., 2019): (i) As time is associated to each graph snapshot, it induces intensive computation requirements and query complexity to retrieve temporal aspects of an individual graph component. (ii) Regarding the memorization of data, it requires high memory on disk because even useless redundant data (i.e. non changing data) are kept.

Regarding evolution types, most temporal graph models focus on capturing one evolution type according to the specific need of their application. Thus, they are not reusable for other application domains. Considering our discussion at the beginning of this section, the distinction between the schema level and instance level is not necessarily formalized in temporal graph models. To the best of our knowledge, *schema evolution* has not been considered among possible temporal evolution of the graph. Schema evolution refers to the creation, deletion and modification of a type of node or edge (Bonifati et al., 2019). As an example, consider a transportation network in which nodes are typed "bus stop" or "taxi stop" and edges, "bus path" or "taxi path". The apparition of Uber among transporation means provides a new type of nodes and edges in the network.

The objective of our work is two-fold. First, we propose to manage the temporal evolution of our model in a different manner than in the snapshots model. We consider the temporal evolution at two levels of graph granularity: at the individual level of a graph component (entity or relationship) and at the global level of the entire graph. While in the snapshots model time is attached at the level of the entire graph, time is attached at each individual graph component in our model. In that way, it is more easy to capture the temporal evolution of an individual graph component. Moreover, we do not need to capture the temporal evolution of the graph by modelling snapshots replication so the data redundancy is reduced. Second, our model integrates two levels of abstraction - schema level and instance level - to capture

different temporal evolution types we discuss. The schema level allows us to capture the temporal evolution of the graph data structure. The instance level allows us to capture the temporal evolution of data as well as the temporal evolution of the graph topology. In that respect, our temporal graph model is generic enough to track temporal evolution at different levels of graph granularity and to support any evolution types.

3 Running example

For a better illustration of our concerns, we consider the problem of tracking information about the evolution of a product during its life cycle as a running example. During its life cycle, a product goes through multiple phases: conception, manufacturing, ..., until its service. Thus, the product goes through different states. The value of the product attributes changes through time. Moreover, the set of the product attributes evolves conforming to the semantics affected to the product by the department involved at each phase (Brunsmann, 2011). Then, product life cycle includes different entities (the product, the product components, department, suppliers etc.) at each phase. Thus, it requires to link data coming from each phase of the life cycle. In conclusion, tracking the evolution of a product during its life cycle involves to track the evolution at the instance level: (a) the temporal evolution of data i.e. the changes in the values of product attributes and (b) the temporal evolution of the linked data topology i.e. which data is added or deleted at each phase. Moreover, it includes the evolution of linked data at the schema level, namely, the addition or removal of new entity types at each phase as well as the addition or removal of attributes of the product through time.

4 Proposition

We propose a generic modelling of temporal graph for two purposes. First, we want a model capable to manage the temporal evolution at a finer granularity than existing snapshot-based approaches i.e. at the entity or relationship level. For that purpose, we assume that each entity or relationship can evolve independently from the evolution of the entire graph (Definitions 4.8, 4.9 and 4.10).

Second, we want a model capable to track different types of temporal evolution to be easily adaptable to application domains, especially the ones that require all temporal evolution aspects. For that purpose, we distinguish two abstraction levels in our modelling: the schema level and the instance level. The *schema level* describes the data structure of the graph which we denote *schema*. A schema describes the organization of entity and relationship types in the graph (Definitions 4.2, 4.3, 4.4 and 4.5). The schema level of our model allows to capture the temporal evolution of the graph schema:

1. the addition or deletion of entity and relationship types over time;
2. the modification of the set of attributes of an entity or relationship type over time, i.e. the addition or deletion of an attribute in the attributes set.

The instance level describes the instantiation of our schema, that is entities and their relationships (Definitions 4.6 and 4.7). The instance level of our model allows to capture the temporal evolution of both data contained in entities and their relationships and graph topology:

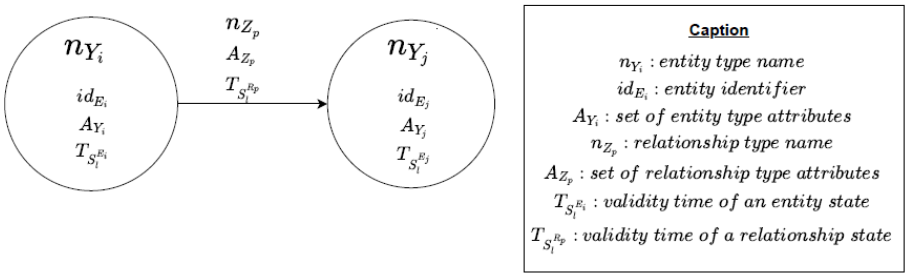
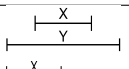
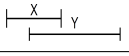


FIG. 1 – *Our temporal graph notation. A node represents an entity state. An edge represents a relationship state.*

Symbol	Interpretation	Illustration
$X\ d\ Y$	X during Y	
$X \circ Y$	X overlaps Y	

TAB. 1 – *ALLEN operators (Allen, 1983).*

3. the addition or deletion of entities and relationships over time;
4. the change in the values of an entity or relationship attributes over time.

We first define formally what a temporal graph is and its structure (Definitions 4.1, 4.2, 4.3, 4.4, 4.6 and 4.7) before the presentation of the temporal evolution management (Definitions 4.8, 4.9 and 4.10).

Definition 4.1. A *Temporal Graph* represents the entire content of the graph in a given lifetime $T = [t_0; t_N]$. It is denoted as $TG = \{S^E; S^R\}$ where S^E is the set of all entity states and S^R the set of all relationship states through T. The Figure 1 presents our notation of a temporal graph.

We define the time units we use. Time can be seen as a set of time grains endowed with an order relation. It can be represented by a straight line on which each grain appears as a point (Canavaggio, 1997). Accordingly, we use two time units in our model: (a) the *instant* which is a particular point on the timeline and (b) the *time interval* which is a segment on the timeline delimited by two specific instants. To describe the possible relations between time intervals, we use two operators from the ALLEN temporal logic: d and $\circ$ (Table 1).

The schema level of our model describes the data structure of the graph in the following definitions.

Definition 4.2. A *schema* describes the organization of entity and relationship types in a graph. It is denoted as $SC = \{Y, Z\}$ where $Y = \{Y_1, ..., Y_p\}$ is a finite set of entity types represented by nodes and $Z = \{Z_1, ..., Z_n\}$ is a finite set of relationship types represented by edges. An illustration of a schema is given in Figure 2.

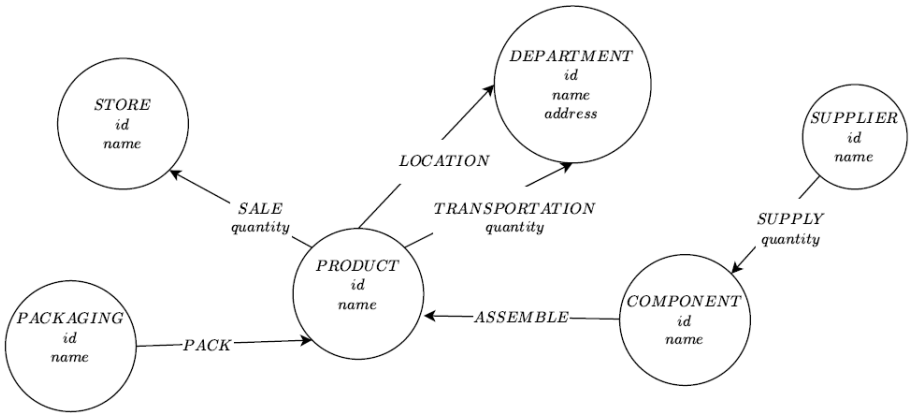


FIG. 2 – *Illustration of a schema of a product life cycle.*

Definition 4.3. An *entity type* describes a collection of entities having the same set of attributes. It is denoted $Y_i = \langle n_{Y_i}, A_{Y_i} \rangle$ where $Y_i \in Y$, n_{Y_i} is the name of the entity type and $A_{Y_i} = \{a_1^{Y_i}; ...; a_n^{Y_i}\}$ its set of attributes.

Definition 4.4. A *relationship type* describes a collection of relationships having the same set of attributes. It is denoted $Z_i = \langle n_{Z_i}, (Y_j, Y_p), A_{Z_i} \rangle$ where $Z_i \in Z$, n_{Z_i} is the name of the relationship type, Y_j is the source entity type of the relationship type, Y_p is the destination entity type of the relationship type and $A_{Z_i} = \{a_1^{Z_i}; ...; a_n^{Z_i}\}$ the attributes set of the relationship type.

Definition 4.5. Each entity or relationship *attribute* is denoted $a_i = \langle n_{a_i}, v_{a_i} \rangle$ where $a_i \in A_{Y_j}$ or $a_i \in A_{Z_j}$, n_{a_i} is the name of the attribute and v_{a_i} is its value.

Example 4.1. *Let's take our running example in Section 3 considering the life cycle of a bike. The Figure 2 illustrates how the different entity types of the bike life cycle are linked at a schema level regardless of the temporal evolution. However, it gives us an idea about the temporal information that we could extract for tracking the temporal evolution of the product: the change in the product attributes value through its life cycle, the delivery date of the product components by suppliers, the duration of the product components assembly, the duration of the product transportation from a department to another, the duration of the product packaging, the duration of the stay of the product in a department, and the sale date of the product to a store.*

The instance level of our model describes instantiated entities and relationships according to the schema level in the following definitions.

Definition 4.6. E is the set of instantiated entities according to a schema SC. Each *entity* $E_i \in E$ is an instance of an entity type $Y_i \in Y$. To identify an entity among the other entities of the same type, it has an identifier id_{E_i}. An entity is denoted $E_i = \langle id_{E_i}, n_{Y_i}, A_{Y_i} \rangle$.

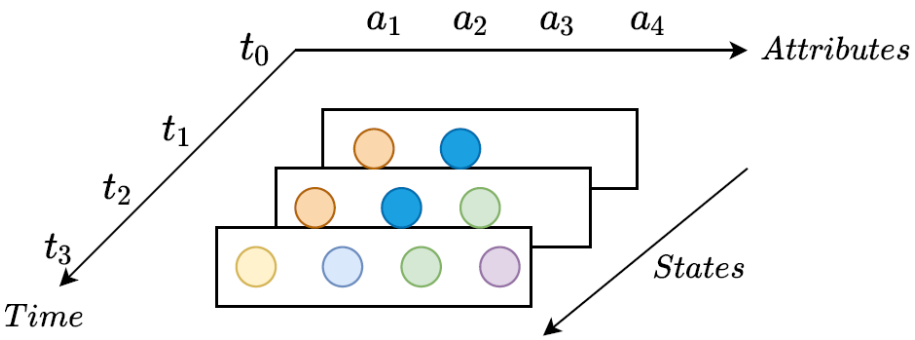


FIG. 3 – *Illustration of the states of an entity during a time window* $[t_0; t_3]$. *Each rectangle illustrates a state at a time* t_i. *Here we consider the validity time of each state as an instant. Each attribute of an entity is illustrated by a circle with an unique color. The attributes are colored as follows:* a_1 *is in orange,* a_2 *is in blue,* a_3 *is in green and* a_4 *is in purple. The change in the attributes value is illustrated by the lightening of the attribute color. For instance, the value of* a_1 *and* a_2 *changes between* t_2 *and* t_3. *The attributes set of an entity can also vary over time. This is illustrated in this Figure by the addition of the attribute* a_3 *between the states at* t_1 *and at* t_2 *and the addition of the attribute* a_4 *between the states at* t_2 *and at* t_3.

Definition 4.7. R is the set of instantiated relationships according to a schema SC. Each *relationship* $R_i \in R$ connecting a pair of entities (E_s, E_p) is an instance of a relationship type $Z_i \in Z$. It is denoted $R = \langle n_{Z_i}, (E_s, E_p), A_{Z_i} \rangle$.

As we discussed at the beginning of this section, we manage the temporal evolution in our model at the level of an entity or relationship. The mechanism of the temporal evolution is based on the concept of states of an entity or relationship.

Definition 4.8. A *state* of an entity or a relationship is defined by the stability of the set of attributes and their values during an interval of time or an instant called the *validity time* and denoted $T_S \; d \; T$. By default, we consider the validity time as a time interval.

We choose to distinguish an entity or a relationship from its states as illustrated in Figure 3. At each change that occurs on an entity or relationship - i.e. the addition or deletion of an attribute (referring to the evolution type 2 we state at the beginning of this section) or the change in the values of an attribute (referring to the evolution type 4 we state at the beginning of this section) - a new state of the entity or of the relationship is created. Naturally, as a state is the lowest level of the graph granularity, it also carries the evolution type 1 and evolution type 3 defined at the beginning of this section. Therefore, through its lifetime, an entity or relationship can go through different states. The model enables to view the different states of an entity or relationship in a time window (Example 4.2).

Definition 4.9. $S^{E_i} = \{S_1^{E_i}, ..., S_p^{E_i}\}$ is the set of states of an entity E_i during T. A state of an entity E_i is denoted $S_j^{E_i} = \langle id_{E_i}, A_{Y_p}, T_{S_j^{E_i}} \rangle$ where id_{E_i} is the identifier of the entity, A_{Y_p}

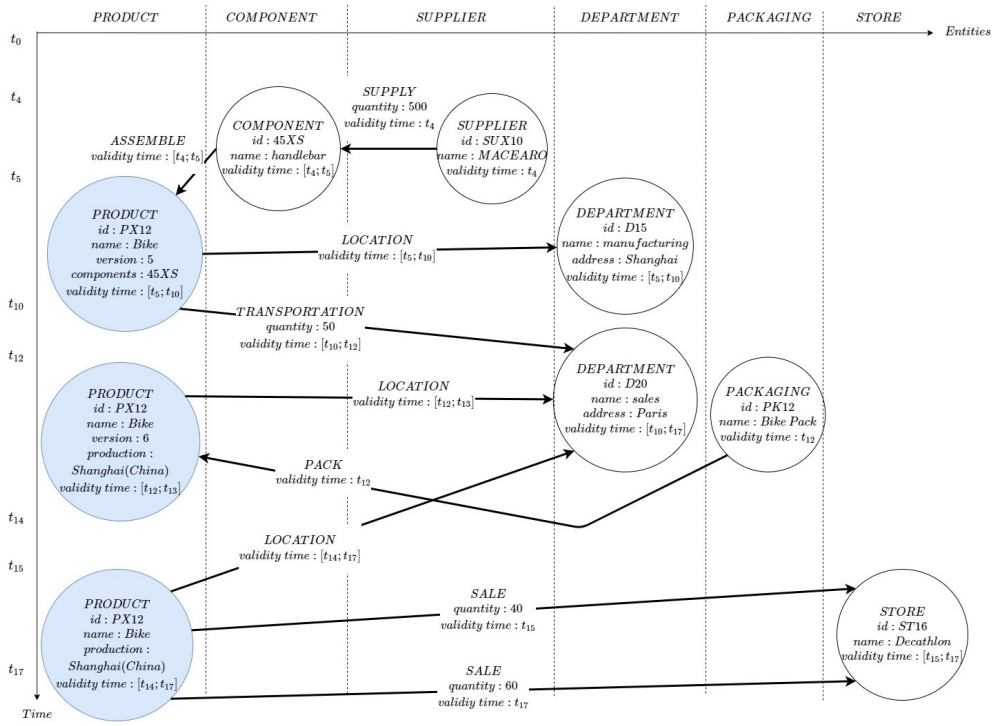


FIG. 4 – *Illustration of the temporal graph representing the life cycle of a bike in a time window* $[t_4; t_{17}]$. *Time is on the y-axis. Each entity type is on a lane of the x-axis. Nodes in blue represent the different states by which the product identified by the name "Bike" goes through the time window.*

is the set of attributes and their associated values during the validity time $T_{S_j^{E_i}}\ d\ T$ in which they are stable.

Definition 4.10. $S^{R_i} = \{S_1^{R_i}, ..., S_p^{R_i}\}$ is the set of states of a relationship R_i during T. A state of a relationship R_i is denoted $S_j^{R_i} = \langle (S_k^{E_p}, S_l^{E_s}), A_{Z_n}, T_{S_j^{R_i}} \rangle$ where $(S_k^{E_p}, S_l^{E_s})$ is a pair of connected entity states, A_{Z_n} is the set of attributes and their associated values during a validity time $T_{S_j^{R_i}}\ d\ T$ in which they are stable. In addition, a state $S_j^{R_i}$ respects the following constraints:

- $S_k^{E_p} \in S^{E_p}$;
- $S_l^{E_s} \in S^{E_s}$;
- $I = T_{S_k^{E_p}} \circ T_{S_l^{E_s}}$ is the time intersection between the validity time $T_{S_k^{E_p}}$ of $S_k^{E_p}$, and the validity time $T_{S_l^{E_s}}$ of $S_l^{E_s}$;
- the validity time $T_{S_j^{R_i}}$ of the relationship state $S_j^{R_i}$ is included in the interval of time I, that is $T_{S_j^{R_i}}\ d\ I$.

Our model concepts	Property graph concepts
an entity state $S_j^{E_i}$	a node
a relationship state $S_j^{R_i}$	an edge
an entity type Y_i	a label tagged on a node
a relationship type Z_i	a label tagged on an edge
an entity identifier id_{E_i}	a property
a set of entity type attributes A_{Y_i}	a set of properties
a set of relationship type attributes A_{Z_i}	a set of properties
a validity time of an entity state $T_{S_j^{E_i}}$	two properties ($start$ and $end\ validity\ time$)
a validity time of a relationship state $T_{S_j^{R_i}}$	two properties ($start$ and $end\ validity\ time$)

TAB. 2 – *Translation rules of our model into the property graph model.*

Example 4.2. *To track the temporal evolution of the bike life cycle, our model allows to select the states of an entity or relationship in a time window. The Figure 4 illustrates a subgraph of the temporal graph of the bike life cycle, i.e. the states of entities and relationships during a time window $W = [t_4; t_{17}]$. We can see that the $PRODUCT$ entity identified by the name "Bike" has three states during W. Moreover, the $SALE$ relationship has two states during W. The other entities and relationships have only one state during W. We can track the different temporal evolution types we discuss:*

- *The evolution type 1 and the evolution type 3 (defined at the beginning of this section) are correlated. The entity type $COMPONENT$ is only added during the interval of time $[t_4; t_5]$ as one of its instantiated entity is only added during this interval of time;*
- *The evolution type 2 (defined at the beginning of this section) is represented by:*
  (a) the deletion of the attribute "$components$" on the $PRODUCT$ entity "Bike" between t_{10} and t_{12}. Indeed, the $PRODUCT$ entity "Bike" arrives at the sales department at t_{12}. The sales department does not need information about the components of the "Bike" hence its deletion;
  (b) the addition of the attribute "$production$" on the $PRODUCT$ entity "Bike" between t_{10} and t_{12}. The sales department must specify information about the production place of the product "Bike" before its sale hence its addition;
- *The evolution type 4 (defined at the beginning of this section) is illustrated by:*
  (a) the change in the value of the attribute "$version$" of the $PRODUCT$ entity "Bike" between t_{10} and t_{12};
  (b) the change in the value of the attribute "$quantity$" of the relationship $SALE$ between t_{15} and t_{17}.

5 Temporal graph implementation

This section illustrates the feasibility of the implementation of our proposed modelling solution with a specific data store. To do so, we first propose a set of translation rules to convert our proposed generic modelling of temporal graph into a property graph model. Then,

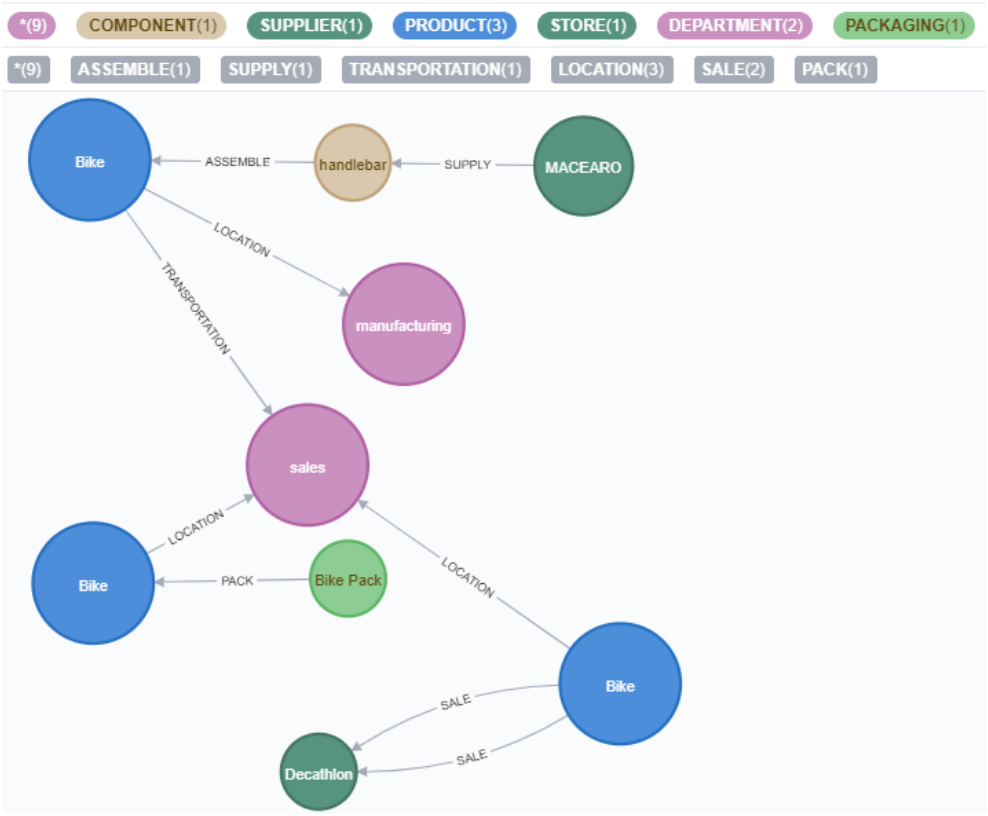


FIG. 5 – *The temporal graph with a lifetime* $T = [t_4; t_{17}]$. *Nodes in blue represent the different states of the product "Bike".*

we implement our running example in a graph database system based on the translation rules. Finally, we query the implemented temporal graph to validate that it is operational.

The property graph model is defined as a directed, labeled, attributed, multi-graph. The edges are directed, nodes and edges are labeled, nodes and edges have associated key/value pair attributes (i.e. properties), and there can be multiple edges between any two nodes (Rodriguez and Neubauer, 2010). From a data modeling point of view, a node represents an entity, an edge represents a relationship between entities, and a property represents a specific feature of an entity or relationship (Angles, 2018). Therefore, we propose the translation rules in Table 2 to convert our proposed generic modelling of temporal graph into a property graph data model.

The property graph model is widely supported by graph database systems such as Nep-

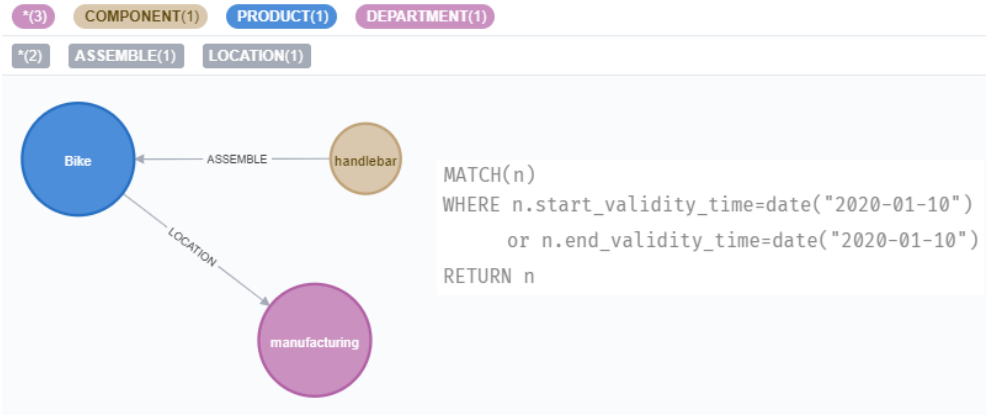


FIG. 6 – *A snapshot of the temporal graph at t_5. t_5 refers to the 10^{th} January 2020. The query language of Neo4j is Cypher. The date format in the Neo4j database system is year-month-day.*

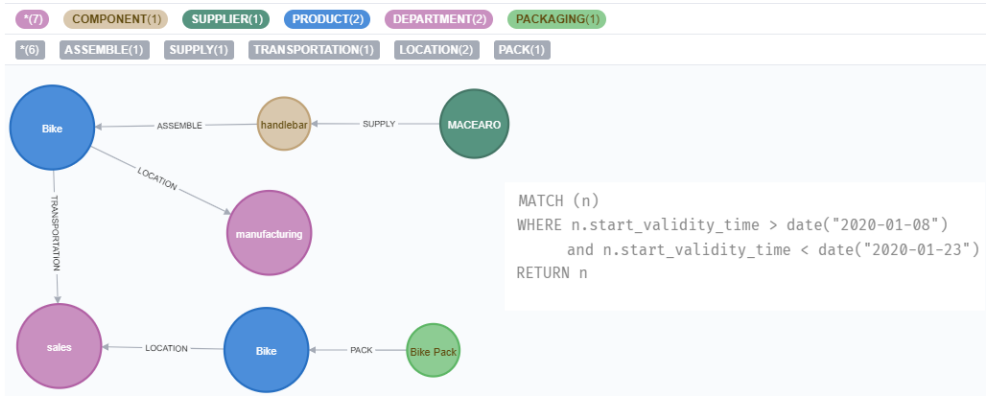


FIG. 7 – *Selection of the temporal graph states at the time-window $[t_4; t_{12}]$. t_4 refers to the 9^{th} January 2020 and t_{12} refers to the 22^{nd} January 2020. Nodes in blue represent the different states of the product "Bike". The query language of Neo4j is Cypher. The date format in the Neo4j database system is year-month-day.*

tune [3], CosmoDB [4], Neo4j [5] or OrientDB [6] (Angles, 2018). We choose to implement our model in the Neo4j database system. Besides, our model is easily implementable in Neo4j thanks to the translation rules we define.

We implement the entire temporal graph illustrated in Figure 4 in Neo4j. We have assigned

3. https://aws.amazon.com/neptune/
4. https://docs.microsoft.com/en-us/azure/ cosmos-db/introduction
5. https://neo4j.com/developer/graph-database/
6. https://orientdb.com/why-orientdb/

real time values to each state validity time. In our case, a time t_i corresponds to a day. The result of our implementation is presented in Figure 5.

To verify that the temporal graph is operational, we first query the graph on the instant t_5. This results of the selection of the states having a validity time including t_5 as illustrated in Figure 6. We can only observe one state of the product "Bike". The result of this query is the equivalent of a snapshot.

Then, we query the graph on the time window $W = [t_4; t_{12}]$. This results on the selection of the states with a validity time overlapping W as illustrated in Figure 7. We observe two states of the product "Bike". Therefore, it confirms that our model manage easily the individual evolution of entities and relationships.

6 Conclusion

Modelling the temporal evolution in a graph requires to adopt a strategic approach. On the one hand, it requires to include the temporal evolution type that the application wants to capture. On the other hand, it requires to include a mechanism to manage the temporal evolution.

In this paper, we have answered these two challenges. First, our model has the advantage to capture different temporal evolution types as it relies on different abstraction levels. The second advantage of our model is that it proposes a mechanism to manage the temporal evolution which is alternative to existing snapshots based models. Instead of replicating the whole graph, our model only replicates the changing entities and their relationships through the concept of states. This results in an easier evolution tracking of individual graph components. The third advantage of our model is that it is easy to implement. It does not require specific developments to be translated into a graph database model.

Our future research directions concern first the evaluation of the performance of our proposal implementation in different graph database systems and then, the temporal graph manipulation. Regarding the performance of the implementation of our proposal, our main concern deals with scalability. We plan to test the scalability of our proposal according to the proportion of evolution in a dataset. Regarding the temporal graph manipulation, our objective is to define a high-level language for querying our temporal graph combining different principles: extension and adaptation to our context of (i) database query languages based on versions (ii) specific operators of temporal graphs and (iii) inferences on graphs.

References

Ahmed, N. K., F. Berchmans, J. Neville, and R. Kompella (2010). Time-based sampling of social network activity graphs. In *Proceedings of the eighth workshop on mining and learning with graphs*, pp. 1–9.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 832–843.

Angles, R. (2018). The Property Graph Database Model. In *AMW*.

Angles, R. and C. Gutierrez (2008). Survey of graph database models. *ACM Computing Surveys (CSUR) 40*, 1–39.

Bonifati, A., P. Furniss, A. Green, R. Harmer, E. Oshurko, and H. Voigt (2019). Schema validation and evolution for graph databases. In *International Conference on Conceptual Modeling*, pp. 448–456. Springer.

Borgwardt, K., H.-p. Kriegel, and P. Wackersreuther (2006). Pattern Mining in Frequent Dynamic Subgraphs. In *Sixth International Conference on Data Mining (ICDM'06)*, Hong Kong, China, pp. 818–822. IEEE.

Brunsmann, J. (2011). Semantic Exploration of Archived Product Lifecycle Metadata under Schema and Instance Evolution. In *SDA*, pp. 37–47. Citeseer.

Canavaggio, J.-F. (1997). *TEMPOS : an historical model for a temporal DBMS*. Theses, Université Joseph-Fourier - Grenoble I.

Chan, J., J. Bailey, and C. Leckie (2008). Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems 16*, 53–96.

Desmier, E., M. Plantevit, C. Robardet, and J.-F. Boulicaut (2012). Cohesive co-evolution patterns in dynamic attributed graphs. In *International Conference on Discovery Science*, pp. 110–124. Springer.

Ebert, J. and A. Franzke (1994). A declarative approach to graph based modeling. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 38–50. Springer.

Fard, A., A. Abdolrashidi, L. Ramaswamy, and J. Miller (2012). Towards Efficient Query Processing on Massive Time-Evolving Graphs. In *Proceedings of the 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE.

Ghrab, A., O. Romero, S. Skhiri, A. Vaisman, and E. Zimányi (2016). Grad: On graph database modeling. *CoRR abs/1602.00503*.

Gómez, L., B. Kuijpers, and A. Vaisman (2019). Online Analytical Processsing on Graph Data. *arXiv:1909.01216 [cs]*.

Holme, P. and J. Saramäki (2012). Temporal networks. *Physics reports 519*(3), 97–125.

Khurana, U. and A. Deshpande (2013). Efficient snapshot retrieval over historical graph data. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 997–1008. IEEE.

Kostakos, V. (2009). Temporal graphs. *Physica A: Statistical Mechanics and its Applications 388*(6), 1007–1023.

Maduako, I., M. Wachowicz, and T. Hanson (2019). STVG: an evolutionary graph framework for analyzing fast-evolving networks. *Journal of Big Data 6*, 55.

Rodriguez, M. A. and P. Neubauer (2010). Constructions from Dots and Lines. *CoRR abs/1006.2361*.

Vernet, M., Y. Pigné, and E. Sanlaville (2018). Algorithmique dans les graphes dynamiques : le cas des flots. In *ROADEF 2018*.

Yang, Y., J. X. Yu, H. Gao, J. Pei, and J. Li (2014). Mining most frequently changing component in evolving graphs. *World Wide Web 17*(3), 351–376.

Zaki, A., M. Attia, D. Hegazy, and S. Amin (2016). Comprehensive Survey on Dynamic Graph Models. *International Journal of Advanced Computer Science and Applications 7*, 573–582.

Zhao, A., G. Liu, B. Zheng, Y. Zhao, and K. Zheng (2020). Temporal paths discovery with multiple constraints in attributed dynamic graphs. *World Wide Web 23*, 313–336.

Résumé

Un graphe temporel est basé sur deux aspects: c'est un graphe qui modélise l'interconnexion des données et qui inclut une dimension temporelle pour modéliser l'évolution temporelle des données. Les graphes temporels sont couramment utilisés dans différents domaines pour capturer les différents types d'évolution qui se produisent dans un graphe au fil du temps. Cependant, les solutions de modélisation du graphe temporel existantes ne permettent pas de gérer tous les types d'évolution temporelle dans le monde réel. Dans cet article, nous proposons un modèle générique capable de gérer l'évolution temporelle à différents niveaux: au niveau du schéma pour capturer l'évolution temporelle dans la structure de données du graphe, et au niveau d'instance pour capturer l'évolution temporelle des données contenues dans les entités et leurs relations ainsi que l'évolution temporelle de la topologie du graphe. Nous complétons notre proposition de solution de modélisation avec un ensemble de règles de traduction compatibles avec le modèle de graphe de propriétés. La faisabilité de notre proposition est illustrée par une implémentation de notre modèle dans le système de base de données Neo4j.