An Integration of Modeling Systems Based on DSM-platform

Lyudmila N. Lyadova, Alexander O. Sukhov, Elena B. Zamyatina

***Abstract* —** An approach of using of the DSM-platform MetaLanguage for integration of various modeling systems is presented. This tool allows to design visual domain-specific modeling languages and to create domain models with developed languages. The MetaLanguage system includes components for describing transformations of models from one formal notation to another. Domain-specific modeling permits various specialists to use concepts from different domains at creating and analyzing of models. An integration of DSM-platforms with tools of models analysis allows to involve domain experts, end-users in the process of constructing and analyzing of models; to reduce the complexity of models development; to fulfill research of models from various points of view with usage of various methods and tools.

***Keywords* —** Domain-specific languages, language workbench, modeling languages, models transformation, simulation.

I. Introduction

Information and analytical systems, which are used for solution of various management tasks, are created with technologies, which are based on the models. Mainly for models creation graphical notations, diagrams of various types are used. These notations and diagrams allow to describe objects of the modeled business system, their properties, relations between them, operations executed over them, business processes, etc.

The important conditions for reducing the complexity of users work are the possibility of integration of various information systems, the reusing of created models, and their transfer from one system to another for solving of various tasks. The transformation of models from one modeling language to another can be required [1].

These requirements can be implemented on the basis of creation of *domain-specific modeling* (DSM) tools, which are called the *DSM-platforms* (*language workbenches*), the main purpose of these tools is development of high-level *domain-specific languages* (DSLs), designed to create models of systems, focused on solving problems in various domains [2], [3], [4]. The language workbench can become the basis for integration of various tools intended for development of information systems, based on the created models (CASE-tools), and for systems analysis (in particular, simulation systems) [5]. At DSLs usage not only domain singularities, but also qualification of users can be considered.

Maximal flexibility of modeling tools may be obtained at creating the multilevel models describing the modeled systems from various points of view and with different levels of details. For matching 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 characteristics that are important from the point of view of the modeling purpose, *metamodel* is a model of the language, which is used for models development, and *meta-metamodel* (*metalanguage*) is a language, on which metamodels are described [6].

As a part of the delivered problem the complex of tasks on creation of DSM-platform, which satisfies the following *requirements*, should be solved:

- possibility of modeling languages constructing for a wide range of domains;
- possibility of multi-level modeling (it allows to modify the metalanguage description, to extend it with new constructions, thus approaching the metalanguage to the specificity of domain);
- possibility of modification of modeling language description without regeneration of source code of DSLs editor;
- automatic support in a consistent state of the metamodels and models description at modification of a metalanguage or a metamodel;
- uniformity of tools of representation, description and usage as models and metamodels: creation of models at different levels of hierarchy and operation with them should be carried out uniformly, using the same tools;
- availability of tools for models transformations that allow to convert models as between different levels of the hierarchy, and within the same level (between various modeling languages);
- usability of language toolkits for various categories of users: professional developers (programmers, system analysts, date base designers, etc.), domain experts, business analysts, end-users.

This work was supported in part by Russian Foundation for Basic Research (grants 14-07-31330, 12-07-00302, 13-07-96506).

E. B. Zamyatina is with the National Research University Higher School of Economics, Perm, Russia (e-mail: E_Zamyatina@mail.ru).

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

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).

II. Development of Models in MetaLanguage

All the possibilities and demands mentioned above are not realized in any language workbench nowadays [7]. But DSM-platform MetaLanguage attempts to overcome these disadvantages. The MetaLanguage system 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.

One of the basic elements of language workbench is the metalanguage. The *basic elements* of the metalanguage of the presented system are entity, relationship and constraint [8]. 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. The *relationship* is used for describing a physical or conceptual links between entities. The Metalanguage system allows to create three types of relationships: *association*, *aggregation*, *inheritance*. The *constraints* define the rules of models constructing. The constraints are defined for the entities and relationships between them.

Let's describe the process of building models using the MetaLanguage system (see Fig. 1).

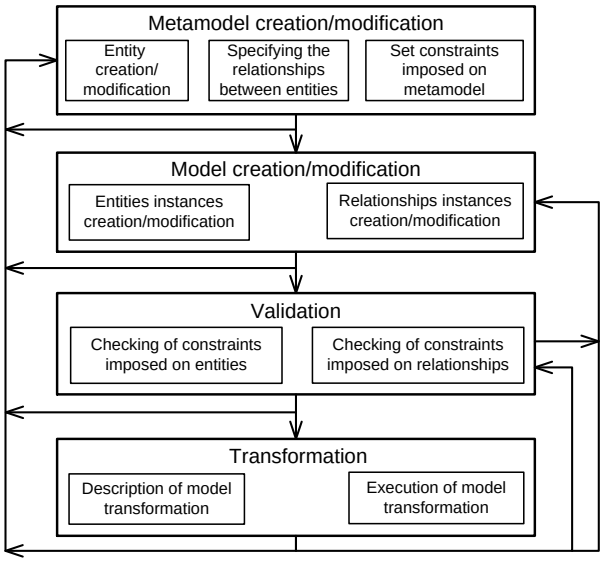


Fig. 1. Process of creation/modification of domain models with MetaLanguage system usage

The first stage supposes *developing of a metamodel*. The metamodel is a domain-specific language intending to solve specific problems of analyzing domain. Metamodel developers have to use a model's editor. The developer obtains an extensible dynamically customizable visual modeling language as a result of metamodel creation.

Then users (model designers, system analysts) can *develop models*, which contain instances of specific entities and relations between them, with application of constructions of created DSL. Thereafter, it is necessary to *validate the developed model*: to check if all constraints for the entities and relations between them are met.

A developer can *store designed models in repository*.

User can *transform model* in accordance to rules defined in system [9]. So the designed model can be translated to one or other languages and can be exported to external program systems (simulation system, CASE-tool, for example).

Developed DSL can be used as a metalanguage. The whole *hierarchy of languages* can be created on its basis. This hierarchy allows to work with models of various abstraction levels, focused on solving of various tasks by different categories of users in terms of their domain [10]. At metamodel modification, the system automatically will make all necessary changes in models created on the basis of this metamodel.

Different categories of users are involved in development of domain-specific languages and creation of models with their usage (see Fig. 2).

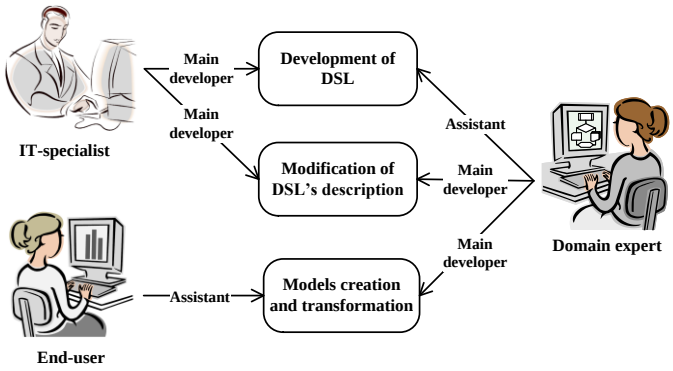


Fig. 2. Involvement of various categories of users in the development of DSLs and models

So developers (IT specialists) with the direct participation of domain experts create DSL, describing the basic concepts of the domain, relationships between them and the constraints, imposed on a metamodel, define rules of models transformations. Domain experts and end-users with the developed language build domain models and fulfill transformations. If it is necessary to modify modeling language the domain experts can independently make appropriate changes in language description or invite the IT-specialists for performing of all necessary modifications.

III. DSM-platform as a Base of Integration

Creation of information systems with usage of modern CASE-tools is based on development of the various models describing the information system domain, defining data structures and algorithms of system functioning [11]–[14]. The choice of tools frequently determines also a choice of language for models description. Thus, the used tool actually "imposes" to developers and users a specific modeling language, which more often operates with terms of some programming paradigm, therefore tools do not allow to domain experts to participate in development and modification of models, that is a necessary condition for creation of effective management systems, increase of efficiency of their adaptation, reducing of maintenance complexity.

One of approaches to solving of this problem is integration of DSM-platforms with tools of information systems development or directly with the information systems, which fulfill interpretation of models at the stage of functioning. Thus, the DSM-platform can become the basis for integration of various tools intended for development of the information systems on the basis of created models and for the analysis of systems via formal models. So, for example, the MetaLanguage system can be integrated with CASE-tools, business analysis systems, simulation systems (see Fig. 3).

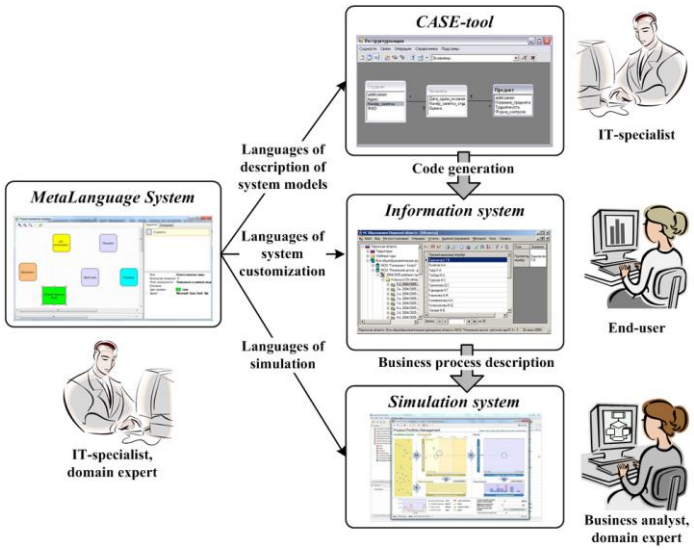


Fig. 3. Scheme of usage of MetaLanguage system in the process of creation and maintenance of information systems

Instead of describing models in the notation of visual general-purpose languages, experts with usage of MetaLanguage system can develop DSLs for creation and maintenance of models. After designing of domain-specific languages, the developers with the participation of domain experts create models of information systems. For export of designed models to CASE-tool, it is necessary preliminarily to fulfill conversion of model description to one of standard modeling languages, supported by this tool.

If the developed product is an information system with interpretation of models, the DSM-platform can be used even at the stage of system functioning, so end-users can modify description of domain model, business processes developed with usage of DSLs to adapt system for new conditions.

For business processes analysis and optimization the simulation systems can be used. For carrying out simulation experiments, it is necessary to transform business processes models, created with usage of DSLs or with notation of other modeling languages, in graphical/textual notation, supported by simulation system. After research of business processes their reengineering with usage of domain-specific languages, created in the MetaLanguage system, without source code regeneration and participation of IT-experts can be fulfilled.

Thus, the MetaLanguage system can be used both at the process of development and maintenance of information systems, and as the extension of systems analysis tools.

IV. Integration of MetaLanguage System with Simulation Systems

It is known that in some cases, simulation is a single method of research of complex dynamic systems, and it is widely applied in various fields of science and industry. Development of science and technologies, and, hence, increase of complexity of researched systems, puts more and more complex tasks for simulation. For obtaining of reliable information during simulation experiments it is required to involve experts from different fields of knowledge, and therefore, simulation software should allow to researchers to work in various modeling environments, using different systems of concepts, varied visual or textual languages. For example, at business processes modeling, researchers can attract graph theory, Petri nets or queuing networks. In this case, it is necessary, that a modeling system submitted the user possibility of using not only of various mathematical apparatus, but also of various modeling languages, which operate with terms clear to various categories of users.

Queuing networks (QN) are widely used to analyze the characteristics of business systems in various areas. Various methods and tools (statistical analysis, simulation, etc.) are used for research. Let's consider the example: we'll design new DSL for *QN modeling* with MetaLanguage system and then we'll define rules for the model transformation from designed DSL to the GPSS modeling language.

The *metamodel* of this domain-specific language contains following *entities* (see Fig. 4):

- *Generator* is the entity, which is responsible for generation of requests flow (*transacts* flow), expecting service in system. Intervals between requests arrivals are the random values with a certain distribution. This entity has the following attributes "Name", "Initial delay", "Amount of transactions", "Priority".
- *Queue* is the entity, representing set of transacts, which expect service. It is waiting buffer of the servicing device if it is occupied. The entity "Queue" has the following attributes: "Name", "Maximum length" and "Current length".
- *Servicing device* is the entity, which is responsible for service of requests. The device possesses limited possibilities of transacts service. Handling of request takes some time. The service time is a random value with a certain distribution function. The attributes of the entity are "Name", "Amount of channels", "Service time".
- *Separator* is the entity, allowing to create multiple copies of transacts, each of which will request claim of service. The attributes of this entity are "Name", "Amount of copies", "Block" (name of the block, to which it is necessary to transmit copy of request for service).
- *Collector* is the entity, allowing to integrate multiple transacts flows into a single flow. The entity "Collector" has the attributes "Name" and "Amount of flows".
- *Terminator* is the entity, deleting transacts from model.

- *Distribution* is the entity, which is parent for the entities "Normal distribution", "Uniform distribution", "Student's distribution", etc.
- *Normal distribution* is a distribution, according to which a generation of new requests and/or their service is fulfilled. This entity has two attributes "Expected value", "Variance".
- *Uniform distribution* is a distribution, according to which generation of new requests and/or their service is fulfilled. This entity has two attributes "Minimum value", "Maximum value".
- *Student's distribution* is a distribution, according to which generation of new requests and/or their service is fulfilled. This entity has attribute "Amount of degrees of freedoms".

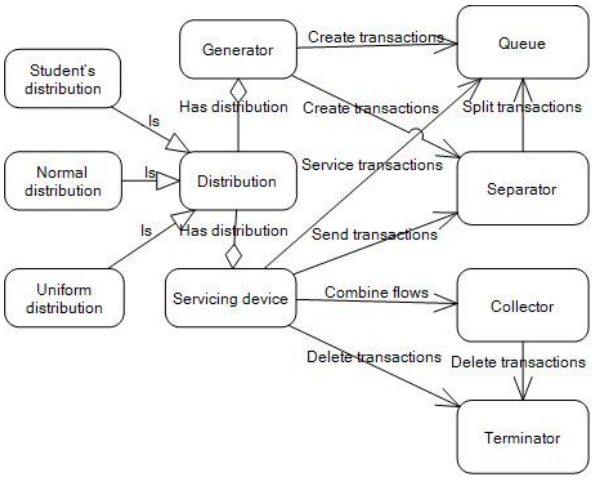


Fig. 4. Simplified metamodel of language for simulation models description

Further, let's describe *relationships* between metamodel entities. As can be seen from Fig. 4, the metamodel contains the following *relationships of association*:

- unidirectional relationship "Create transactions", connecting the entity "Generator" with entities "Queue" and "Separator", shows that after creation of requests they can be placed in a queue for service or be split into multiple flows;
- bidirectional relationship "Service transactions" allows to indicate, what device handles requests in a queue and where they go after service;
- unidirectional relationship "Send transactions", connecting entities "Servicing device" and "Separator", allows to split requests into multiple flows;
- unidirectional relationship "Combine flows" connects entities "Servicing device" and "Collector" and indicates, what collector combines flows of requests for service after their handling by multiple servicing devices;
- unidirectional relationship "Split transactions", connecting entity "Separator" and "Queue", allows to indicate in what queues requests after their separation into several flows should be placed;
- unidirectional relationship "Delete transactions", connecting entities "Servicing device" and "Collector" with entity "Terminator", allows to indicate that after service or combine the transacts should be removed.

The metamodel of modeling language for creation of QN-models also contains three inheritance "Is" that connect the abstract entity "Distribution" with child entities "Normal distribution", "Uniform distribution", "Student's distribution". Child entities inherit all parent entity's relationships.

The aggregation "Has distribution" allows to specify distribution, according to which generation of new requests (transacts) and/or their service is fulfilled.

In Fig. 5 the example of model, constructed with usage of the developed modeling language, is presented. As can be seen from the figure, the model contains generator, four servicing devices (SD1, SD2, SD3, SD4), four queues (QQ1, QQ2, QQ3, QQ4), separator, collector and terminator; two distributions is used. This model describes QN for any domain.

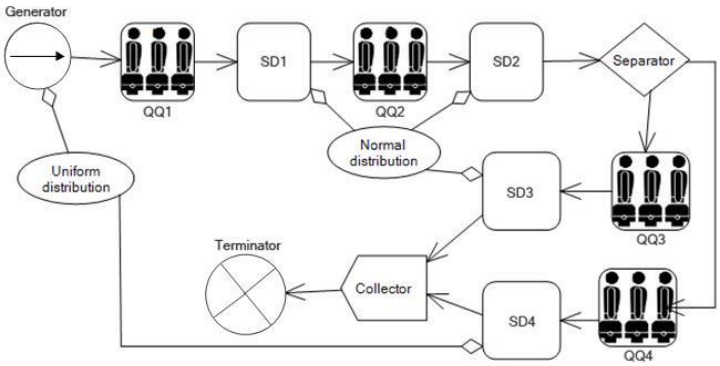
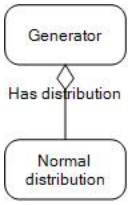


Fig. 5. Exapmle of QN-model

Let's describe the transformation rules of the constructed modeling language to the notation of GPSS language. User can generate the program in GPSS language applying these rules to the constructed model, and then user can make the analysis of model with usage of the simulation system GPSS.

The rule "Generator_Norm", which converts an instance of entity "Generator", connected by an instance of aggregation relationship with an instance of entity "Normal distribution", into the appropriate command of GPSS language, looks like:



→

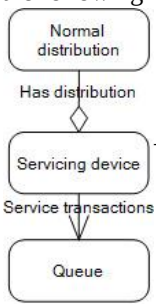
```
GENERATE
    (NORMAL(1,
    <<Normal distribution.Expected value>>,
    <<Normal distribution.Variance>>)), ,
    <<Generator.Initial delay>>,
    <<Generator.Amount of transactions>>,
    <<Generator.Priority>>
```

Symbols "<<" (double opening angle brackets) and ">>" (double closing angle brackets) are used for selection of rule dynamic part, which allows to get values of attributes of entities and relationships instances.

Rules for other types of distributions are described similarly.

The transformation rule "Queue", which converts connected instances of entities "Queue", "Servicing device", "Normal

distribution" into the appropriate code of GPSS language, has the following form:




```
QUEUE    <<Queue.Name>>
SEIZE    <<Servicing device.Name>>
DEPART   <<Queue.Name>>
ADVANCE  (NORMAL(1, <<Normal
         distribution.Expected value>>,
         <<Normal distribution.Variance>>))
RELEASE  <<Servicing device.Name>>
```

The rule "Separator" transforms an instance of entity "Separator" into SPLIT command of GPSS language. This rule looks like:

Separator →

```
SPLIT  <<Separator.Amount of copies>>,
       <<Separator.Block>>
```

The rule "Collector", which converts an instance of entity "Collector" into ASSEMBLE command of GPSS language, has the following form:

Collector → `ASSEMBLE  <<Collector.Amount of flows>>`

The rule "Terminator", which converts an instance of entity "Terminator" into the appropriate command of GPSS language, looks like:

Terminator → `TERMINATE  1`

After applying of the described transformations to the model presented in Fig. 5 the MetaLanguage system has generated the following code in GPSS language:

```
GENERATE (UNIFORM(1, 2, 8)),,20,100,1
  QUEUE    QQ1
  SEIZE    SD1
  DEPART   QQ1
  ADVANCE (NORMAL(1, 3, 1))
  RELEASE SD1
  QUEUE    QQ2
  SEIZE    SD2
  DEPART   QQ2
  ADVANCE (NORMAL(1, 3, 1))
  RELEASE  SD2
  SPLIT    1, QQ4
  QUEUE    QQ3
  SEIZE    SD3
  DEPART   QQ3
  ADVANCE (NORMAL(1, 3, 1))
  RELEASE  SD3
  QUEUE    QQ4
  SEIZE    SD4
  DEPART   QQ4
  ADVANCE (UNIFORM(1, 2, 8))
  RELEASE  SD4
  ASSEMBLE 2
TERMINATE 1
```

The generated code of model was used for simulation running in GPSS system. The translation of any other visual model developed with created DSL, won't demand additional efforts of model designer or programmer.

V. Conclusion

The Metalanguage system, including transformation component, supports integration of different modeling systems. It provides interoperability of the languages and models in different information and analytical systems. This DSM-platform allows to reduce the complexity of analysts work, to increase efficiency of information systems functioning. Presented language workbench is quite convenient and flexible tool for building of modeling languages and transformation rules of the created models. Usage of the system allows to create DSLs and to determine transformations operatively. Users don't need programming language to develop languages or models. They operate with visual constructions or textual code of initial and target modeling languages.

The research prototype of MetaLanguage system has been used for development of several domain-specific languages, in particular, language for modeling of administrative regulations [15], for manufacturing processes modeling, applications for mobile devices, etc.

In the future, it is planned to develop the tools, which allow to automate the creation of visual DSLs on the basis of ontologies, that are obtained on the basis of on the analysis of corpus of domain documents. It will reduce the complexity of the creation of such languages and allow domain experts to participate in the process of DSLs creation.

In addition, it is planned to fulfil integration of the MetaLanguage system with simulation system Triad.Net [16]-[18].

References

[1] S. Sendall, W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development", *IEEE Software*, vol. 20, pp. 42–45, 2003.

[2] 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*, Orlando, 2009, pp. 147–153.

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

[4] Z. Melis, J. Zacek, F. Hunka, "Domain-specific modelling of business processes", in *Proc. of the 2013 International Conference on Systems, Control, Signal Processing and Informatics*, Rhodes Island, Rhodes Island, 2013, pp. 481–487.

[5] K. Balasubramanian, D. C. Schmidt, Z. Molnar, A. Ledeczi, "Component-based system integration via (meta)model composition", in *Proc. of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Tucson, Arizona, 2007, pp. 93–102.

[6] J. M. Alvarez, A. Evans, P. Sammut, "Mapping between levels in the metamodel architecture", in *Proc. of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Toronto, 2001, pp. 34–46.

[7] A. O. Sukhov, "Comparing of the system of visual domain-specific languages development", *Mathematics of Program Systems*, Vol. 9, pp. 84-111, 2012. (in Russian)

[8] A. O. Sukhov, L. N. Lyadova, "MetaLanguage: a tool for creating visual domain-specific modeling languages", in *Proc. of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering*, Perm, Russia, 2012, pp. 42–53.

[9] A. O. Sukhov, L. N. Lyadova, "Horizontal transformations of visual models in MetaLanguage system", in *Proc. of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering*, Kazan, Russia, 2013, pp. 31–40.

[10] E. B. Zamyatina, L. N. Lyadova, A. O. Sukhov, "Multilanguage modeling with MetaLanguage DSM-platform usage", *Informatization and Communication*, no. 5, pp. 11-14, 2013. (in Russian)

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

[12] J.-M. Favre, “Towards a basic theory to model driven engineering”, in *Proc. of the Workshop on Software Model Engineering*, Lisboa, 2004, pp. 48–55.

[13] 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.

[14] J. Hutchinson, M. Rouncefield, J. Whittle, “Model driven engineering practices in industry”, in *Proc. of the 33rd International Conference on Software Engineering*, Waikiki, USA, 2011, pp. 633–642.

[15] L. N. Lyadova, A. O. Sukhov, “Modeling of administrative regulations using opportunities of MetaLanguage language workbench”, in *Proc. International Conference on Information Systems Development Technologies*, Gelendzhik, 2013, part 2, pp. 45–49. (in Russian)

[16] I. S. Volegov, E. B. Zamyatina, “An ontological approach to the integration of the components of a simulation model Triad.Net”, in *Proc. of the International Scientific Conference on Open Semantic Technologies of Intelligent Systems*, Minsk, 2012, pp. 229–236. (in Russian)

[17] A. I. Mikov, E. B. Zamyatina, E. A. Kubrak “Implementation of simulation process under incomplete knowledge using domain ontology”, in *Proc. of the 6-th EUROSIM Congress on Modeling and Simulation*, Ljubljana, 2007, vol. 2, pp. 1–7.

[18] E. B. Zamyatina, A. I. Mikov, “Software tools of simulation system Triad.Net for it’s adaptability and openness”, *Informatization and Communication*, no. 5, pp.130–133, 2012. (in Russian)