Fundamentals of Generalized and Extended Graph-Based Structural Modeling

Marcin Jodłowiec(✉), Marek Krótkiewicz, and Piotr Zabawa

Department of Applied Informatics, Wrocław University of Science and Technology, Wybrzeże Stanisława Wyspiańskiego 27, 50-370 Wrocław, Poland
{marcin.jodlowiec,marek.krotkiewicz,piotr.zabawa}@pwr.edu.pl

Abstract. The subject of the paper is connected to defining data structures, which are or can be used in metamodeling and modeling disciplines. A new and general notion of Extended Graph Generalization has been introduced. This notion enables to represent arbitrarily complex such the structures. A way of introducing constraints, which allows to reduce this general form to any well known structure has been introduced as well. As the result of the extension and generalization mechanisms applied to the original graph definition any form of graph generalization exceeding well-known structures can be defined. Moreover, the way of associating any form of data to each such structure has been defined. Notions introduced in the paper are intended to be used while defining novel family of metamodels.

Keywords: Graph · Hypergraph · Multigraph · Data modeling · Graph generalization · Graph extension · Metamodeling

1 Introduction

The paper is dedicated to the subject of defining abstract structures in the context of data modeling. These structures are or may be apllied in the disciplines of data modeling, knowledge representation as well as defining modeling languages and models in these languages. The last area is applied in software engineering and can be used for the model-driven automated generating of software systems [9]. In the software engineering domain there are some commonly known standards, which are continuously developed by the Object Management Group (OMG), like the Meta-Object Facility (MOF), Unified Modeling Language (UML) and the remaining standards, which support the approach to the software development processes contained in the concept of the Model-Driven Architecture (MDA) [11]. The approaches and standards mentioned above are based on the graph notion in its basic version, which limits the full use of the complete potential of the modeling as long as the basic data structures are used.

A graph is composed of the two fundamental semantic categories: Vertex – a connected element and Edge – a connecting element. The Vertex type elements

N. T. Nguyen et al. (Eds.): ICCCI 2020, LNAI 12496, pp. 27–41, 2020.
https://doi.org/10.1007/978-3-030-63007-2_3

have the only ability to be connected, while the Edge type elements have the only ability to connect the Vertex type elements. In the case of the classical, primal graph an Edge may connect exactly two Vertex type elements. This concept has been developed and reached many solutions. There are many concepts of structural world representations, which are known from the scientific literature like graphs and hypergraphs [3,13] and scientific research dedicated to the more general structures, like ubergraphs [6] is carried out. These structures are needed for modeling phenomena not only in the field of information systems but also in many different areas (see e.g. [10]). This research forms a deep groundwork for elaborating tools and standards used for data modeling [4]. The structures, which are more general than graphs are also applicable for the problems connected to data modeling like e.g. constructing intermediate models used for the transformations of data models expressed in different metamodels [2].

In contrast to the approaches, which just adapt existing formalisms or extend them, in this work it was decided to make a systematization of the extensions and generalizations needed at modeling through the proposing a new approach to the graph-based modeling. As the result of observations made when defining own metamodels the authors came into conclusion that the graph notion is too simple (in the semantic sense) and it should be generalized to reduce some important limitations as well as extend to some extra elements, which allow for enriching the graph semantics. The graph notion is however a very good starting point for developing generalizations and extensions mentioned above. An extended and generalized structure, which is common for many concepts cited above is presented in the paper. It may form a basis for constructing arbitrary data structures applicable in the computer science. A decomposition of the identified features of the proposed structure into generalizations and extensions is also introduced in the paper. The introduced concept assumes a far-reaching flexibiliy with regard to configuring meta-structures through the flexible choice of particular generalizations and extensions. With this approach it is possible to create the whole family of meta-structures adapted to the specific needs. The defined solution covers also the known concepts like *graph*, *mutligraph*, *hypergraph*, *ubergraph* reducing them to the one, coherent, complementary, universal form, which in turn has important additional features.

The required general graph structure named *Extended Graph Generalization* (EGG) is introduced and defined in the paper, first with the help of a set theory based formal definition contrasted with analogical graph definition and then - with the abstract syntax expressed in the UML together with the constraints specified in both natural language and in the Object Constraint Language (OCL). The semantics of abstract syntax elements is also specified in natural language. Moreover, the concrete syntax is introduced to allow defining EGG in a graphical form. The EGG application is presented on the example of interrelationships in a social network.

2 Formal Definitions

The following section shows formal definition of a graph and extended and generalized *Extended Graph Generalization* structure proposed hereby. A graph has been defined in a standard way, focusing on its two fundamental sets: Vertex, Edge.

Definition 1. *A graph G is an ordered pair of a set V of vertices and a set E of edges. It is written as:*

$$G = (V, E), \tag{1}$$

where

$$|V| = n,\ n \in \mathbb{N}^+,\ |E| = m,\ m \in \mathbb{N}. \tag{2}$$

Each edge $e \in \mathbb{N}$

$$e = \{v_a, v_b\},\ |e| = 2 \tag{3}$$

is such the multiset of vertices

$$v_a, v_b \in V,\ 1 \leq a, b \leq n. \tag{4}$$

Analogously to *graph*, a *multi-graph* is defined as an ordered pair of a set V of vertices and a *multiset* E of edges.

The EGG structure comprise all the extensions and generalizations proposed for a Graph. The EGG definition has been supplemented by a unique identifier *id*. It is essential for distinguishing Vertex, Edge and EGG instances.

Definition 2. *An Extended Graph Generalization EGG is an id, tuple of a set V of vertices, a set E of edges, set of nested Extended Graph Generalizations EGG^N, and set D of data. It is written as:*

$$EGG = (id, V, E, EGG^N, D), \tag{5}$$

where id is a unique identifier,

$$|V| = n_V,\ n_V \in \mathbb{N}, \tag{6}$$

$$|E| = n_E,\ n_E \in \mathbb{N}, \tag{7}$$

$$|EGG^N| = n_{EGG},\ n_{EGG} \in \mathbb{N}, \tag{8}$$

$$n_V + n_E + n_{EGG} \geq 0. \tag{9}$$

Each vertex $v \in V$ is a tuple of an id, and a set of data D_v:

$$v = (id, D_v). \tag{10}$$

Each edge $e \in E$ is a tuple of an id, a multiset C_e of connection tuples, and a set of data D_e:

$$e = (id, C_e, D_e), \tag{11}$$

where

$$C_e = \{(\mu_a, \theta_a), (\mu_b, \theta_b), (\mu_c, \theta_c), \ldots\},\ |C_e| \geq 0, \tag{12}$$

where

$$\mu_a, \mu_b, \mu_c, \ldots \in V \cup E \cup EGG^N, \tag{13}$$

$$\theta_a, \theta_b, \theta_c, \ldots \in \{EdgeToElement, ElementToEdge, Bidirectional\}. \tag{14}$$

EdgeToElement, ElementToEdge, Bidirectional represent direction of navigation i.e. possibility of traversing from one EGG *element to another.* $\mu_a, \mu_b, \mu_c, \ldots$ *represent elements of* EGG *i.e. vertices, edges and nested Extended Graph Generalizations. For the sake of brevity, it was assumed that:* $EdgeToElement \equiv \rightarrow$, $ElementToEdge \equiv \leftarrow$ *and* $Bidirectional \equiv \leftrightarrow$.

Set D of data:

$$D = \{d_x, d_y, d_z, \ldots\},\ |D| \geq 0. \tag{15}$$

Each of $d_x, d_y, d_z, \ldots$ *is an abstract concept which represents data with any internal structure.*

3 *Extended Graph Generalization* Abstract Syntax

The abstract syntax of *Extended Graph Generalization* expressed in UML 2.5.1 and OCL 2.4 is presented on Fig. 1. The fact that *Extended Graph Generalization* is the set of entities of Element abstract type belongs to its most important features. According to the polymorphism concept each Element type entity is of exactly one type: Vertex, Edge or EGG. Moreover, each Element type entity may contain a set of an arbitrary number of abstract Data type entities.

Generalization-related constraints:

- NOT HYPER [**nH**] – the connection arity limit, which must be = 2; the multiplicity constraint 2 for the connection property belonging to Edge
  `context Edge inv: self.connection -> size() = 2`
- NOT ULTRA [**nU**] – connecting other connections excluded; no generalization between Edge and abstract category Element
- NOT MULTI [**nM**] – the elements cannot be multiply connected by the same connecting element; the uniqueness of connection property elements in the Edge
  `context Edge inv: self.connection -> isUnique(e : Element | e.id)`
- NOT SHARED AGGREGATION [**nS**] – Element type entities may belong to exactly one EGG; the shared aggregation end in the association between Graph and Element has the multiplicity constraint equal to 1
  `context Element inv: self.graph -> size() = 1`

The EGG is presented formally in Sect. 2, while in Sect. 3 the *Extended Graph Generalizations* abstract syntax is shown. In both cases a convention based on the complete EGG form that is the one having all generalizations and extensions is assumed. As the result, both approaches are possible: adding particular generalizations and extensions to the fundamental Graph structure like SHARED

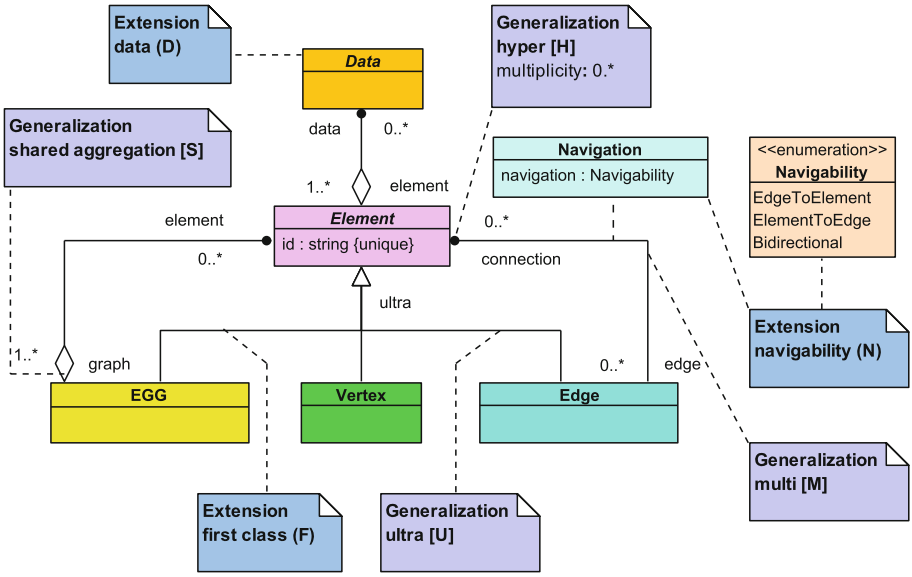

Fig. 1. *Extended Graph Generalization* abstract syntax expressed in UML 2.5.1 and OCL 2.4

AGGREGATION or adding their negations marking the exclusions of a particular generalization or extension from the complete EGG like NOT SHARED AGGREGATION.

EGG constitutes a set of the Element abstract type entities. The set may be empty, which means that an EGG having no elements may exist. It is also important that each Element entity must belong to at least one EGG entity and it may belong to many EGG entities as well. It guaranties realization of the SHARED AGGREGATION feature. The EGG itself is an Element specialization, which provides the possibility of nesting the Graph type entities. In the consequence, the EGG constitutes the FIRST CLASS semantic category (first class citizen) [12] in the EGG, which in turn means that EGG as well as Vertex and Edge can be connected using Edge.

Vertex is the Element abstract type specialization. Vertex is a first class category in EGG and Vertex, EGG, and Edge can be connected using Edge.

Edge is the Element abstract type specialization, which guaranties realization of the ULTRA feature. According to [1] the Edge is of the same importance in representing data structures as the Vertex is. It has the connection property, which joins Element abstract type entities. Edge may contain any number of elements, which provides the realization of HYPER feature. The case in which Edge does not join any Element type entity is also allowed. There is no limit for joining the same elements many times by the connection property, which in turn corresponds to the fature specified as MULTI. In other words,

each Element type entity may be connected with the help of the connection property by any number of Edge type entities.

Data has the abstract type, that is it requires a concretization by concrete data structures. The Data type entities are contained also in abstract type entity Element. It means that Data type entities are contained in such entities like EGG, Vertex or Edge. It should be remarked that being a part of Data type entity does not have exclusive character, that is thay may be shared between other entities. This category provides DATA feature.

Element has abstract character and requires a concretization by one of the following structures: EGG, Vertex or Edge. The Element type entities are contained in EGG type entities. This association does not have exclusive character, that is Element type entities may belong to many EGG type entities while they should belong to at least one of them. The Element type entities may be connected by any number of Edge type entities through the connection property belonging to Edge. The Element type contains the id attribute, which is the unique text chain.

connection is the property contained in Edge type. The association class, which joins Edge with Element contains the Navigability type *navigation* attribute. It provides the NAVIGABLE feature.

4 *Extended Graph Generalization* Features

The basic, classical *graph* (1) has connected element and connecting element and for the purpose of discussion in the paper it constitutes the basic structure, the basis and fundament for other generalizations and extensions. A set of generalizations and extensions being the essence of the *Extended Graph Generalization* is presented below.

Generalization is a removal of constrains existing in Graph while *extension* is an addition of new elements to the Graph, i.e. the EGG as a new semantic category, Data and Navigability.

Generalizations:

- HYPER (**H**): the lack of the limit on the connection arity, which may be ≥ 0. Realized by the $0..*$ multiplicity constraint for the connection property belonging to the Edge.
- ULTRA (**U**): the possibility of joining different connections. Realized by the generalization between the Edge and the Element abstract category. It is related to the 12, 13 i 14 expressions in the formal definition.
- MULTI (**M**): the elements may be connected multiple times by the same connecting element. Realized by the lack of the constraint to the uniqueness of the connection property elements in the Edge. It is related to the statement that E is a multiset.
- SHARED AGGREGATION (**S**): the Element entities may belong to more than one EGG. This feature is represented by the shared aggregation end of the association between the EGG and the Element having the multiplicity constraint defined as $1..*$.

Extensions:

- FIRST CLASS (**F**): EGG becomes the first-class semantic category that is it exists as an independent entity and it may be a connected element anologically to the Vertex and Edge type elements.
- DATA (**D**): the EGG type elements, Vertex and Edge may contain data sets represented by the Data abstract semantic category. In the formal definition the *Extended Graph Generalization* is related to the (5, 10, 11, 15) expressions.
- NAVIGABLE (**N**): the connecting elements have the navigability property, which makes it possible to add extra constraints in the scope of the possibility of traversing from one EGG element to another. This feature is realized by the association class between the Edge and the Element. In the formal definition of EGG it is related to the (11) and (12) expressions.

Constraints:

- *acycled* (**A**): acyclic – the constraint, which makes the structure acyclic,
- *planar* (**P**): planar – the constraint, which makes the structure planar.

Many constrains may be defined, but they do not constitute the basis for modifying the basic EGG structure, as they only put additional constraints on the chosen form of the EGG.

5 *Extended Graph Generalization* Semantics

Element it is the abstract structure, which represents one of the following elements: EGG, Vertex, Edge. It has the *id* attribute, which is responsible for the uniqueness of the identifiers of the mentioned above semantic categories. Moreover, the Data element is associated to this category, which means that each element inheriting from the Element category may have a set of data.

EGG it is a recursive structure, i.e. the EGG may be composed of other EGG type structures. This is a first-class element, i.e. it may exist as an independent entity, including the possibility of containing other elements and it may be joined by the Edge on a pair with the Vertex and the Edge. The EGG itself does not have any defined semantics, because it is a universal structure dedicated to constructing the data models.

Vertex it is an element, which may be independent, but always in an EGG as well as in the connection realized by the Edge. From the semantic point of view the Vertex is a connected element and it is its only responsibility.

Edge is a connecting element and it is its main responsibility. But it may also have the responsibility of the connected element.

Data is an abstract element responsible for representing data. Data may have a simple form, e.g labels or simple values or a more complicated form. Data has abstract character and the definition of the data structure inside Data is out of the scope of the *Extended Graph Generalization* definition. Data is not independent entity, i.e. it has supplementary role for EGG, Vertex and Edge. Data may be shared between different elements.

connection constitutes a property of the association between the Edge and the abstract Element entity. It groups the Element type entities within a concrete Edge type entity. From the semantic point of view it joins elements, which are in an association represented by the Edge.

6 *Extended Graph Generalization* Concrete Syntax

Vertex is represented by a circle with the *id* placed inside or close to it (Fig. 2). The $\bigcirc id$ lub $id : Vertex$ symbols are introduced to unify formal notation.

Fig. 2. Graphical representation of Vertex

Edge is represented by a diamond with the *id* placed inside or close to it (Fig. 3). The $\diamond id$ lub $id : Edge$ symbols are introduced to unify formal notation.

Fig. 3. Graphical representation of Edge

connection of diagram elements, that is a particular Edge with other element or with itself is represented by a solid line joining a particular Edge with a joined element. The small filled dot is placed at the end which is at Edge side. In the case of navigability its direction is determined by an arrow or arrows (Fig. 4). The $\bullet\!\!-$ symbol and symbols taking into account navigability $\bullet\!\!\rightarrow$, $\bullet\!\!\leftarrow$, $\bullet\!\!\leftrightarrow$ are introduced to unify formal notation.

Fig. 4. Graphical representation of connection

Figure 5 illustrates three EGG elements, namely Edge, connection and Vertex and it shows the following examples of joining Edge with Vertex: $\diamond edge \bullet\!\!\rightarrow \bigcirc vertex$ as well as two Edge type elements: $\diamond edge \bullet\!\!\rightarrow \diamond\, edge$.

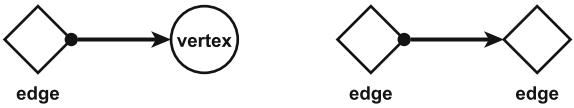

Fig. 5. Graphical representation of Edge, connection and Vertex

EGG is represented by a rectangle with the corners rounded. An EGG g_0 and three nested EGGs: g_1, g_2, g_3 are depicted on Fig. 6. EGGs are independent each of other. EGG g_1 i g_2 share with g_3 some area. As the result, the situation where some Edge and Vertex are shared by these EGGs can be illustrated.

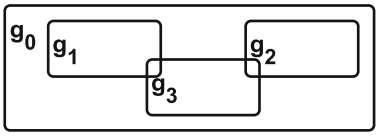

Fig. 6. Graphical representation of EGGs

Data is represented by a rectangle. The Data d_1 is shown on Fig. 7.

Fig. 7. Graphical representation of Data

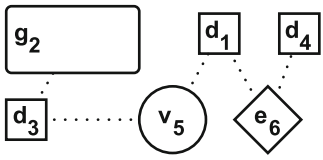

Fig. 8. Graphical representation of connected Data

Data is associated to to EGG, Edge or Vertex with the help of a dotted line (Fig. 8).

Both color and character format do not have any meaning for the grammar.

6.1 *Extended Graph Generalization* Diagram Example

An example $\mathsf{EGG}^{\mathsf{HUMS}}_{\mathsf{FDN}}$ symbolized by g_0 and expressed in the graphical concrete syntax is presented on Fig. 9.

The diagram from Fig. 9 is also presented in the form of formal symbolic notation.

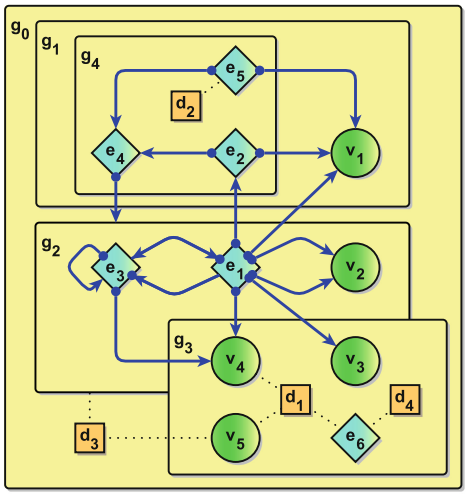

Fig. 9. *Extended Graph Generalization* diagram example

$g_0 = (\text{“}g0\text{”}, \varnothing, \varnothing, \{g_1, g_2, g_3\}, \varnothing);$
$g_1 = (\text{“}g1\text{”}, \{v_1\}, \varnothing, \{g_4\}, \varnothing);$
$g_2 = (\text{“}g2\text{”}, \{v_2, v_3, v_4\}, \{e_1, e_3\}, \varnothing, \{d_3\});$
$g_3 = (\text{“}g3\text{”}, \{v_3, v_4, v_5\}, \{e_6\}, \varnothing, \varnothing);$
$g_4 = (\text{“}g4\text{”}, \varnothing, \{e_2, e_4, e_5\}, \varnothing, \varnothing).$

$v_1 = (\text{“}v1\text{”}, \varnothing);$
$v_2 = (\text{“}v2\text{”}, \varnothing);$
$v_3 = (\text{“}v3\text{”}, \varnothing);$
$v_4 = (\text{“}v4\text{”}, \{d_1\});$
$v_5 = (\text{“}v5\text{”}, \{d_1, d_3\}).$

$e_1 = (\text{“}e1\text{”}, \{(v_1, \rightarrow), (v_2, \rightarrow), (v_2, \rightarrow), (v_3, \rightarrow), (v_4, \rightarrow), (e_2, \rightarrow), (e_3, \leftrightarrow)\}, \varnothing);$
$e_2 = (\text{“}e2\text{”}, \{(v_1, \rightarrow), (e_4, \rightarrow)\}, \varnothing);$
$e_3 = (\text{“}e3\text{”}, \{(v_4, \rightarrow), (e_1, \leftarrow), (e_3, \rightarrow)\}, \varnothing);$
$e_4 = (\text{“}e4\text{”}, \{(g_2, \rightarrow)\}, \varnothing);$
$e_5 = (\text{“}e5\text{”}, \{(v_1, \rightarrow), (e_4, \rightarrow)\}, \{d2\}).$

7 A Social Network Model Example Expressed in the EGG Categories

A domain example of the EGG structures usage for representing social networks is shown in order to present the concept introduced in the paper. The way the examples are presented illustrates how the addition of subsequent features (generalizations and extensions) allows to represent the subsequent relationship kinds. The role of vertices is played in the examples by persons, while the role of edges – the connections between them. Figure 10 shows EGG, which is semantically equivalent to the graph structure. A structure, which represents mutual bi-directed inter-personal relationships *is friend of* is shown on the diagram.

The *chats with* relationship between persons, which talk between each other is shown on Fig. 11. The $\mathsf{EGG}^{\mathsf{H}}$ structure was applied to express this information.

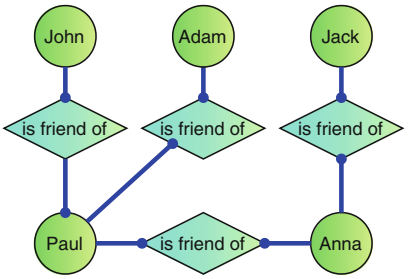

Fig. 10. Exemplary EGG depicting *is friend of* relationship in social network

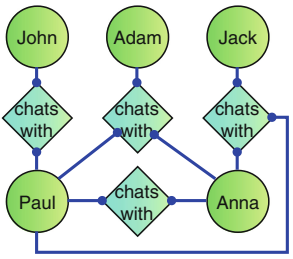

Fig. 11. Exemplary $\mathsf{EGG}^{\mathsf{H}}$ depicting *chats with* relationship in social network

Figure 12 shows a social network, which takes into account the persons grouping aspect. The subgroups can be identified inside the groups as well. In order to realize this kind of relationships the connections between connections (group – subgroup) must be applied. The $\mathsf{EGG}^{\mathsf{HU}}$ structure is used for this purpose. In order to enrich this structure by data annotations, the $\mathsf{EGG}_{\mathsf{D}}^{\mathsf{HU}}$ structure with both verticeas and edges annotations is used in the way shown on Fig. 13.

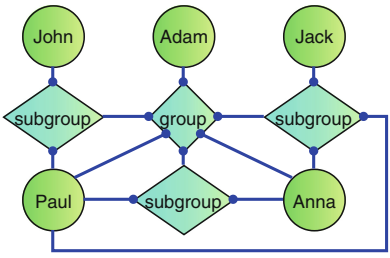

Fig. 12. Exemplary $\mathsf{EGG}^{\mathsf{HU}}$ depicting *group* and *subgroup* relationship in social network

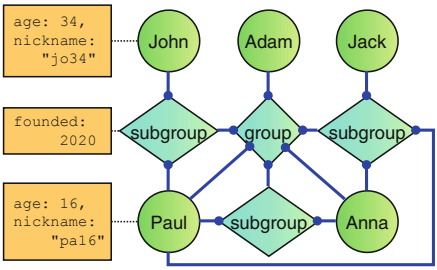

Fig. 13. Exemplary $\mathsf{EGG}_{\mathsf{D}}^{\mathsf{HU}}$ depicting *group* and *subgroup* relationship in social network and data annotation of nodes and vertices

Some connections require the ability of connecting the same element multiple times. The *follows* relationship is an example of such the case. It is used to express that a person is interested in the events initiated by other person. However, there is no need to forbid the option of following updates to own events. Figure 14 shows how this requirement is achieved with the application of the

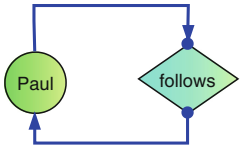

Fig. 14. Exemplary $\mathsf{EGG}_{\mathsf{N}}^{\mathsf{M}}$ depicting *follows* relationship in social network

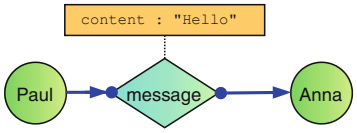

Fig. 15. Exemplary $\mathsf{EGG}_{\mathsf{DN}}$ depicting *message* relationship with data annotation in social network

$\mathsf{EGG}_{\mathsf{N}}^{\mathsf{M}}$ structure. This example, together with the one from Fig. 15 illustrates connections navigability as well.

Figure 16 presents the $\mathsf{EGG}_{\mathsf{F}}^{\mathsf{HU}}$ structure. It shows the relationship between a person named *Jack* in the *follows updates of* connection with EGG, which represents the network of relations followed by *Jack*. The example is extended according to the Fig. 17 to involve the $\mathsf{EGG}_{\mathsf{F}}^{\mathsf{HUS}}$ structure. It models the sharing some elements by persons named *Sally* and *Jack* within their followed networks.

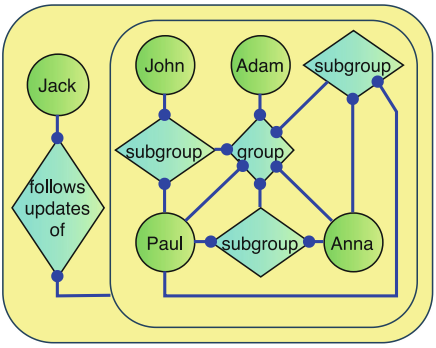

Fig. 16. Exemplary $\mathsf{EGG}_{\mathsf{F}}^{\mathsf{HU}}$ depicting *follows updates of* relationship between node and EGG in social network

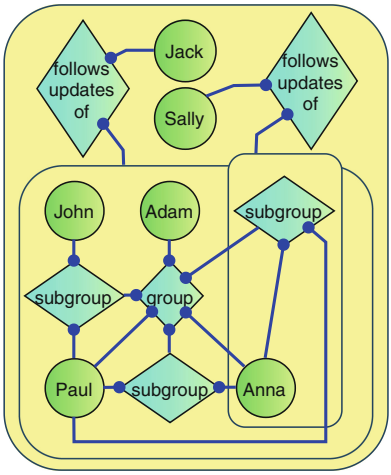

Fig. 17. Exemplary $\mathsf{EGG}_{\mathsf{F}}^{\mathsf{HUS}}$ depicting *follows updates of* relationship between node and EGG in social network with shared sub-EGG

8 Conclusions

The concept, which constitutes the basis for defining contemporary known data structures has been revisited in the paper. As the analysis result, the connected

element (Vertex) and the connecting element (Edge) as well as data representation element (Data) have been identified. Then, the set theory based definition (Definition 2) being a generalization of the fundamental set theory based graph definition (Definition 1) has been introduced. This generalization has been named *Extended Graph Generalization* (EGG). The notion is compliant to all known data structures. As the result of introduction of the general EGG definition all possible data structures definitions have been caught in one place. A kind of standardization has been proposed this way, which is expected to be helpful for future research work in the metamodeling and modeling domains.

Then, the abstract syntax of EGG (Definition 2) has been introduced and it has been enriched by syntactical elements semantics description. The abstract syntax and the Definition 2 are equally general. The EGG abstract syntax enables defining features like Generalizations, Extensions and Constraints. It is worth noticing that two kinds of constraints may be introduced to the abstract syntax. The first group of constraints constitutes negation of Generalizations and Extensions features and introduces structural changes to the EGG, while the second one (Constraints) has the form, which does not modify the EGG structure.

The EGG structure constitutes both generalization and extension of classical graphs and it is not possible to express it in the form of the graph without a modification of graph elements semantics. For example the ULTRA feature consisting in that the *edge* may also play the role of a connected element can be expressed with the help of a classical graph only if the Vertex semantics is changed - it should take the edge functions. Analogically, in the case of HYPER feature, where the Edge arity is greater than 2 the Edge element should be represented by nodes, which again would change the original semantics of classical graph elements. That is why the EGG is actual generalization and extension of the graph concept and not just a typical construction of more complex structures based on the graph definition. As the result, the new general EGG structure has been defined, which, together with the mentioned features, constitutes a form convenient for defining more expressive and semantically rich metamodels and models than the graph ones. The examples of such metamodels are the Association-Oriented Metamodel (AOM) [5,7] and the Context-Driven Meta-Modeling (CDMM) language [8,14,15]. Their definitions based on the EGG structure will be presented in succeeding publications.

The introduced EGG abstract syntax with semantics constitutes the basis for defining EGG concrete syntax. The semantics of its syntactical elements is identical to the semantics of their counterparts from the EGG abstract syntax. The EGG concrete syntax is general enough to apply it for representing arbitraly complex EGG structures as well as their any constraints in the uniform way. It is also universal enough for constructing EGG graphs or their particular special forms in any metamodel, including AOM and CDMM.

The need for defining own symbolic language has been shown in the paper. The graphical language introduced in the paper as a concrete syntax corresponds to the symbolic language. As the result of the possible introducing concrete syntaxes based on common abstract syntax the languages application fields can be differentiated.

A very important element of the paper is that both in the Definition 2 and in the EGG abstract syntax the clear separation between data and the structure has been introduced. Since now, the structure can be named the data carrier while data can be told to be spread on the structure. The data (Data) itself may have any form. Particularly, Data may have the EGG structure as one possible form. It is, at the same time the most general data form. This fact was not shown in the EGG abstract syntax due to the need of underlying the arbitrariness of Data and the possibility of the use of Data for representing Constraints in the EGG. As the result, Data can play two roles in the EGG. It could represent domain-specific data and/or represent information about the EGG itself, which are in turn used in Constraints. The arbitrariness of the forms and roles that could be associated to Data in the EGG abstract syntax has been underlined by making this syntactical element abstract.

References

1. Bildhauer, D.: Associations as first-class elements. In: Proceedings of the 2011 Conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference, DB&IS 2010, pp. 108–121. IOS Press, NLD (2011)
2. Boyd, M., McBrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 69–109. Springer, Heidelberg (2005). https://doi.org/10.1007/11603412_3
3. Bretto, A.: Hypergraph Theory: An Introduction. Mathematical Engineering. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-00080-0
4. Ebert, J., Winter, A., Dahm, P., Franzke, A., Süttenbach, R.: Graph based modeling and implementation with EER/GRAL. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, pp. 163–178. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0019922
5. Jodłowiec, M.: Complex relationships modeling in association-oriented database metamodel. In: Nguyen, N.T., Hoang, D.H., Hong, T.-P., Pham, H., Trawiński, B. (eds.) ACIIDS 2018. LNCS (LNAI), vol. 10752, pp. 46–56. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75420-8_5
6. Joslyn, C., Nowak, K.: Ubergraphs: A definition of a recursive hypergraph structure. arXiv preprint arXiv:1704.05547 (2017)
7. Krótkiewicz, M.: A novel inheritance mechanism for modeling knowledge representation systems. Comput. Sci. Inf. Syst. **15**(1), 51–78 (2018)
8. Krótkiewicz, M., Zabawa, P.: AODB and CDMM modeling – comparative case-study. In: Nguyen, N.T., Hoang, D.H., Hong, T.-P., Pham, H., Trawiński, B. (eds.) ACIIDS 2018. LNCS (LNAI), vol. 10752, pp. 57–68. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75420-8_6

9. Luoma, J., Kelly, S., Tolvanen, J.P.: Defining domain-specific modeling languages: collected experiences. In: 4th Workshop on Domain-Specific Modeling (2004)
10. McQuade, S.T., Merrill, N.J., Piccoli, B.: Metabolic graphs, life method and the modeling of drug action on mycobacterium tuberculosis. arXiv preprint arXiv:2003.12400 (2020)
11. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley Professional, Boston (2004)
12. Strachey, C.: Fundamental concepts in programming languages. Higher-Order Symb. Comput. **13**(1–2), 11–49 (2000)
13. Voloshin, V.I.: Introduction to Graph and Hypergraph Theory. Nova Science Publisher, New York (2009)
14. Zabawa, P.: Meta-modeling. In: Nguyen, N.T., Hoang, D.H., Hong, T.-P., Pham, H., Trawiński, B. (eds.) ACIIDS 2018. LNCS (LNAI), vol. 10752, pp. 91–101. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75420-8_9
15. Zabawa, P., Hnatkowska, B.: CDMM-F – domain languages framework. In: Świątek, J., Borzemski, L., Wilimowska, Z. (eds.) ISAT 2017. AISC, vol. 656, pp. 263–273. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67229-8_24