Applying Graphs for Multilevel Domain Model Description

Alexander O. Sukhov

Perm State National Research University, Perm, Russia. Sukhov.PSU@gmail.com

Abstract. In the article the approach to creation of the formal description of the metalanguage, used for development of visual domain-specific languages of information systems modeling, based on directed pseudo-metagraphs, is considered. Using graph models for formal metalanguage definition of system MetaLanguage allows us to describe its properties, to develop algorithms of horizontal and vertical metamodels and models transformation.

Keywords: pseudo-metagraphs, graph grammars, domain-specific languages, metalanguage.

Introduction

Now the technologies based on using of models in information systems development process are widely applied. Model is an abstract description of system characteristics which are important from the viewpoint of modeling purposes. Model is described in some formal language. To each task solution can be applied a modeling language which uses concepts and relations from the information system domain. The systems life cycle is based on usage of several models that are described from the various points of view and with different levels of abstraction. Such approach is caused by that system development process consists of several stages: analysis, design, implementation, testing.

Several levels of models are created at system designing: the data that are stored in system database is a state model of the information system domain; their description, which providing a data interpretation or code generation to work with them, is a metamodel; for developing this model special formal language, which allows to work in terms of the appropriate domain, is applied – the meta-metamodel here is used.

In fact, system creation with usage of modern workbenches represents the development of domain-specific languages (DSLs) – information system meta-metamodels. DSLs are easy to understand for users as they operate with domain terms. Therefore now a large number of DSLs is developed for using in different domains, for example, for business processes modeling and the designing applications for mobile devices.

Despite all DSL advantages they have one big disadvantage – complexity of the designing. The language workbench or DSM-platform is the instrumental software intended to support development and maintenance of DSLs. Usage at DSLs creation a language workbench considerably simplifies the process of their designing [1]. The language used to create other languages is called metalanguage.

The MetaLanguage system is being developed at the Department of Software and Computing Systems Mathematical Support of Perm State National Research University. MetaLanguage system is a tool for creating visual dynamic adaptable domain-specific modeling languages used for development of information system [2]. To describe the metamodels MetaLanguage toolkit uses metalanguage, which basic constructions are entity, relation and constraint.

Multilevel Mathematical Model of Metalanguage

Using constructions entity and relation it is possible to build any model, including an invalid in the current domain. There are various formalisms for specifying syntax of visual languages: automatic models [3], algorithmic nets [4], graph grammars [5], etc.

Most of the existing approaches to definition of visual languages syntax consider a concrete syntax, and only in rare cases – abstract syntax. The abstract syntax of visual modeling languages does not need all those details that are presented in a concrete syntax: it is possible to abstract from the choice of symbols used to display the language elements, and their geometrical parameters, etc.

Graph grammars are proposed to define the formal rules of models creation. Graph grammar is a generalization of Chomsky grammars on graphs. To define a grammar it is required to specify the finite sets of terminal and nonterminal symbols, a finite set of production rules, and select the start symbol in nonterminal symbols set. For representation graph grammars it is necessary to choose such type of graphs which would be provided the opportunity for an iteratively metamodels definition, unified representation and description of domain models and metamodels. There are several types of graphs that are used for representation graph grammars: the classical graphs, digraphs, multigraphs, pseudographs, hi-graphs, hypergraphs, metagraphs and others.

Production rules in graph grammar contain the left- and the right-hand side. If we generalize the classic definition of graph grammars, then right-hand side of the rule may be not only a labeled graph, but the code in any programming language, and also a fragment of a visual model described in other notation. That is why the graph grammar can be used for generation syntax correct models and for refactoring of existing models, code generation and model transformations from one modeling language to another.

As an analysis result of various representations of graph grammars it was determined that the most appropriate formalism for describing the syntax of visual modeling languages in MetaLanguage system are graph grammars, which are constructed on pseudo-metagraphs [6]. Let's define the domain metamodel and model, applying the selected formalism.

Metamodel Graph

Let $Ent = \{ent_i\}, i \in \mathrm{N}$ (N is a set of natural numbers) be a set of metamodel entities that is finite at every fixed moment of time, but is extended at entity creation and is reduced at removing.

Let's designate each entity as a tuple

$$ent_i = \{EName_i, EICount_i, EAttr_i, EOpp_i, ERest_i, EUnique_i\},$$

where $EName_i$ is an entity name, $EICount_i$ is a amount of entity instances, $EAttr_i$ is a entity attributes, $EOpp_i$ is a entity operations, $ERest_i$ is a set of constraint imposed on the entity, $EUnique_i$ is a flag of uniqueness. Sets $EAttr_i$, $EOpp_i$, $ERest_i$ are finite at every fixed moment of time.

Let's divide all characteristics of i-th entity on two groups EG_i^1 and EG_i^2. The first group consists of those characteristics, which will be represented by separate nodes in graph model: sets of attributes, operations, and constraints imposed on the entity, i.e.

$$EG_i^1 = \{EAttri, EOppi, EResti\}.$$

Characteristics of second group $EG_i^2 = \{EName_i, EICount_i, EUnique_i\}$ (entity name, amount of entity instances, flag of uniqueness) will be attributed to node of the corresponding entity directly.

$Rel = \{rel_i\}, i \in \mathrm{N}$ denotes a set of metamodel relations that is finite at every fixed moment of time, but extends at relation creation and reduces at removing.

Let relation be a tuple

$$reli = \{RNamei, RTypei, RAttri, RMulti, RResti, RUniquei\},$$

where $RName_i$ is a relation name, $RType_i$ is a relation type, $RAttr_i$ is a relation attributes, $RMult_i$ is a multiplicity, $RRest_i$ is a relation constraints, $RUnique_i$ is a flag of uniqueness. Sets $RAttr_i$, $RRest_i$ are finite at every fixed moment of time.

Characteristics of *i*-th relation will be divided into two groups RG_i^1 and RG_i^2. The first group comprises a set of relation attributes and constraints imposed on the relation. The second group includes the following characteristics: «name», «type», «multiplicity», «flag of uniqueness», i.e.

$$RG_i^1 = \{RAttr_i, RRest_i\}, RG_i^2 = \{RName_i, RType_i, RMult_i, RUnique_i\}.$$

Consider directed pseudo-metagraph $GMM = (V, E)$. Let a set of metamodel graph nodes is a union of seven disjoint subsets:

$$V = Ent \bigcup_{i=1}^{|Ent|} EAttr_i \bigcup_{i=1}^{|Ent|} EOpp_i \bigcup_{i=1}^{|Ent|} ERest_i \bigcup Rel \bigcup_{i=1}^{|Rel|} RAttr_i \bigcup_{i=1}^{|Rel|} RRest_i . \quad (1)$$

The set of pseudo-metagraph arcs E is divided into six disjoint subsets:

- $EEA = \{eea_i\}, i = \overline{1, |Ent|}$ is a set of arcs connecting each metamodel entity with set of attributes belonging to it;
- $EEO = \{eeo_i\}, i = \overline{1, |Ent|}$ is a set of arcs connecting each metamodel entity with set of operations over it;
- $EER = \{eer_i\}, i = \overline{1, |Ent|}$ is a set of arcs connecting each metamodel entity with set of constraints imposed on it;
- $ERA = \{era_i\}, i = \overline{1, |Rel|}$ is a set of arcs connecting each metamodel relation with set of its attributes;
- $ERR = \{err_i\}, i = \overline{1, |Rel|}$ is a set of arcs connecting each metamodel relation with set of constraints imposed on it;
- $EERR = \{eerr_i\}, i \in \mathrm{N}$ is a set of arcs, appropriate to links between entities and relations, that is finite at every fixed moment of time, but is extended at entity (relation) creation and is reduced at removing.

Thus, we see that

$$E = EEA \cup EEO \cup EER \cup ERA \cup ERR \cup EERR\,. \qquad (2)$$

The metamodel graph is a directed pseudo-metagraph $GMM = (V, E)$, for which (1) and (2), where V is a nonempty set of graph nodes, E is a set of graph arcs.

Let's consider an example. We will construct a metamodel graph for the entity «Use Case» of UML Use Case diagrams. Attributes of the entity «Use Case» are «Name», «Description», «Creation_Date». Operations that can be performed on entity – «SetName()», «SetDescription()», «SetDate()», i.e. for given entity

$$EAttr_i = \{\text{«}Name\text{»}, \text{«}Description\text{»}, \text{«}Creation\_Date\text{»}\},$$

$$EOpp_i = \{\text{«SetName()»}, \text{«SetDescription()»}, \text{«SetDate()»}\},\ ERest_i = \varnothing\,.$$

The metamodel graph corresponding to a fragment of the «Use Case» entity shown in Fig. 1.

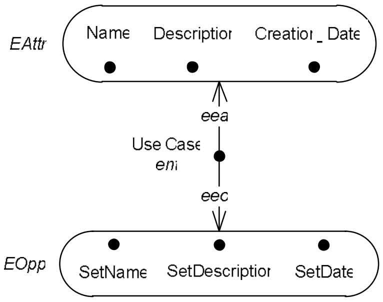

Fig. 1. Fragment of metamodel graph for «Use Case» entity

As can be seen from figure $EEA = \{eea_{i_1}\}$, $EEO = \{eeo_{i_1}\}$, $EER = \varnothing$, $EERR = \varnothing$.

Model Graph

The model is actually an «instance» of metamodel in which:

- the attributes of entity is a concrete values;
- there are no operations over entity instances and constraints imposed on the entity and relation instances;
- inheritance relation instances can’t be created.

Let's designate a set of all models which have been created based on the current metamodel through $M = \{m_k\}, k \in \mathrm{N}$ that is finite at every fixed moment of time, but extends at model creation and reduces at removing.

Let's introduce following notation:

- $EntI_i$ is a set of instances of *i*-th entity;
- $EAttrI_{j_i}$ is a set of attribute values for *j*-th instance of *i*-th entity;
- $RelI_k$ is a set of instances of *k*-th relation;
- $RAttrI_{k_l}$ is a set of attribute values for *k*-th instance of *l*-th relation.

Sets $EntI_i$, $EAttrI_{j_i}$, $RelI_k$, $RAttrI_{k_l}$ are finite at every fixed moment of time, but extend at entity (relation) instance creation and reduce at removing.

Examine the directed pseudo-metagraph $GM = (VI, EI)$. Let a set of model graph nodes is a union

$$VI = \bigcup_{i=1}^{|Ent|}\left(EntI_i \bigcup_{j=1}^{|EAttr_i|} EAttrI_{j_i} \right) \bigcup_{k=1}^{|Rel|}\left(RelI_k \bigcup_{l=1}^{|RAttr_k|} RAttrI_{l_k} \right). \tag{3}$$

Consider the following example. Let's create a model graph for instance of «Use Case» entity (Fig. 2).

From a figure it is apparently that $EAttrI_i$ = {«Pass_exam», «Use Case describes passing an exam process», «21/06/09»}.

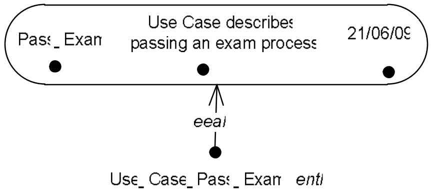

Fig. 2. Model graph corresponding to «Use Case» entity instance

The set EI divides into three disjoint subsets:

- $EEAI = \{eeaI_i\}, i = \overline{1, |EntI|}$ is a set of arcs connecting each entity instance with set of attributes belonging to it;
- $ERAI = \{eraI_i\}, i = \overline{1, |RelI|}$ is a set of arcs connecting each relation instance with set of attributes belonging to it;

- $EERRI = \{eerrI_i\}, i \in \mathrm{N}$ is a set of arcs corresponding to the links between entity instances and relation instances.

Thus, we see that

$$EI = EEAI \bigcup ERAI \bigcup EERRI\ . \qquad (4)$$

You can see from the Fig. 2 that for represented instance of «Use Case» entity $EEAI = \{eeaI_{i_1}\}$, $EERRI = \varnothing$.

The model graph is a directed pseudo-metagraph $GM = (VI, EI)$, for which (3) and (4), where VI is a nonempty set of graph nodes, EI is a set of graph arcs.

Conclusion

For unified models creation the mathematical model – graph grammars based on pseudo-metagraphs – was constructed. This formalism has allowed to describe basic elements and algorithms which MetaLanguage uses in its work: algorithms for creation/modification of domain metamodels and models, algorithms for vertical models transformation, algorithms for constraint checking.

References

1. Lyadova L.N., Sukhov A.O. Visual Languages and DSL-Tools: Development Methods and Means (in Russian) // Proceedings of the Congress on Intelligence Systems and Technologies “AIS-IT’10”. – 2010. – Vol. 1. – P. 374-382.

2. Sukhov A.O. The Development Environment of Visual Domain-Specific Modeling Languages // Mathematics of Program Systems. – 2008. – P. 84-94.

3. Stasenko A.P. Automat Model of Visual Description (in Russian) // Computational Technologies. – 2008. – Vol. 13. – P. 70-87.

4. Korolev O.F. Algorithmic Net as a Visual Programming Language (in Russian) // SPIIRAS Proceedings. – 2005. – Vol. 2. – P. 130-137.

5. Rekers J., Schuerr A. A Graph Grammar approach to Graphical Parsing // Visual Languages, Proceedings, 11th IEEE International Symposium. – 1995. – P. 195-202.

6. Sukhov A.O. Analysis of Formalisms for Visual Modeling Languages Description // Modern Problems of Science and Education. – 2012. – Vol. 2. – P. 1-8.