Semantic Domain-specific Languages

Vitaly S. Gumirov
Founder and CEO of Eyeline.mobi Group
Novosibirsk, Russian Federation
vit@eyeline.mobi

Peter Y. Matyukov
Co-founder KIRIK.io and VP Innovation Eyeline.mobi Group
Novosibirsk, Russian Federation
peter@kirik.io

Dmitry E. Palchunov
Laboratory of Computability Theory and Applied Logic
Sobolev Institute of Mathematics, Novosibirsk State University
Novosibirsk, Russian Federation
palch@math.nsc.ru

***Abstract* — A rationale for the introduction of semantic domain-specific languages (sDSL) is discussed from the point of view of problems faced by IT industry. Goals and technological approaches for the development of sDSL are described. As well as mathematical model and foundation for such a concept is presented.**

***Keywords* — semantic modelling, artificial intelligence, machine learning, semantic smart contracts, semantic model, Δ_0-formula, fragment of atomic diagram**

I. Introduction

The traditional approach to software development based on algorithmic languages consists of the following basic stages:

1. Requirements specification (or Technical Requirements Document)
2. System architecture development
3. Algorithm programming
4. Testing

These stages should not necessarily follow each other in time, however, in a general case, they can be carried out interactively. The main problem arising in the process can be defined as this:

Destruction of meanings (semantics) when converting specifications into code.

II. Problem Statement

What does destruction of semantics mean in practice? It leads to a number of problems:

1. The costs of technical support and ownership raises in the course of time.

To track errors or make changes in the system one has to find the passage of code that matches the meaning (semantics) of the requirements, while the code does not have a direct link to the initial requirements. This link only exists in the brain of the programmer who wrote this passage of code.

2. High development and integration costs.

To manage the requirements, quality, and the architecture developers have to use special procedures to maintain links between the initial requirements and their corresponding passages of code, as well as procedures of change management that would allow to carry such links from the semantics (the requirements), through the architecture and design over to the code. Not only these procedures raise the cost of development, but result in yet other procedures for control of compliance and quality.

3. Lower quality due to discrepancies between specifications and systems.

Inability to directly verify conformity of code to the initial specification (semantics in the subject domain terms) by the subject domain specialists (e.g. by the customer's employees) leads to decrease in quality of the system, if discrepancies exist.

4. Higher costs of quality assurance.

The necessity to maintain quality standards – conformity of the code to the initial requirements in the first place – demands introduction of special procedures, which, as might be expected, makes the whole development process costlier.

5. Lower system efficiency

A system in the course of time tends to collect features that were developed, but whose documentation got missing, or became obsolete. These undocumented features cause drop in the system's performance and efficiency.

6. Gradual deterioration in security.

The said undocumented features can also cause unforeseeable system usage scenarios and security risks as the consequence.

At the same time the digitalization, whose advances we presently witness, assumes ever higher degree of automatization of many aspects of our life. Looking at the labor market we see that the demand for programmers does not subside, instead, it rises[1]. Scanty of this resource

[1] According to the U.S. Bureau of Labor Statistics, software developer jobs are expected to grow 17% from 2014 till 2024. https://www.forbes.com/sites/quora/2017/01/20/will-the-demand-for-developers-continue-to-increase/#3dcc636033ee

leads to growth of average wages in the sector, but, at the same time, to degradation of the average level of programmers' professional competence.

It also causes disbalances in many industries as the demand for IT specialists in some regions simply cannot be satisfied due to global mobility of the labor force[2].

III. Semanic Modelling And Semantic Domain-specific Languages

The answer to the above-mentioned challenges is wide adoption of semantic modelling, particularly semantic domain-specific languages of specification (hereinafter ***sDSL***).

The concept of semantic modelling (semantic programming) was proposed and developed in 1980s by Sergey Goncharov, Yury Ershov and Dmitry Sviridenko [1,19].

The basis of the semantic modelling (programming) concept is the idea of *executable specifications* stating that a subject domain specialist can use a semantic specification language to define specifications that will be automatically executed and/or converted into a valid code. Semantic modelling uses Δ_0-subset of first-order predicate logic formulas.

Definition. By the term *Semantic Domain-specific Languages of Specification* we mean such a domain-specific language that can be translated in Δ_0-formula language.

A distinct semantic domain-specific language of specification (as a syntactical representation of Δ_0-formulas domain-specific extension) can be devised to meet the needs of any subject domains. The possibility to create such domain-specific extensions is described in [2, 18]. The following systems serve as successful implementation examples of this approach.

1. Eyeline Service Delivery Platform, a system that is used by such industry giants as Mobile TeleSystems, Beeline, T-Mobil, SMART Philippines, Yellow Pages for mobile and fintech services management.

2. LibrettoLabs Ontobox, a system used in the retail sector.

Other perspective areas where semantic domain-specific languages can be successfully applied:

- Automation of business processes involving different industrial entities (semantic smart contracts)[3]
- Intellectual property management
- Automation of legal services
- Fintech, banks, insurance
- Project management automation
- Investment portfolio management
- Healthcare
- Telecommunications
- Robotics
- Driverless technologies
- Regtech

IV. Objectives of Semantic Domain-specific Languages

Designing a sDSL for a specific subject domain pursues the following goals:

1. Lowering costs of subject domain specialists training.
2. Lowering costs of integration and ownership of systems built on the sDSL basis.
3. Predictability of specifications made on the sDSL basis.
4. Feasibility of localization.
5. Feasibility of quality control automation for systems created on the sDSL basis.

V. Concept of Semantic Domain-specific Languages

Development and application of semantic domain-specific languages is based on the ***subject domain semantic model*** that will be set forth in the following paragraph.

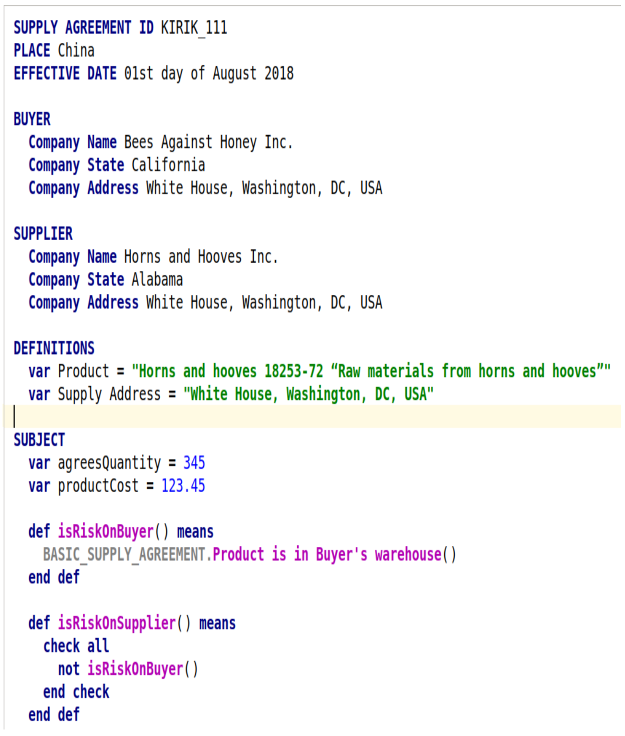
Based on the logical language of Δ_0-formulas the $\Delta°$ language can be run on the Semantic Machine, while sDSL gives a subject domain specialist the means to write specifications using a familiar vocabulary.

The key requirement here is that translation to $\Delta°$ language is made **automatically** – without human interference – since it eliminates the very possibility of mistakes and discrepancies in interpretation of meanings and, what is also important, reduces the workload.

There is not a well-established definition of domain-specific language (DSL) at present, so we will interpret in in the widest sense. One of the possible definitions is this: semantic domain-specific language (sDSL) is a representation method of subject domain knowledge that facilitates creation of executable semantic specifications by a specialist of this subject domain. sDSL can include, inter alia, graphics, textual information, and even voice/sound clips.

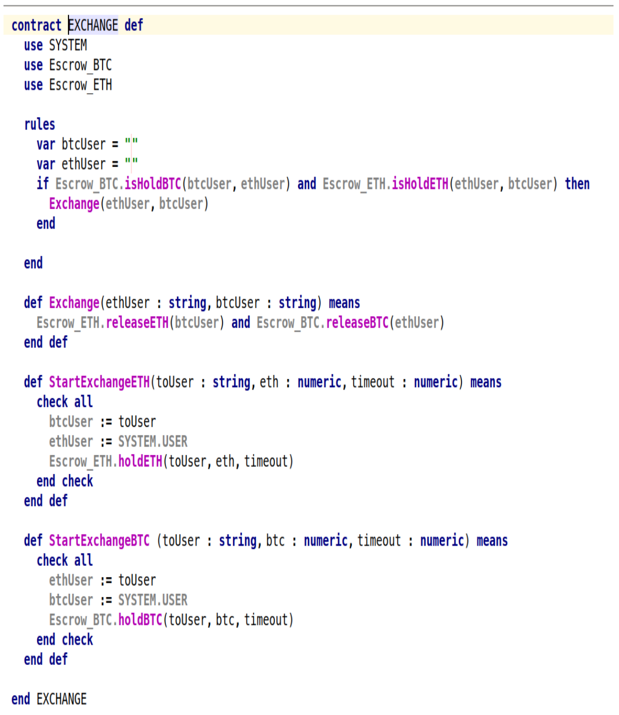
[2] For instance, experts in the power industry complain that digitalization makes impossible employment of IT specialists at pawer plants in remote areas because of inability to ensure adequate wages in this sector in these areas as compared to the average wages of IT workers in the country.

[3] See semantic smart contract platform KIRIK.io as an example [20]



```
SUPPLY AGREEMENT ID KIRIK_111
PLACE China
EFFECTIVE DATE 01st day of August 2018

BUYER
  Company Name Bees Against Honey Inc.
  Company State California
  Company Address White House, Washington, DC, USA

SUPPLIER
  Company Name Horns and Hooves Inc.
  Company State Alabama
  Company Address White House, Washington, DC, USA

DEFINITIONS
  var Product = "Horns and hooves 18253-72 “Raw materials from horns and hooves”"
  var Supply Address = "White House, Washington, DC, USA"

SUBJECT
  var agreesQuantity = 345
  var productCost = 123.45

  def isRiskOnBuyer() means
    BASIC_SUPPLY_AGREEMENT.Product is in Buyer's warehouse()
  end def

  def isRiskOnSupplier() means
    check all
      not isRiskOnBuyer()
    end check
  end def
```

Diagram 1: sDSL example for typical legal contracts.



```
contract EXCHANGE def
  use SYSTEM
  use Escrow_BTC
  use Escrow_ETH

  rules
    var btcUser = ""
    var ethUser = ""
    if Escrow_BTC.isHoldBTC(btcUser, ethUser) and Escrow_ETH.isHoldETH(ethUser, btcUser) then
      Exchange(ethUser, btcUser)
    end

  end

  def Exchange(ethUser : string, btcUser : string) means
    Escrow_ETH.releaseETH(btcUser) and Escrow_BTC.releaseBTC(ethUser)
  end def

  def StartExchangeETH(toUser : string, eth : numeric, timeout : numeric) means
    check all
      btcUser := toUser
      ethUser := SYSTEM.USER
      Escrow_ETH.holdETH(toUser, eth, timeout)
    end check
  end def

  def StartExchangeBTC (toUser : string, btc : numeric, timeout : numeric) means
    check all
      ethUser := toUser
      btcUser := SYSTEM.USER
      Escrow_BTC.holdBTC(toUser, btc, timeout)
    end check
  end def

end EXCHANGE
```

Diagram 2: Specification example in Δ° language

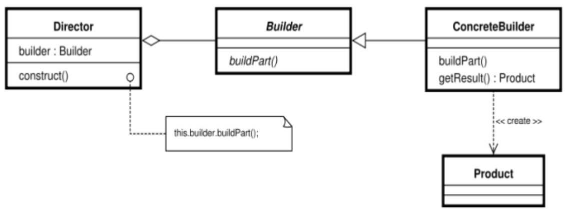


Diagram 3: Example of a specification represented graphically in UML

It is worth mentioning NLP (Natural Language Processing) and AI (Artificial Intelligence), they both can be used when making specifications in DSL.

For instance, a specialist utters the text while AI creates the sDSL specification asking, when necessary, to specify certain points, or repeat missing phrases.

In another example AI analyses the text of a legal contract written in the natural language and automatically creates a sDSL specification.

VI. Semantic Model

A semantic model consists of the subject domain ontology and the substantial description of this subject domain. The formal representation of the ontology and the theory of the domain is made in the language of finite fragments of atomic diagrams, or in its extension – the Δ_0-formula language.

When developing a semantic domain-specific language we use a 4-layer knowledge representation model realized in the form of the subject domain semantic model [3-6]. The semantic model includes the ontology and three other layers of subject domain knowledge representation: general (theoretical) knowledge, empirical knowledge, and estimated (probabilistic) knowledge [4, 6] (fig. 1).

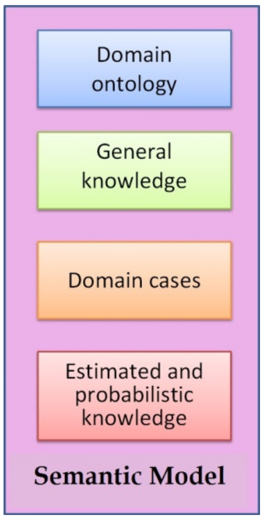


Fig. 1. Semantic model

1. The domain ontology contains definitions, the meaning specification of the key concepts that form the subject domain description vocabulary. In some cases, a hierarchy of ontologies can be considered instead of a single ontology. In such a case each ontology has its own set of key concepts (that is its own signature) and a set of definitions specifying the meaning of these key concepts.

In our model-theoretic approach to formalization of knowledge we use the following definition of ontology:

Definition [7, 8]**.** Ontology of the subject domain *SD* is a pair $O = < SA, \sigma >$, where σ is a set of key concepts of the domain, SA is a set of analytical sentences describing the meaning of these key concepts.

Analytical sentences are sentences, whose true value is defined solely by the meaning of the concepts (signature symbols) they contain [7-9].

Using ontologies, we can supplement the atomic diagram fragment of the model we build. To do it we can, for instance, use the relations between concepts presented in the ontology – synonymy, "hyponym-hypernym", etc.

On the other hand, since all concepts present in the text have certain meaning described by the ontology, the domain ontology should be true in the model whose atomic diagram fragments we build. In other words, the set of analytical sentences SA must be true in our model.

The subject domain substantial description present in its semantic model contains three other layers of knowledge representation:

2. General (universal) statements – laws and postulates of the domain. This is the theoretical knowledge of the domain: general principles, laws and regularities, everything that we know beyond all doubt. The general knowledge is the known part of the domain theory. This is a synthetical knowledge, whereas the ontology presents analytical knowledge. Unlike analytical, synthetical knowledge is not derived from the meaning of the concepts in use, its truth depends on the conditions of the real world. Therefore, this knowledge can transform – true statements can become false, while false statements can, vice versa, become true. But even so the meaning of the concepts in use can remain untouched. These statements have a common, universal character. Contrary to the next layer of representation – the empirical knowledge – they do not describe a particular situation, but the domain as a whole and are considered true in all situations.

3. Empirical knowledge describes factual situations, the domain cases. Description of each precedent (case) is formally presented as an atomic diagram fragment of the algebraic system. Knowledge of domain cases is formally presented as a case model that is a Boolean-valued model [10-12].

4. Probabilistic, fuzzy, and estimated knowledge. This knowledge is taken from external sources, or generated on the basis of the knowledge present at the first three layers of the semantic model. Fuzzification of the Boolean-valued case model [11, 12] described in the previous paragraph results in a fuzzy model where true values of formulas are numbers in the range [0, 1]. By analyzing the empirical data of the semantic model and comparing it with the general and ontological knowledge present in that semantic model the probabilistic and estimated knowledge is obtained. The formal representation of probabilistic and estimated knowledge is a set of sentences $\Gamma \subseteq S(\sigma_A)$ and a mapping $\mu\colon \Gamma \to [0,\ 1]$ associating each sentence $\varphi \in \Gamma$ with its fuzzy true value $\mu(\varphi) \in [0,\ 1]$ [11, 12].

Now we need to give some definitions and designations. Information on model theory is found in [13, 14].

We consider models $\mathfrak{A} = < A;\ \sigma > = < A;\ P_1, \dots, P_n, c_1, \dots, c_l >$ of signature $\sigma = < P_1, \dots, P_n, c_1, \dots, c_l >$, where A is the universum of model $\mathfrak{A}$, $P_1, \dots, P_n$ are symbols of predicates, $c_1, \dots, c_l$ are symbols of constants. $S(\sigma)$ designates the set of sentences – formulas without free variables – of the signature σ. $\mathfrak{A} \vDash \varphi$ means that sentence φ is true in the model $\mathfrak{A}$.

For a model $\mathfrak{A}$ of a signature σ let us denote $\sigma_A = \sigma \cup \{c_a \mid a \in A\}$; we suppose that $c_a \notin \sigma$ for any $a \in A$. $\mathfrak{A}_A$ designates the model of the signature σ_A, whose reduction to the signature σ is the model $\mathfrak{A}$, and $c_a^{\mathfrak{A}_A} = a$ for $a \in \mathfrak{A}$ is satisfied.

Let us denote:
$D(\mathfrak{A}) = \{\varphi \in S(\sigma_A) \mid \mathfrak{A}_A \vDash \varphi$ and the sentence φ is quantifier-free$\}$ – the elementary diagram of the model $\mathfrak{A}$.

Note that we consider signature σ consisting of only predicate symbols and constants. We call sentence φ atomic, if $\varphi = P(c_1, \dots, c_n)$, where $P, c_1, \dots, c_n \in \sigma_A$.

Assume that $\mathfrak{A}$ is a model of a signature σ. Atomic diagram of the model $\mathfrak{A}$ is usually considered as a set of sentences
$\{\ \varphi \in S(\sigma_A) \mid \mathfrak{A}_A \vDash \varphi$ and the sentence φ is atomic $\}$.

As we consider fragments (that is subsets) of atomic diagrams of models, it is more appropriate to use a different definition.

Definition. Atomic diagram of the model $\mathfrak{A}$ is a set of sentences
$AD(\mathfrak{A}) = \{\ \varphi \in S(\sigma_A) \mid \mathfrak{A}_A \vDash \varphi$ and either the sentence φ is atomic, or $\varphi = \neg\, \psi$ and the sentence ψ is atomic $\}$.

Note that atomic diagram $AD(\mathfrak{A})$ determines model $\mathfrak{A}$ with accuracy up to isomorphism. This means in particular that atomic diagram $AD(\mathfrak{A})$ uniquely determines elementary diagram $D(\mathfrak{A})$ and $AD(\mathfrak{A}) \vdash D(\mathfrak{A})$ is satisfied.

Therefore, the problem of model construction can be converted to the problem of construction of its atomic diagram.

In our model-theoretical approach to representation of subject domain knowledge we can distinguish two methods of formalization: semantic and syntactic. Knowledge of the domain including the ontological knowledge can be presented either using the algebraical system, or as a set of sentences of first order predicate logic.

Semantic methods are based on algebraical systems. Syntactic methods are based on use of theories, particularly elementary theories of algebraical systems. Each method has its advantages and shortcomings.

We use a synthesis of semantic and syntactic methods [15]. To formalize knowledge derived from the natural language texts and to presented it in semantic models we use finite fragments (subsets) of the models' atomic diagrams.

Model's atomic diagram is a set of atomic sentences – predicates of constants and atomic sentence negations – that are true in the model. The model's signature is enriched by additional constants – names of elements. As it was stated above an atomic diagram defines the model with the accuracy up to isomorphism. Therefore, on the one hand, the aggregate of finite fragments of atomic diagrams of the model completely determines the model, that is completely defines the semantics. On the other hand, the model's atomic diagram as well as all its fragments are some sets of sentences, so that description of knowledge with the use of atomic diagram fragments remains in the context of syntactic approach. Therefore presentation of knowledge using atomic diagram fragments of models is an integration of semantic and syntactic approaches.

Besides that, the use of finite fragments of atomic diagrams as a basic object allows us to solve two following problems.

First, we get rid of infinity. Indeed, on the one hand, the considered algebraic systems are normally infinite. On the other hand, any theory in the first order predicate logic is also infinite for the reason that the set of identically true sentences is infinite per se. Thus, we use finite sets of sentences – finite atomic diagram fragments – to approximate both infinite models and infinite theories.

Second, use of models' atomic diagrams fragments allows us to solve even more important and complicated problem – the necessity to work with algebraic systems of different signatures. Axiomatization as well as truth values of sentences in classes of models in the classical model theory only consider classes of algebraic systems that have the same signature. On the other hand, in the practical applications necessity arises of working with classes of models of different signatures [16]. Moreover, a situation is common when the signature of the model in question is not known beforehand: for example, in the case where the model is just being built.

This situation arises e.g. when we retrieve knowledge from natural language texts: the signature of an algebraic system that we build can be extended at any time. Specifically, if we combine knowledge derived from different documents we have to combine its signatures – sets of concepts present at these documents. We solve this problem using models' atomic diagrams, we merge several finite fragments of the diagram into one.

We use Δ_0-formulas to formally and logically present domain knowledge as a semantic model. For this purpose, we will use the following Δ_0-formula definition.

Definition.

1. If P is a predicate symbol, $P \in \sigma$, and $t_1, \ldots, t_n$ are symbols of constants of the signature σ, or variables, then $P(t_1, \ldots, t_n)$ is a Δ_0-formula.
2. If φ and ψ are Δ_0-formulas, then $\varphi \& \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, and $\neg \varphi$ are Δ_0-formulas.
3. If φ is a Δ_0-formula, x is a variable, and l is a finite list, then $(\forall\ x \in l)\varphi$ and $(\exists\ x \in l)\varphi$ are Δ_0-formulas.
4. There are no other Δ_0-formulas.

It is clearly seen that any atomic formula is a Δ_0-formula. Moreover, a conjunction of all formulas included in an arbitrary finite fragment of an atomic diagram is also a Δ_0-formula. Therefore, we can consider the language of Δ_0-formulas to be an extension of the language of finite fragments of atomic diagrams. Particularly, the above described semantic model of a domain can be completely described using the language of Δ_0-formulas.

It is important to note that the language of Δ_0-formulas as well as the language of finite fragments of atomic diagrams is decidable. One of our tasks is creation of effective resolution algorithms for the language of Δ_0-formulas and for the language of finite fragments of atomic diagrams.

Let us describe the construction process of model's atomic diagram fragment. Atomic sentences are generated by automated parsing applied to natural language texts describing a given subject domain. So we build a fragment of model's atomic diagram – the aggregate of atomic sentences. Amalgamation of several atomic diagram fragments corresponding to different natural language texts is possible. The software LogicText [17] is used that generates model's atomic diagram fragment from a natural language text.

VII. The Process of sDSL Development

When developing an sDSL the following rule should be observed: unlike programming languages, constructions of sDSL should answer the question WHAT, not HOW. For instance, sDSL can contain an instruction "Transfer money from account A to account B", but should not contain detailed algorithmic commands that fulfill this instruction. In other words, fulfillment should not be present at the specification level.

Below is the suggested process of sDSL development:

1. Build an object-oriented model of the subject domain.
2. Extract a subset of the object-oriented model (data, predicates, class methods) that should be brought to the specification level and have representation in the sDSL syntax/notation).
3. Implement basic predicates/basic functions that will be used in the sDSL. These can be database systems, machine learning and neural network systems, and/or other systems made by means of traditional programming languages.
4. Choose the notation type for the sDSL: text, graphics, or a Natural Language Processing (NLP) system.
5. Implement automatic translation from the sDSL to Δ° language.

VIII. Conclusion

The technologies based on semantic domain specific languages will enable a new paradigm shift for the entire IT industry. Finaly, we can bridge the gap between specifications and the execution, between domain specialists and the computer, by providing them with tools allowing to directly create executable specifications. This methodology is proven by the experience of Eyeline.mobi group which uses semantic modelling technology for telecommunication and fintech industries since 2012. This technology allows telco companies to serve more than 200 mln end users, and allows to process thousands of transactions per second. Transactions support is enabled by that the semantic engine is based on a ternary logic of semantic modelling theory.

On the other side, we hope that our new developments and release of Delta0 Semantic Definition Kit (`d0sl SDK`) will open new opportunities for researchers, students and developers working in different AI related areas.

References

[1] Goncharov S.S., Ershov Yu.L., Sviridenko D.I. Methodology aspects of the semantic programming // Scientific knowledge: logic, notions, structure. – Novosibirsk, Nauka 1987, pp. 154-184.

[2] Gumirov V.Sh. Object-oriented variation of the ∑-specifications language [Online]. Available: https://goo.gl/UjvUrp

[3] D.Palchunov, G.Yakhyaeva, O.Yasinskaya. Software system for the diagnosis of the spine diseases using case-based reasoning // The Siberian Scientific Medical Journal. 2016. Vol. 36, No. 1, p. 97-104.

[4] C.Naydanov, D.Palchunov, P.Sazonova. Development of automated methods for the critical condition risk prevention, based on the analysis of the knowledge obtained from patient medical records. In: Proceedings of the International Conference on Biomedical Engineering and Computational Technologies (SIBIRCON / SibMedInfo — 2015), 2015, Novosibirsk, p. 33-38.

[5] Palchunov D., Yakhyaeva G., Dolgusheva E. Conceptual Methods for Identifying Needs of Mobile Network Subscribers // Proceedings of the Thirteenth International Conference on Concept Lattices and Their Applications, Moscow, Russia, July 18-22, 2016, p. 147-160.

[6] Naidanov C. A., Palchunov D. E., Sazonova P. A. Model-theoretic methods of integration of knowledge extracted from medical documents // Vestnik NSU Series: Information Technologies. - 2015. - Volume 13, Issue No 3. - P. 29–41. - ISSN 1818-7900. (in Russian).

[7] Palchunov D.E. Modeling of reasoning and formalization of reflection I: Model theoretical formalization of ontology and reflection. Filosofiya nauki, No. 4 (31), 2006, p. 86-114 (in Russian).

[8] Palchunov D.E. The solution of the problem of information retrieval based on ontologies // Bisnes-informatika, No. 1, 2008, — p. 3–13 (in Russian).

[9] Carnap, R. Meaning and Necessity. A Study in Semantics and Modal Logic. —Chicago, 1956. — 220 p.

[10] Pal'chunov D.E., Yakhyaeva G.E. Interval Fuzzy Algebraic Systems // Mathematical Logic in Asia. Proceedings of the 9th Asian Logic Conference'05. World Scientific Publishers, 2006. Pp. 191-202.

[11] Pal'chunov D.E., Yakhyaeva G.E. Fuzzy algebraic systems // Vestnik NGU. Seriya: Matematica, mexanika, informatika, vol. 10, no. 3, 2010, p. 75-92 (in Russian).

[12] Pal'chunov D.E., Yakhyaeva G.E. Fuzzy logics and fuzzy model theory // Algebra and Logic, vol. 54, no. 1, 2015, p. 74-80.

[13] Chang C.C., Keisler H.J. Model theory. — Moscow: Mir, 1977. — 615 p. (in Russian).

[14] Ershov Yu. L., Palyutin E. A. Mathematical Logic. — Moscow: Nauka, 1979. — 317 p. (in Russian).

[15] Makhasoeva O.G., Palchunov D.E. Semi-automatic methods of a construction of the atomic diagrams from natural language texts // Vestnik NSU: Information Technologies - 2014. Vol. 12, No. 2, p.64–73. - ISSN 1818-7900 (in Russian)

[16] Kleschev A.S., Artemeva I.L. Mathematical models of subject domain ontologies. Parts 1-3 // Nauchno-tekhnicheskaya informatsiya. Seriya 2. 2001, No. 2, p. 20-27, No. 3, p.19-29, No. 4, p. 10-15. (In Russian).

[17] Makhasoeva O.G., Palchunov D.E. Program system for the construction of the atomic diagram of a model from natural language texts. (in Russian). Certificate of the State Registration of the computer program. No. 2014619198, registered 10.09.2014.

[18] Gumirov V.Sh. Object-oriented variation of the ∑-specifications language // Computational theory and specification languages, Novosibirsk, 1995 - Vol. 152: Vychislitelnye sistemy, pp. 3-19.

[19] S.S. Goncharov, D.I. Sviridenko Σ-Programming Amer. Math. Soc. Transl. (2) Vol. 142, 1989

[20] KIRIK.io whitepaper, 2018 //[Online]. Available: http://bit.ly/2Gjn2wE