An Approach to Development of Visual Modeling Toolkits

Alexander O. Sukhov, Lyudmila N. Lyadova

***Abstract* —** The approaches based on applying of metamodeling and domain-specific languages are widely used in software engineering. There are many different tools for creating visual domain-specific modeling languages with a possibility of determining user's graphical notations. However these tools possess disadvantages. The article presents an approach to the development of language workbench that allows to eliminate some restrictions of existing DSM-platforms. The MetaLanguage system is designed for creation of visual dynamic adaptable domain-specific modeling languages and for models construction with these languages. It allows executing transformations of the created models in various textual and graphical notations. Basic metalanguage constructions of this system are described. The formal description of modeling languages metamodel used in MetaLanguage is given. The architecture of MetaLanguage toolkit is presented.

***Keywords* —** DSM-platform, graphs, metamodeling, visual domain-specific languages.

I. Introduction

Creation of information systems with usage of the modern tools is based on the development of the various models describing the domain of information system, defining data structures and algorithms of system functioning. At implementation of *model-driven approach* to software development the models become a central element at all stages of system creation. The model-based approach is capable at information system creation to unite efforts of developers and domain experts. This approach makes the system more flexible, since for its change there is no necessity of modification of source code "by hand", it is enough to modify a visual model, and with this task even nonprofessional programmers can cope [1], [2].

At usage of this approach the models describing system from the various points of view, with a different level of abstraction and with usage of various modeling languages are created. For coordination of various system descriptions it is necessary to construct the whole hierarchy of models: model, metamodel, meta-metamodel, etc., where *model* is an abstract description of system (object, process) characteristics that are important from the point of view of the modeling purpose. A model is created with usage of specific modeling language. A *metamodel* is a model of the language, which is used for models development, a *meta-metamodel (metalanguage)* is a language, on which metamodels are described.

For model-based approach implementation it is necessary to use toolkit, which will be convenient to various participants of system development process. The general-purpose modeling languages, such as UML, are not able to cope with this task, because they have some disadvantages [3], [4]:

– Diagrams are complicated for understanding not only for experts, who take part in system engineering, but in some cases even for professional developers.
– Object-oriented diagrams can not adequately represent domain concepts, since work is being done in terms of "class", "association", "aggregation", etc., rather than in domain terms.

That is why at implementation of model-based approach the domain-specific modeling languages (DSMLs), created to work in specific domains, are increasingly used. Domain-specific languages are more expressive, simple on applying and easy to understand for different categories of users as they operate with domain terms. For this reason now a large number of DSMLs is designed for creation of systems in different domains: artificial intelligence systems, distributed systems, mobile applications, real-time and embedded systems, simulation systems, etc. [5]–[7].

Despite of all DSMLs advantages they have one big disadvantage – complexity of their designing. If general purpose languages allow to create programs irrespectively to domain, in case of DSMLs for each domain, and in some cases for each task, it is necessary to create new domain-specific language. Another shortcoming of visual domain-specific language is that it is necessary to create convenient graphical editors to work with it.

To support the process of development and maintenance of DSMLs the special kind of software – *language workbench* (*DSM-platform*) is used [8]. Usage at DSMLs creation of a language workbench considerably simplifies the process of their designing. There are various DSM-platforms for creating visual DSMLs with the ability of determining user's graphical notation: MetaEdit+, Microsoft DSL Tools, Eclipse GMF, QReal, etc. Let's consider the most advanced language workbenches [9], [10].

This work was supported in part by Russian Foundation for Basic Research (grant 14-07-31330).

A. O. Sukhov is with the National Research University Higher School of Economics, Perm, Russia (phone: (+7) 912-589-0986; e-mail: Sukhov.psu@gmail.com).

L. N. Lyadova is with the National Research University Higher School of Economics, Perm, Russia (e-mail: LNLyadova@gmail.com).

II. Related Works

MetaEdit+ is a multiplatform language workbench that enables users to simultaneously work with several projects, each of which can have a few models [11], [12]. At usage of this DSM-platform besides a possibility of domain-specific language creation, the developer receives the CASE tool, into which this language is integrated. MetaEdit+ allows to use several DSMLs at system creation.

The approach based on metamodels (models of modeling languages) interpretation, instead of code generation, which is used in MetaEdit+ allows changing the DSMLs definition at run-time. The system allows working with languages and metalanguages universally, using the same tools. The disadvantage of MetaEdit+ is that this DSM-platform for export of models uses an own file format (MXT) and this affects the openness of technology.

Microsoft DSL Tools [13] and Eclipse GMF [14] technologies provide the user with advanced IDE MS Visual Studio and Eclipse respectively. Thanks to this there is a possibility of code completion on high-level languages "by hand", but it can lead to occasion of inconsistency of diagrams and source code.

Technology Eclipse GMF is most powerful of the above. However, its usage is impeded by high complexity, frequent releases of new versions and lack of documentation. In fact, Eclipse GMF is in a stage of intensive development.

Eclipse environment provides the user with tab GMF Dashboard, which allows to accelerate DSMLs development process by automatically generating some language components. On GMF Dashboard tab the sequence of the operations is represented. Tools of creation of plug-ins for Eclipse, which allow to build diagrams in current domain are realized with these operations.

The multiplatform system QReal [15] allows to define metamodels both in visual and textual view, therefore developers have a possibility to select the most suitable for them format of language description representation. Availability of an interpreter of behavioral diagrams and a debugger of the generated code puts this system to the same position as Microsoft DSL Tools, Eclipse GMF, which use for these purposes IDE. In QReal there is not a possibility of modification of DSMLs description at run-time.

Cases when DSMLs becomes part of other applications are common. For example, a specially designed language for describing business processes can be used in document circulation model. Therefore one more important characteristic of the DSM-platforms is the alienability of DSMLs from the development environment. Microsoft DSL Tools, Eclipse GMF are strongly associated with the development platforms – MS Visual Studio and Eclipse, respectively, therefore languages created by these workbenches can't be exported to external system.

The analysis of DSM-platforms has revealed some restrictions inherent in the majority of the considered systems:

1. Impossibility of multilevel modeling. Presence of such possibility would allow making changes at metalanguage description, to extend it with new constructions, thus bringing the metalanguage to the specifics of domain.
2. Modification of DSMLs description leads to necessity of regeneration of language editor: for modification DSMLs at first it is necessary to change its metamodel, to regenerate the source code of the editor, and only then it is possible to begin build models.
3. "Excess" functionality of the language workbench, which is not used at DSMLs creation. This functionality complicates the study of tools by the users, which are not professional programmers.
4. Lack of tools of horizontal models transformations. These means allow not only to create unified system description on the basis of the models constructed at various stages of system development, but also to generate source code according to user-specified template or to make conversion of the model described with one modeling language to model fulfilled in other graphical notation.

The MetaLanguage system eliminates some restrictions of the considered DSM-platforms.

III. MetaLanguage System

The DSM-platform MetaLanguage is designed to create visual dynamic adaptable domain-specific modeling languages, to construct models using these languages and to transform created models in various textual and graphical notations.

A. Metalanguage of MetaLanguage System

One of the basic elements of language workbench is the *metalanguage* (*meta-metamodel*), which is the language for describing of other languages. Thanks to presence of metalanguage the DSM-platform allows to create domain-specific languages for the various domains that operate with familiar for the user concepts. The main difference between metalanguage of MetaLanguage system from the MOF (Meta Object Facility) approach, used in the majority of DSM-platforms, is that thanks to interpretation of models at various abstraction levels, instead of the source code generation on their basis, it is possible to modify of DSML's constructions in dynamics, at models creation. Besides, the process of metamodel creation becomes multilevel, so developer defining metamodel and selecting it as the metalanguage, can use it to create other metamodels, and this process can be infinite.

The *basic elements* of the metalanguage of MetaLanguage system are entity, relationship and constraint.

The *entity* describes a particular construction of modeling language, i.e. it is the domain object, important from the point of view of the solving problem.

Visual language constructions in rare cases exist independently, more often they are in some way related to each other, therefore at metamodel creation importantly not only to define the basic language constructions, but also correctly specify the relationships between them. The *relationship* is used for describing a physical or conceptual links between entities. Metamodel allows to create three types of relationships: *association*, *aggregation*, *inheritance*.

In practice quite often there are cases when it is necessary to impose some *constraints* on entities and relationships. Some of constraints are set by metamodel structure, and others are described on some languages. All constraints imposed on the metamodel in MetaLanguage system can be divided into two groups: constraints imposed on entities and constraints imposed on relationships.

Let's consider an example. A fragment of metamodel for UML Use Case diagrams is shown in Fig. 1. The metamodel contains two entities "Actor" and "Use Case".

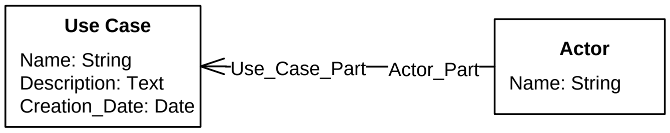

Fig. 1. Fragment of metamodel for UML Use Case diagrams

The entity "Use Case" has following attributes: "Name", "Description", "Creation_Date". The attribute "Name" has a string type and defines the "Use Case" name. The attribute "Description" sets the short description of the "Use Case". An attribute of entity "Actor" is a string attribute "Name", which specifies the name of the Actor.

B. Architecture of MetaLanguage System

The architecture of MetaLanguage is presented in Fig. 2. Uniform storage of all information about the system is the *repository*. It contains information about metamodels, models, entities, relationships, attributes, constraints. Information about the models and metamodels is stored uniformly, that allows to work with it by the same tool.

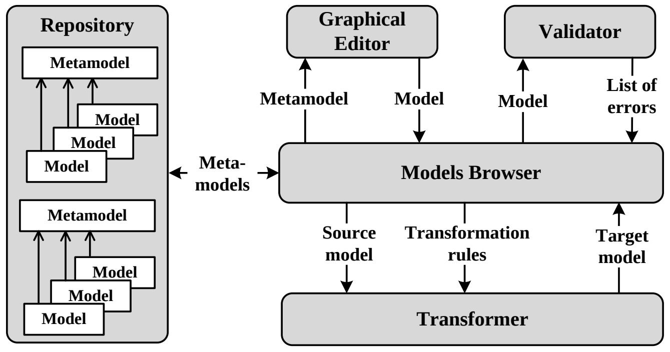

Fig. 2. Architecture of MetaLanguage system

The *browser of models* allows to load/save metamodels together with the models created on their basis, to fulfill over metamodels and models various operations (editing, constraint checking, transformation, etc.). The *graphical editor* is the component, which provides the user the tools for metamodels and models creation. The *validator* allows to check constraints specified by user at metamodel describing. The *transformer* is the component that provides the ability to fulfill horizontal transformations of models to text on target programming language or to visual models, described in other graphical notation.

Having described the basic components of a MetaLanguage system, let's consider how visual domain-specific modeling languages are designed with these tools.

Process of DSML definition begins with metamodel creation. For this purpose it is necessary to specify the main constructions of created language, to define relationships between them, to set constraints imposed on the metamodel entities and relationships. After building of metamodel the developer gets a customizable extensible visual modeling language.

Then the user can design models containing objects that describe specific domain concepts and links between them with using created DSML.

The validator should check up whether model correspond to constraints, which were imposed on metamodel elements.

Then the developer can save the constructed metamodels and models in the form of XML-files or transform these models to other textual or graphical notation.

At metamodel modification the system automatically makes all necessary changes in the models, which are created on the basis of this metamodel.

Using constructions "entity" and "relationship" it is possible to build any model, including an incorrect model in the current domain.

The metamodel of visual modeling language is a graph. There are several types of graphs that can be used for representation of visual languages: the classical graphs, digraphs, multigraphs, pseudographs, hi-graphs, hypergraphs, metagraphs and others [16]-[18].

As an analysis result of various types of graph it has been defined that the most appropriate formalism for describing the syntax of visual modeling languages in MetaLanguage system is pseudo-metagraph [19].

Metagraph is an ordered pair $G = (V, E)$, where V is a finite nonempty set of nodes, E is a set of edges. Each edge $e_k = (V_i, V_j), V_i, V_j \subseteq V$ connects two subsets of nodes.

Let's describe with usage of this formalism a metamodel of a visual modeling language.

IV. FORMAL DESCRIPTION OF A MODELING LANGUAGE METAMODEL

Let $Ent = \{ent_i\}, i \in \aleph$ ($\aleph$ is a set of natural numbers) is a set of metamodel entities, number of set elements is potentially unlimited, but at every fixed point in time is finite.

The set of metamodel relationships denotes as $Rel = \{rel_i\}, i \in \aleph$, number of set elements is potentially unlimited, but at every fixed point in time is finite.

Let's introduce the following designations:

1) $EAttr_i = \{eattr_{i_j}\}, \overline{i = 1, |Ent|}, j \in \aleph$ is the set of metamodel graph nodes, which correspondence to entities attributes;
2) $RAttr_k = \{rattr_{k_l}\}, \overline{k = 1, |Rel|}, l \in \aleph$ is the set of metamodel graph nodes, which correspondence to relationships attributes;

3) $ERest_i = \{erest_{i_j}\}, \overline{i = 1, |Ent|}, j \in \aleph$ is the set of metamodel graph nodes, which correspondence to constraints imposed on entities;
4) $RRest_k = \{rrest_{k_l}\}, \overline{k = 1, |Rel|}, l \in \aleph$ is the set of metamodel graph nodes, which correspondence to constraints imposed on relationships;
5) $EEA = \{eea_i\}, \overline{i = 1, |Ent|}$ is the set of metamodel graph arcs connecting each entity with the set of its attributes;
6) $ERA = \{era_k\}, \overline{k = 1, |Rel|}$ is the set of metamodel graph arcs connecting each relationship with the set of its attributes;
7) $EER = \{eer_i\}, \overline{i = 1, |Ent|}$ is the set of metamodel graph arcs connecting each entity with the set of its constraints;
8) $ERR = \{err_k\}, \overline{k = 1, |Rel|}$ is the set of metamodel graph arcs connecting each relationship with the set of its constraints;
9) $EERR = \{eerr_i\}, i \in \aleph$ is the set of arcs corresponding to links between entities and relationships.

The number of elements of sets $EAttr_i$, $RAttr_k$, $ERest_i$, $RRest_k$, EEA, ERA, EER, ERR, $EERR$ potentially is not limited, but it is finite at every fixed point in time.

The metamodel graph is the directed pseudo-metagraph $GMM = (V, E)$, where V is a nonempty set of graph nodes, E is set of graph arcs and these sets are defined by (1) and (2):

$$V = Ent \cup \left(\bigcup_{i=1}^{|Ent|} EAttr_i \right) \cup \left(\bigcup_{i=1}^{|Ent|} ERest_i \right) \cup \cup Rel \cup \left(\bigcup_{k=1}^{|Rel|} RAttr_k \right) \cup \left(\bigcup_{k=1}^{|Rel|} RRest_k \right) \tag{1}$$

$$E = EEA \cup EER \cup ERA \cup ERR \cup EERR \tag{2}$$

Let's consider an example. We will construct a metamodel graph for the entity "Use Case" of UML Use Case diagrams. Metamodel of this diagram type is shown in Fig. 1. Attributes of the entity "Use Case" are "Name", "Description", "Creation_Date", i.e. for given entity

$EAttr_i$ = {"Name", "Description", "Creation_Date"}.

The metamodel graph corresponding to the fragment of the "Use Case" entity is shown in Fig. 3. As can be seen from the figure:

$ERest_i = \varnothing$, $EEA = \{eea_i\}$, $EER = \varnothing$, $EERR = \varnothing$.

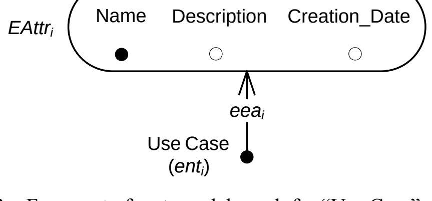

Fig. 3. Fragment of metamodel graph for "Use Case" entity

The model graph is defined similarly. The *model graph* is directed pseudo-metagraph GM = (VI, EI) where VI is a nonempty set of graph nodes, EI is set of graph arcs and these sets are defined by (3) and (4):

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

$$EI = EEAI \cup ERAI \cup EERRI \cup T \tag{4}$$

In the graph model definition the following notation is used:
1) $EntI_i$ is the set of instances of i-th entity;
2) $EAttrI_{i_j}$ is the set of attribute values for j-th instance of i-th entity;
3) $RelI_k$ is the set of instances of k-th relationship;
4) $RAttrI_{k_l}$ is the set of attribute values for l-th instance of k-th relationship;
5) $EEAI$ is the set of arcs connecting each entity instance with set of attributes belonging to it;
6) $ERAI$ is the set of arcs connecting each relationship instance with set of attributes belonging to it;
7) $EERRI$ is the set of arcs corresponding to the links between entities instances and relationships instances;
8) T is the set of arcs of model graph, connecting instances of entities and relationships with those entities and relationships, on which basis they are created.

The number of elements of all these sets potentially is not limited, but it is finite at every fixed point in time.

On the basis of this mathematical model the operations of creation and interpretation of graph models were defined.

Actually these operations are *algorithms of vertical transformations of models* in forward and reverse direction. So at model creation the user, operating with metamodel entities and relationships, creates their instances, thus actually there is a mapping of metamodel graph to model graph. This mapping corresponds *to operation of graph model creation*. After model creation it is necessary to perform checking of the constraints, imposed on metamodel, and, in case of need, conversion of model description to other notation. At execution of these operations the system makes interpretation of model elements, i.e. defines with what entities and relationships they have been created. The mapping of model graph to metamodel graph is used for this purpose. This mapping corresponds to *operation of model graph interpretation*.

With usage of this formalism the mathematical model, which is a basis for implementation of the MetaLanguage system, has been constructed. According to the mathematical model and the requirements to this language workbench, the environment for visual DSMLs creation is designed and algorithms of the MetaLanguage functioning are developed: algorithms for creation/modification/removal of metamodels and models elements, algorithms of constraints checking, algorithms for vertical and horizontal models transformations.

V. Creation of DSML with Usage of the MetaLanguage System

Let's consider an example: construct with usage of the MetaLanguage system the domain-specific language for creation of models of "Smart House" systems.

At first let's analyze the components, which can be a part of "Smart House" systems. The basic elements of systems of this type are:

– life-support systems: heating, air conditioning and ventilation, lighting, security;
– sensors (devices that are responsible for obtaining of various readings and their sending to central panel): motion, leakings, fire and a smoke, closing/opening of object;
– system management tools: voice control, remote control (from a remote computer, from phone, etc.), touch control (control by using of the touch screen of a central panel);
– central panel, which is responsible for receiving of data from sensors, management of life-support systems and obtaining of commands from the user.

For a unified description of entities corresponding to the different life-support systems, let's define the abstract entity "Life-support system", which has the following attributes: "Name", "Manufacturer", "Cost", "State" (defines the state of the system in current time). "Life-support system" entity has the following child entities: "Heating system", "Air conditioning and ventilation system", "Lighting system", "Security system" (see Fig. 4).

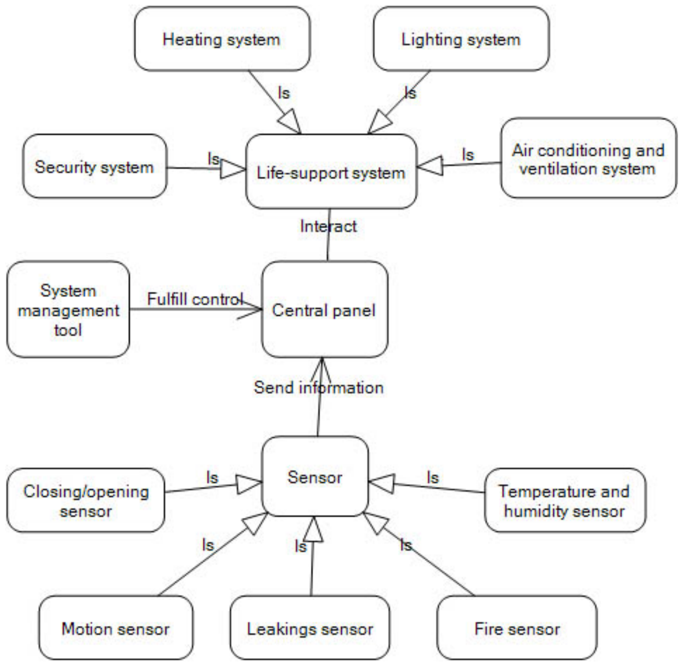

Fig. 4. Metamodel of visual DSML, created in MetaLanguage system

Entities "Heating system" and "Air conditioning and ventilation system" in addition to the inherited attributes have their own attribute "Temperature" containing value of temperature, which is necessary for supporting indoors. Entity "Lighting system" also has its own attributes: "Level of illumination", "Light sources". Entity "Security system" includes system of protection from leakings and ignitions, system of automatic fire extinguishing and video surveillance system. In addition to inheriting from the entity "Life-support system" attributes this entity has its own attribute "State of security", which can take one of two values: "There is a safety violation" or "Violations of safety is not present".

Abstract entity "Sensor" is the parent for entities corresponding to all types of system sensors. It has the following attributes: "Name", "Manufacturer" and "Cost".

Entities "Motion sensor" and "Closing/opening sensor" in addition to inherited attributes have their own attribute "State", which detects movement in the room, closing/opening of the object.

Entity "Leakings sensor" corresponds to a sensor, which is created to detect emergency situations associated with water or gas leakings. This entity has its own attribute "Pressure level".

Entity "Fire sensor" in addition to parent's attributes has its own attribute "Sensitivity", which determines the level of sensor sensitivity to smoke blanketing and temperature changing.

Entity "Temperature and humidity sensor" presents a device, which is responsible for readings of temperature and level of humidity. This entity has its own attributes "Temperature" and "Humidity level".

Entity "System management tool" defines one of the devices, which allow the user to send commands to the central control panel and to inspect the operation of the system. This entity has the following attributes: "Name", "Tool Type" (remote, voice, touch panel), "Manufacturer" and "Cost".

Entity "Central panel" describes the central element of the "Smart House" system, it receives all necessary information from sensors, and it is the center of management of all system components. With a central panel the user interacts through "System management tool". Attributes of this entity are "Manufacturer", "Cost" and "System Components".

After describing of all entities of a metamodel it is necessary to define the relationships between them. The metamodel contains following associations:

– "Send information" is the unidirectional relationship connecting the abstract entity "Sensor" with concrete entity "Central panel".
– "Interact" is the bidirectional relationship describing the interaction of the abstract entity "Life-support system" and concrete entity "Central panel".
– "Fulfill control" is the unidirectional relationship connecting the entities "System management tool" and "Central panel".

In addition to associations the metamodel contains nine inheritance relationships "Is".

Fig. 5 shows one of many possible models of "Smart House" system, constructed in MetaLanguage system with the usage of designed DSML.

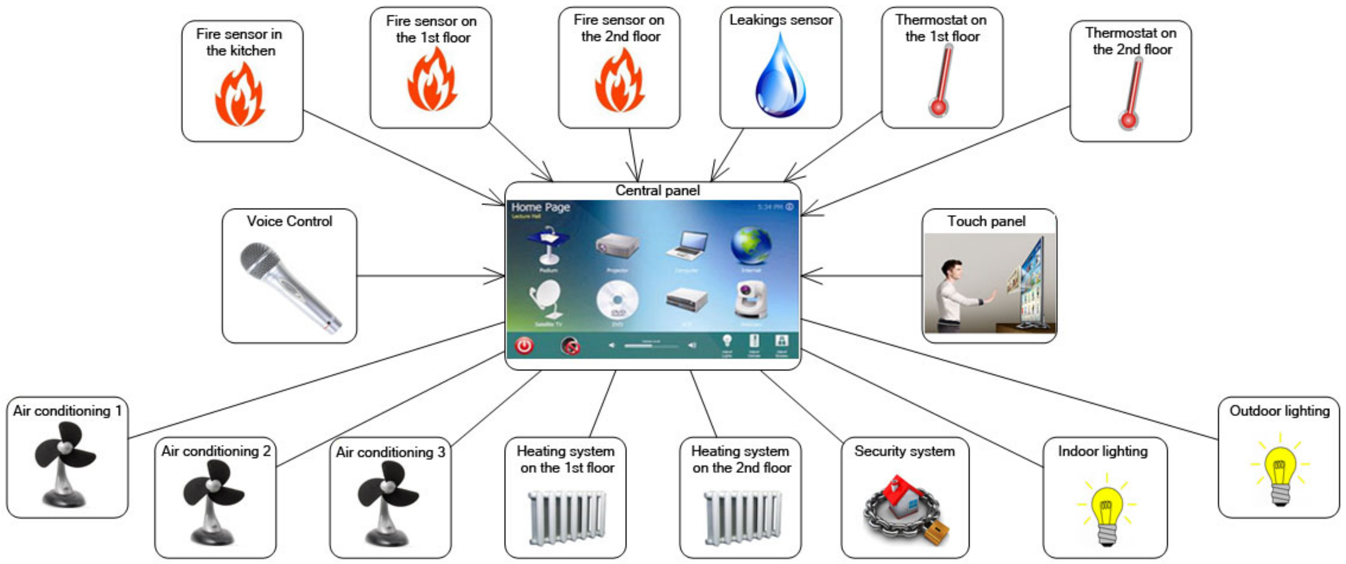
Fire sensor in the kitchen
Fire sensor on the 1st floor
Fire sensor on the 2nd floor
Leakings sensor
Thermostat on the 1st floor
Thermostat on the 2nd floor
Central panel
Home Page
Voice Control
Touch panel
Air conditioning 1
Air conditioning 2
Air conditioning 3
Heating system on the 1st floor
Heating system on the 2nd floor
Security system
Indoor lighting
Outdoor lighting

Fig. 5. Model of "Smart House" system

VI. Conclusion

Approaches to development of tools for visual domain-specific modeling languages creation are considered.

The MetaLanguage system allows to describe domain-specific languages, to create models with their usage and to fulfill transformations of the created models in other textual and graphical notations.

This language workbench is simple to use, therefore not only professional programmers, but also domain experts, for example, business analysts, can work with this tools.

For unified models creation the mathematical model – graph grammars based on pseudo-metagraphs – was constructed. This formalism has allowed to describe basic elements of metalanguage and algorithms, which are used at its functioning: algorithms for creation/modification of domain metamodels and models, algorithms for vertical and horizontal models transformations, algorithms for constraints checking.

The MetaLanguage system was approved at the creation of DSMLs and models for several domains (administrative regulations, queuing systems, etc.).

References

[1] R. France, B. Rumpe, "Model-driven development of complex software: a research roadmap," in *Proc. of the Workshop on the Future of Software Engineering*, Washington, 2007, pp. 37–54.

[2] J. Hutchinson, M. Rouncefield, J. Whittle, "Model driven engineering practices in industry," in *Proc. of the 33rd International Conference on Software Engineering*, New York, 2011, pp. 633–642.

[3] W. J. Dzidek, E. Arisholm, L. C. Briand, "A realistic empirical evaluation of the costs and benefits of UML in software maintenance," *IEEE Transactions on Software Engineering*, vol. 34, pp. 407–432, 2008.

[4] M. Velter. (March 2011). MD*/DSL best practices Update March 2011. [Online]. Available: http://www.voelter.de/data/pub/DSLBestPractices-2011Update.pdf.

[5] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, E. Neema, "Developing applications using model-driven design environments," *Computer*, vol. 39, pp. 33–40, 2006.

[6] M. Erwig, E. Walkingshaw, "A DSL for explaining probabilistic reasoning," in *Proc. of the 2nd International Conference on Software Language Engineering*, Berlin, 2009, pp. 164–173.

[7] R. Walter, M. Masuch, "PULP scription: a DSL for mobile HTML5 game applications," in *Proc. of the 11th International Conference on Entertainment Computing*, Berlin, 2012, pp. 504–510.

[8] M. Fowler. (June 2005). Language workbenches: the killer-app for domain specific languages? [Online]. Available: http://www.martinfowler.com/articles/languageWorkbench.html.

[9] S. Kelly, "Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM," in *Proc. of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications at OOPSLA 2004*, Portland, 2004, pp. 87–96.

[10] T. Ozgur, "Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for domain-specific modeling in the context of the model-driven development," master thesis, Karlskrona, Blekinge Institute of Technology, 2007.

[11] J. Karna, J.-P. Tolvanen, S. Kelly, "Evaluating the use of domain-specific modeling in practice," in *Proc. of the 9th Workshop on Domain-Specific Modeling at OOPSLA 2009*, Orlando, 2009, pp. 147–153.

[12] J.-P. Tolvanen, R. Pohjonen, S. Kelly, "Advanced tooling for domain-specific modeling: MetaEdit+," in *Proc. of the 7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007*, Montreal, 2007, pp. 48–55.

[13] S. Cook, G. Jones, S. Kent, A. C. Wills, *Domain-specific development with Visual Studio DSL Tools*. Reading. Addison-Wesley, 2007, 560 p.

[14] R. C. Gronback, *Eclipse modeling project: a domain-specific language toolkit*. Reading: Addison-Wesley, 2009, 706 p.

[15] A. N. Terekhov, T. A. Bryksin, YU. V. Litvinov, "QReal: platform of visual domain-specific modeling," *Software engineering*, vol. 6, pp. 11–19, 2013.

[16] A. Basu, R.W. Blanning, "Graphs, hypergraphs, and metagraphs," in *Metagraphs and Their Applications*. New York: Springer US, pp. 1–12, 2007.

[17] B. Courcelle, "Recognizable sets of graphs, hypergraphs and relational structures: a Survey Developments in Language Theory," in *Lecture Notes in Computer Science*, pp. 1–11, 2005.

[18] J. Power, K. Tourlas, "Abstraction in reasoning about higraph-based systems: foundations of software science and computation structures," in *Lecture Notes in Computer Science*, pp. 392-408, 2003.

[19] A. O. Sukhov. (March 2012). Analysis of formalisms for visual modeling languages description. *Modern Problem of Science and Education*. [Online]. Vol. 2. Available: http://www.science-education.ru/102-5655.