Generation of Domain-Specific Languages on the Basis of Ontologies

Alexander O. Sukhov
Department of Business Informatics
National Research University Higher School of Economics
Perm, Russian Federation
E-mail: Sukhov.psu@gmail.com

***Abstract* — Usage of visual domain-specific languages in software engineering allows to simplify the process of software creation and to attract to it the experts in domain, who are not professional programmers. However creation new domain-specific language is the nontrivial task, therefore the problem of automation of their development process is the topical task. For the automation, designing of visual modeling languages it is offered to use the ontologies received as a result of the analysis of text corpus. In article, the approach to automatic creation of visual modeling languages on the basis of domain ontologies is considered.**

Keywords — domain-specific modeling languages; ontologies; DSM-platform; MetaLanguage; metagraphs.

I. INTRODUCTION

The domain-specific modeling languages (DSMLs), which are designed to solve a particular class of problems in a specific domain, are increasingly used at software development and maintenance process. Unlike the general-purpose languages, DSMLs are more expressive, easy to use and clear to various categories of users, as they operate with domain terms, which are familiar to users [1, 2]. 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. [3-7].

Despite of all DSMLs advantages they have one big disadvantage – complexity of their designing. If general-purpose languages allow to create models irrespectively to domain, in case of DSMLs for each domain, and in some cases, for each task, it is necessary to create a 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. Therefore, a problem of automation of DSMLs development process is rather topical.

To support the process of development and maintenance of DSMLs the special kind of software – *language workbench* (*DSM-platform*) is used. 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. However, these tools do not allow to automatic create DSMLs.

This problem can be solved by the development of methods and tools, which will allow on the basis of a set of documents, available in domain, to build a conceptual domain model and automatically design a modeling language, which corresponds to singularities of domain, needs of various categories of users [8].

Let's consider the most advanced language workbenches [9].

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 [10]. 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.

DSL Tools [11] and Eclipse GMF [12] technologies provide the user with advanced IDE MS Visual Studio and Eclipse respectively. State Machine Designer [13], in fact, is an add-on to DSL Tools, which eliminates some of its shortcomings. However, the State Machine Designer allows creating DSMLs only using UML Activity Diagrams that considerably limits the range of solving tasks.

Multiplatform system QReal [14, 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 in one row with tools MS DSL Tools, Eclipse GMF, which use for these purposes IDE.

This paper is supported by the Russian Foundation for Basic Research (grant 14-07-31330).

The analysis of DSM-platforms has revealed the following main restrictions inherent in the majority of the considered systems [16]:

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 transformation. 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.

Moreover, the considered tools do not allow to fulfill automatic construction of DSMLs on the basis of domain analysis. This possibility allows to:

- simplify the process of DSMLs creating;
- create DSMLs, approached to specific domain;
- attract to DSMLs development process the users, who are not the professional programmers.

Thus, it is possible to say that creation of methods and tools of automatic DSMLs creation is the topical task, the solution of which will significantly simplify the process of visual domain-specific languages designing for various domains, and also will submit a possibility of involvement the experts in the process of development and maintenance of information systems.

For development of methods and tools for automatic of visual DSMLs designing, it is necessary to solve the following tasks:

1. Construct a mathematical model, which will allow to unify describe domain ontologies and metamodels of visual languages.
2. Develop rules of ontologies transformation in constructions of visual modeling languages.
3. Develop metrics and methods of comparing of DSMLs, which submit a possibility to estimate proximity of generated automatically domain-specific languages to domain specificity.
4. Implement the developed methods in dynamic library.
5. Integrate the created library into the MetaLanguage system.
6. Perform approbation of the received results by development of visual modeling languages for various domains.

III. METALANGUAGE SYSTEM

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

This language workbench is designed to create visual dynamic adaptable domain-specific modeling languages, construct models using these languages and transform of the created models in various textual and graphical notations [17, 18].

A. Metalanguage of MetaLanguage System

One of the basic elements of language workbench is the *metalanguage* (meta-metamodel) – language for describing of other languages (metamodels). Thanks to presence of metalanguage, the DSM-platform allows to create domain-specific languages for the various domains that operate with familiar to user concepts. The main difference between metalanguages 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 dynamic, during models creation. Besides, the process of metamodel creation becomes multilevel, so having defined a metamodel and having selected it as a metalanguage, the developer can use this meta-metamodel for creation of other metamodels, and this process can be infinite.

The basic elements of the metalanguage of MetaLanguage system are entity, relationship and constraint [19].

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. 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 between them. Some of constraints are set by metamodel structure, and others are described on some language.

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

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. The attribute of entity "Actor" is a string attribute "Name", which specifies the name of the Actor.

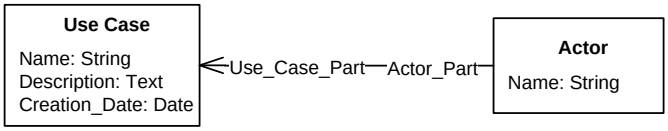

Fig. 1. Fragment of metamodel for UML Use Case diagrams

B. Architecture of MetaLanguage System

The architecture of MetaLanguage system 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 a single tool. 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.

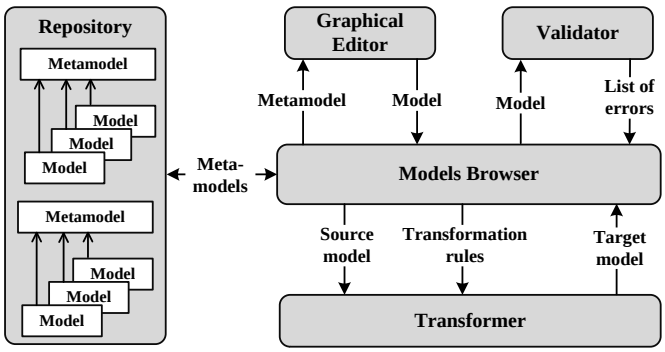

Fig. 2. Architecture of MetaLanguage system

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

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 using created DSML, the user should design models containing objects that describe specific domain concepts and links between them.

The validator should check up whether model satisfies 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.

For automatic creation of DSMLs, it is necessary that the language workbench on the basis of domain description fulfilled creation of language's metamodel. Since the structure of the metalanguage of MetaLanguage system is similar to ontologies description languages, then it was decided to use ontologies as the basis of automatic DSMLs designing. There are various systems for the automatic construction of ontologies on the basis of a text corpus: OwlExporter [20], OntoGrid [21], etc. These systems allow to fulfill ontology creation on the basis of initial set of documents. The resulting ontology will be used by the MetaLanguage system for automatic creation of visual DSMLs (see Fig. 3).

IV. Automatic Creation of DSMLs

Formally ontology is the tuple $O = \{T, R, I\}$, where

- T is the finite nonempty set of domain concepts;
- R is the finite set of relationships between concepts;
- I is the finite set of interpretation of ontology concepts and relationships.

Thus, it is possible to say that the ontology is a directed labeled graph. The metamodel of visual modeling language is a graph also. For this reason the formalism that allows to describe domain ontology and metamodel of visual DSML in unified form is a labeled graph.

There are several types of graphs that can be used for representation of visual languages and ontologies: the classical graphs, digraphs, multigraphs, pseudographs, hi-graphs, hypergraphs, metagraphs and others.

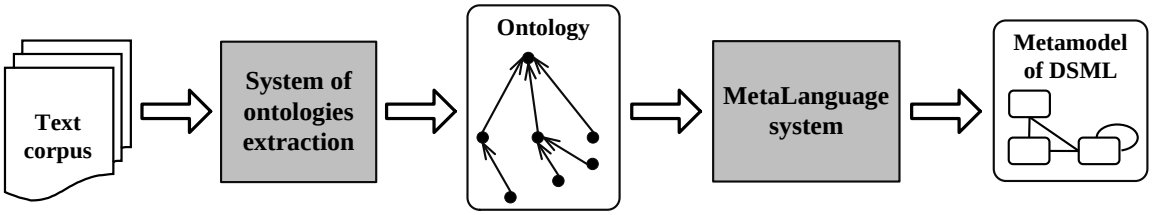

Fig. 3. Automatic creation of metamodel of DSML

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 are pseudo-metagraphs [22].

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.

This type of graphs allows to reduce the number of graph arcs and to make the model more structured, logical. Therefore, in the metamodel graph the attributes and constraints of each entity and relationship are united in the separate sets. The model becomes more demonstrative, clear and corresponding to the solving task.

A. Metamodel Graph

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

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 $Rel = \{rel_k\}, k \in \aleph$, number of set elements is potentially unlimited, but at every fixed point in time is finite.

Let's introduce the following designations:

- $EAttr_i = \{eattr_{i_j}\}, \overline{i = 1, |Ent|}, j \in \aleph$ is the set of metamodel graph nodes, which corresponds to entities attributes;
- $RAttr_k = \{rattr_{k_l}\}, \overline{k = 1, |Rel|}, l \in \aleph$ is the set of metamodel graph nodes, which corresponds to relationships attributes;
- $ERest_i = \{erest_{i_j}\}, \overline{i = 1, |Ent|}, j \in \aleph$ is the set of metamodel graph nodes, which corresponds to constraints imposed on entities;
- $RRest_k = \{rrest_{k_l}\}, \overline{k = 1, |Rel|}, l \in \aleph$ is the set of metamodel graph nodes, which corresponds to constraints imposed on relationships;
- $EEA = \{eea_i\}, \overline{i = 1, |Ent|}$ is the set of metamodel graph arcs connecting each entity with the set of its attributes;
- $ERA = \{era_k\}, \overline{k = 1, |Rel|}$ is the set of metamodel graph arcs connecting each relationship with the set of its attributes;
- $EER = \{eer_i\}, \overline{i = 1, |Ent|}$ is the set of metamodel graph arcs connecting each entity with the set of its constraints;
- $ERR = \{err_k\}, \overline{k = 1, |Rel|}$ is the set of metamodel graph arcs connecting each relationship with the set of its constraints;
- $EERR = \{eerr_i\}, i \in \aleph$ 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):

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

$$E = EEA \bigcup EER \bigcup ERA \bigcup ERR \bigcup EERR. \quad (2)$$

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 a fragment of the "Use Case" entity is shown in Fig. 4.

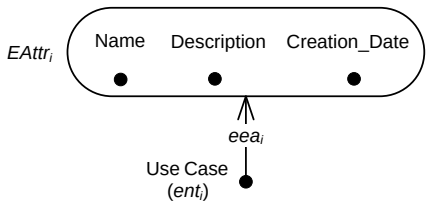

Fig. 4. Fragment of metamodel graph for "Use Case" entity

As can be seen from the figure

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

B. Ontology Graph

Let's introduce the following designations:

- $T = \{t_i\}, i \in \aleph$ is the set of ontology graph nodes, which corresponds to concepts of ontology;
- $OAttr_i = \{oattr_{i_j}\}, \overline{i = 1, |T|}, j \in \aleph$ is the set of ontology graph nodes, which corresponds to concept attributes, which are not relationships;
- $OInst_i = \{oinst_{i_j}\}, \overline{i = 1, |T|}, j \in \aleph$ is the set of ontology graph nodes, which corresponds to concept instances;

- $ORel = \{orel_i\}, i \in \aleph$ is the set of ontology graph nodes corresponding to relationships between concepts.
- $OA = \{oa_i\}, \overline{i = 1, |T|}$ is the set of ontology graph arcs connecting each concept with the set of its attributes;
- $OI = \{oi_i\}, \overline{i = 1, |T|}$ is the set of ontology graph arcs connecting each concept with the set of its instances;
- $TR = \{tr_i\}, i \in \aleph$ is the set of ontology graph arcs corresponding to links between concepts and relationships.

The number of elements of sets *T*, $OAttr_i$, $OInst_i$, *ORel*, *OA*, *OI*, *TR* potentially is not limited, but it is finite at every fixed point in time.

The *ontology graph* is the directed pseudo-metagraph $GO = (VO, EO)$, where *VO* is a nonempty set of graph nodes, *EO* is set of graph arcs and these sets are defined by (3) and (4):

$$VO = T \bigcup \left(\bigcup_{i=1}^{|T|} \left(OAttr_i \bigcup OInst_i \right) \right) \bigcup ORel , \quad (3)$$

$$EO = OA \bigcup OI \bigcup TR . \quad (4)$$

C. Mapping of Ontology Graph to Metamodel Graph

At mapping of a domain ontology to a modelling language metamodel it is necessary to fulfill the following algorithm:

1. To eliminate synonymy (to merge nodes containing synonymic concepts).
2. To delete instances of concepts and "is instance" relationships.
3. For each concept of ontology to create in a metamodel an entity with the concept's attributes.
4. For each "is a" relationship of ontology to create an inheritance relationship in a metamodel.
5. For each "is part of" relationship of ontology to create an aggregate relationship in a metamodel.
6. For other relationships of ontology to create an association relationship in a metamodel.

According to this algorithm, let's determine the mapping of the ontology graph to the metamodel graph, this mapping corresponds to operation of automatic creation of the metamodel graph.

Let's introduce the following designations:

- $OInh = \{oinh_i\}, i \in \aleph$ is the set of ontology graph nodes corresponding to "is a" relationships between concepts;
- $OAggr = \{oaggr_i\}, i \in \aleph$ is the set of ontology graph nodes corresponding to "is part of" relationships between concepts;
- $OInstR = \{oinstr_i\}, i \in \aleph$ is the set of ontology graph nodes corresponding to "is instance" relationships between concepts;
- $RInh = \{rinh_i\}, i \in \aleph$ is the set of metamodel graph nodes corresponding to inheritance relationships;
- $RAggr = \{raggr_i\}, i \in \aleph$ is the set of metamodel graph nodes corresponding to aggregation relationships;
- $RAssoc = \{rassoc_i\}, i \in \aleph$ is the set of metamodel graph nodes corresponding to association relationships between concepts.

The number of elements of sets *OInh*, *OAggr*, *OInstR*, *RInh*, *RAggr*, *RAssoc* potentially is not limited, but it is finite at every fixed point in time.

The mapping $conc : T \to Ent$ for each ontology concept puts in correspondence a metamodel entity.

The mapping $attr : OAttr \to EAttr$ for each ontology concept's attribute puts in correspondence a metamodel entity's attribute.

The mapping $inh : OInh \to RInh$ for each ontology "is a" relationship puts in correspondence a metamodel inheritance relationship.

The mapping $aggr : OAggr \to RAggr$ for each ontology "is part of" relationship puts in correspondence a metamodel aggregation relationship.

The mapping $assoc : ORel / \left(OInh \bigcup OAggr \bigcup OInstR \right) \to RAssoc$ for each other ontology relationship puts in correspondence a metamodel association relationship.

The mapping $ar : OA \to EEA$ for each arc oa_i of ontology graph puts in correspondence an arc eea_i of metamodel graph.

The mapping $rr : TR \to EERR$ for each arc tr_i of ontology graph puts in correspondence an arc $eerr_i$ of metamodel graph.

Thus, the creation of the metamodel graph on the basis of ontology graph is determined by mappings *conc*, *attr*, *inh*, *aggr*, *assoc*, *ar*, *rr*.

D. "Smart House" Description Language

Let's consider an example. Suppose that it is necessary to construct a visual modeling 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 a central panel):

motion, leakings, fire and a smoke, closing/opening of object;

- system management tools: voice control, remote control (from a remote computer, 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.

The ontology received as a result of the analysis of a text corpus is presented in Fig. 5.

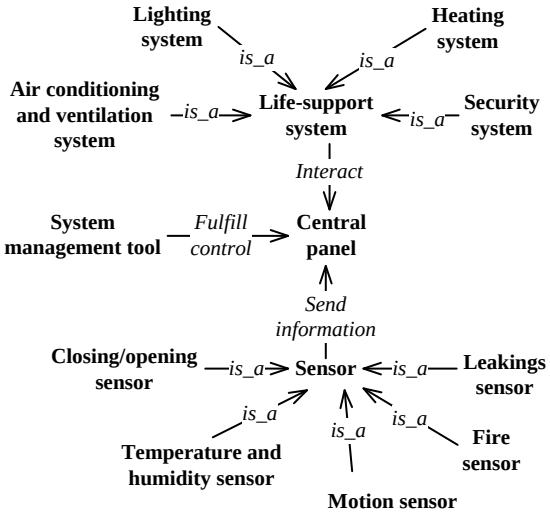

Fig. 5. Ontology of the strucure of "Smart House" system

Concept "Life-support system" is the parent for concepts "Heating system", "Air conditioning and ventilation system", "Lighting system", "Security system". Concept "Sensor" is the parent for concepts corresponding to all types of system sensors. The relationship "Send information" connects the concept "Sensor" with the concept "Central panel". The relationship "Interact" describes the interaction of the concepts "Life-support system" and "Central panel". The relationship "Fulfill control" connects the concepts "System management tool" and "Central panel".

In this case, according to the considered earlier algorithm, the MetaLanguage system has constructed the metamodel presented in Fig. 6.

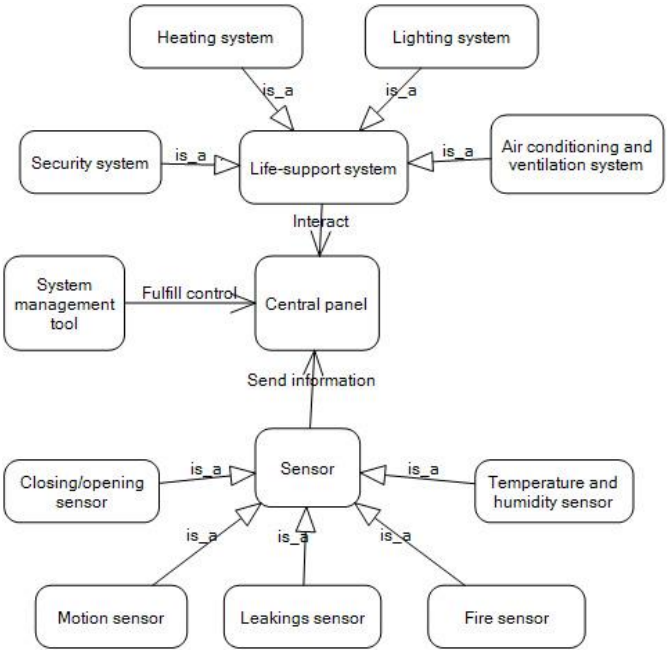

Fig. 6. Metamodel of "Smart House" Description Language

Fig. 7 shows one of many possible models of "Smart House" system, constructed in MetaLanguage system with the usage of designed DSML.

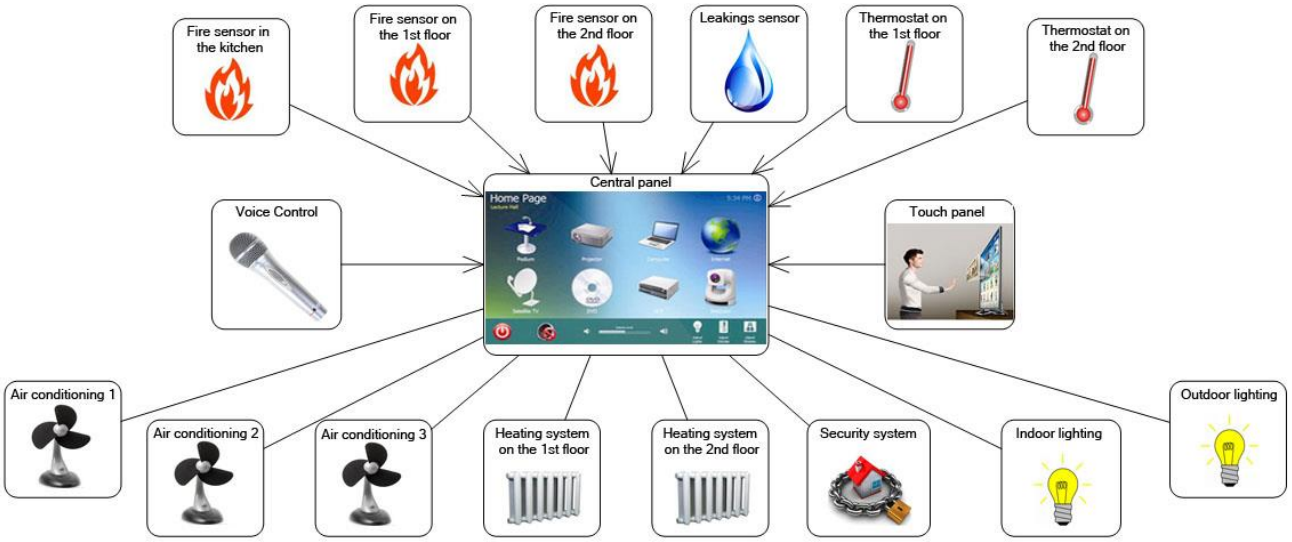

Fig. 7. Model of "Smart House" system

V. Conclusion and Future Works

The DSMLs creation is a difficult task. To automate the designing of visual modeling languages the approach, allowing on the basis of domain ontology to generate a language description, is offered. It allows reducing complexity and time of DSMLs development, and gives the chance to nonprofessional programmers to develop their own languages. In the future, it is planned to develop metrics and methods of comparing of DSMLs, which submit a possibility to estimate proximity of generated automatically domain-specific languages to domain specificity and to generate visual DSMLs for other domains.

References

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

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

[3] Bryksin T.A., Litvinov YU.V. Sreda vizual'nogo programmirovaniya robotov QReal: Robots. Materialy mezhdunarodnoj konferentsii “Informatsionnye tekhnologii v obrazovanii i nauke”, 2011, pp. 332-334 (in Russian).

[4] Erwig M., Walkingshaw E. A DSL for Explaining Probabilistic Reasoning. Proceedings of the 2nd international conference on Software Language Engineering, 2009, pp. 164-173.

[5] Mezhuev V.I. Predmetno-orientirovannoe modelirovanie raspredelennykh prilozhenij real'nogo vremeni. Sistemy obrabotki informatsii, 2010, no. 5(86), pp. 98-103 (in Russian).

[6] Walter R., Masuch M. PULP Scription: A DSL for Mobile HTML5 Game Applications. Proceedings of the 11th International Conference on Entertainment Computing, 2012, pp. 504-510.

[7] Sukhov A.O. Integratsiya sistem imitatsionnogo modelirovaniya i predmetno-orientirovannykh yazykov opisaniya biznes-protsessov. Matematika programmnykh sistem, 2009, vol. 6, pp. 79-84 (in Russian).

[8] Sukhov A.O. Razrabotka predmetno-orientirovannykh yazykov na osnove ontologij. Sbornik tezisov konferentsii “Sovremennye problemy matematiki i ee prikladnye aspekty”, 2013, pp. 45-45 (in Russian).

[9] Sukhov A.O. Sravnenie sistem razrabotki vizual'nykh predmetno-orientirovannykh yazykov. Matematika programmnykh sistem, 2012, vol. 9, pp. 84-111 (in Russian).

[10] Tolvanen J.-P., Pohjonen R., Kelly S. Advanced Tooling for Domain-Specific Modeling: MetaEdit+. Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA, 2007, pp. 48-55.

[11] Cook S., Jones G., Kent S., Wills A.C. Domain-Specific Development with Visual Studio DSL Tools. Reading: Addison-Wesley, 2007, 560 p.

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

[13] Larionov A.V. Razrabotka vizual'nogo yazyka avtomatnogo programmirovaniya. Available at: http://is.ifmo.ru/papers/StateMachineDesigner.pdf (accessed 3 April 2014) (in Russian).

[14] Terekhov A.N., Bryksin T.A., Litvinov YU.V. QReal: platforma vizual'nogo predmetno-orientirovannogo modelirovaniya. Programmnaya inzheneriya, 2013, no. 6, pp. 11-19 (in Russian).

[15] Terekhov A.N., Bryksin T.A., Litvinov YU.V., Smirnov K.K., Nikandrov G.A., Ivanov V.YU., Takun E.I. Arkhitektura sredy vizual'nogo modelirovaniya QReal. Sistemnoe programmirovanie, 2009, vol. 4, pp. 171-196 (in Russian).

[16] Sukhov A.O. Sravnenie sistem razrabotki vizual'nykh predmetno-orientirovannykh yazykov. Matematika programmnykh sistem, 2012, no 9, pp. 84-111 (in Russian).

[17] Sukhov A.O., Lyadova L.N. MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages. Proceedings of the 6th Spring/Summer Young Researchers’ Colloquium on Software Engineering, 2012, pp. 42-53.

[18] Sukhov A.O. Instrumental'nye sredstva sozdaniya vizual'nykh predmetno-orientirovannykh yazykov modelirovaniya. Fundamental'nye issledovaniya, 2013, no 4, vol. 4, pp. 848-852 (in Russian).

[19] Lyadova L.N., Sukhov A.O. Yazykovoj instrumentarij sistemy MetaLanguage. Matematika programmnykh sistem, 2008, vol. 5, pp. 40-51 (in Russian).

[20] Witte R., Khamis N., Rilling J. Flexible Ontology Population from Text: The OwlExporter. Available at: http://www.lrec-conf.org/proceedings/lrec2010/pdf/932_Paper.pdf.

[21] Gusev V.D., Zavertajlov A.V, Zagorujko N.G., Kovalyov S.P., Nalyotov A.M., Salomatina N.V. System “OntoGrid” for construction of ontologies. Available at: http://www.dialog-21.ru/Archive/2005/Zagoruiko%20Gusev%20Zavertailov/ZagoruykoNG.htm (in Russian).

[22] Sukhov A.O. Analiz formalizmov opisaniya vizual'nykh yazykov modelirovaniya. Sovremennye problemy nauki i obrazovaniya, 2012, no 2. Available at: http://www.science-education.ru/102-5655 (in Russian).