Principles of organization and automation of the semantic computer systems development

Vladimir Golenkov, Daniil Shunkevich, Irina Davydenko, Natalia Grakova
Belarussian State University Informatics and Radioelectronics
Minsk, Belarus
golen@bsuir.by, shunkevich@bsuir.by, davydenko@bsuir.by, grakova@bsuir.by

***Abstract*—The work is devoted to the principles of development of semantic computer systems of a new generation based on the Open Semantic Technology for Intelligent Systems Design (OSTIS Technology). The advantages of the transition from traditional computer systems to semantic computer systems from the point of view of the design process are substantiated, and the advantages of implementation of automation tools for design activities as semantic computer systems are considered.**

***Keywords*—design automation tools, semantic technology, semantic network, knowledge base, problem solver, ontology, multi-agent system**

I. Introduction

At the present stage of development of computer technologies, various types of design automation systems (CAD systems) are widely used in almost all production areas. The use of systems of this kind is relevant both in the production of material objects and in the development of computer systems. Modern CADs allow automating many processes related to both directly designing an object and its development and implementation, and, as a result, can significantly reduce production time and cost of the product, as well as minimize the number of errors associated with human factors. An important direction of the development of CADs is their intellectualization, which imposes fundamentally new requirements on the development of CAD technologies.

In a number of previously published works, the authors proposed the concept of the Open Semantic Technology for Intelligent Systems Design (***OSTIS Technology***) [1], [2], focused on the development of computer systems of the new generation (first of all – hybrid intelligent systems [3]) which will be called semantic computer systems or ***ostis-systems***, if it is necessary to emphasize their compliance with the standards of *OSTIS Technology*. The *model of hybrid knowledge bases of ostis-systems* and models for representing various types of knowledge within the framework of such a knowledge base [4], as well as *model of a hybrid problem solver*, which allows to integrate various problem solving models [5], were also proposed.

The main requirement for *OSTIS Technology* is to ensure the possibility of joint use within the *ostis-systems* of various types of knowledge and various problems solving models with the possibility of unlimited expansion of the list of knowledge used in *ostis-system* and problem solving models without significant labor costs. The consequence of this requirement is the need to implement the component approach at all levels, from simple components of knowledge bases and problem solvers to whole *ostis-systems*.

To meet these requirements, the most important task is not only the development of appropriate *ostis-systems* models and their components, but also the development of an integrated methodology and appropriate tools for automating the construction and modification of *ostis-systems*.

Thus, within the framework of this work, attention is paid to the principles of organizing and automating the development process *ostis-systems*, which underlie the relevant methodology and tools, including the principles of regulation and stimulation of activities aimed at developing *ostis-systems* and their components, and also considered the main advantages of the transition from traditional computer systems to *ostis-systems* from the point of view of the design process and maintenance of such systems.

However, the transition from traditional computer systems to *ostis-systems* will allow not only to obtain a number of advantages associated with such key properties of *ostis-systems*, such as hybridity, modifiability and learnability, which are discussed in detail in the above works, but it will also allow to bring to a fundamentally new level the processes of designing and maintaining of computer systems and the degree of automation of such processes. In this case, it is assumed that the building automation and modification tools for *ostis-systems* should be implemented as a *ostis-system* themselves and integrated with the system being developed, which will give a number of additional benefits.

Thus, within this work attention is paid to the principles of organizing and automating the development process of *ostis-systems*, which underlie the relevant methodology and tools, including the principles of regulation and stimulation of activities aimed at *ostis-systems* and their components developing, and also considered the main advantages of the transition from traditional

computer systems to *ostis-systems* from the point of view of the design process and maintenance of such systems.

II. Intellectualization of design automation tools

Much attention in modern literature is given to various approaches to the construction of intelligent CAD systems. Unfortunately, in many cases, intellectualization refers to adding the simplest adaptive functions («intelligent cursor», «intelligent menu», etc.), however, an analysis of the sources revealed key areas in the field of the intellectualization of CADs, within which there are significant results.

One of the most important and most promising areas in the field of building intelligent CAD is **generative design** [6], which assumes that the computer system itself acts as an active participant in the design process. According to the concept of generative design, the designer sets the required minimal description of the parameters of the designed object, after which the system independently generates the initial version of the designed object model, which is further refined and updated in dialogue with the human designer. Important for the development of this direction was the introduction at first of CNC machines, and then 3D printing technologies into mass production, making it possible to manufacture objects of a much more complex shape based on their detailed model. Taken together with the development of computing powers this made it possible to solve various optimization problems using CAD, in particular, the problem of topological optimization (eliminating unnecessary material from a part while maintaining key properties such as maximum load).

Autodesk Dreamcatcher [7] is the most advanced and currently popular tool implementing the concept of generative design, which allows to develop designs of parts for various products based on the design intent expressed by people in natural language and then modify projects to simplify their production on a specific equipment.

Another important area of CADs intellectualizatio, which is significantly less developed in comparison to the one discussed earlier, is the direction suggesting that CAD should also perform the **learning system** [8] functions. It is important to note that both the learning of the designer and the learning of the system itself in the process of work are considered. In turn, the designer's learning can also be considered in two aspects: learning in working with CAD systems, that is, studying the functionality and principles of working with a particular system, as well as learning in the actual subject domain in which the design is carried out, that is, studying the detailed aspects of the designed objects, their purpose, design features, etc. It is obvious that the development of this direction imposes additional requirements on the technologies underlying such systems, in particular, requires the coordinated use of heterogeneous information and various models of information processing.

Another important area of intellectualization is **simulation modeling**. Simulation can be widely used at different stages, in particular:

- modeling the behavior of the object of development under the influence of various factors, in a variety of external environment, etc., which is especially important when developing complex expensive systems designed to work in unpredictable conditions;
- project management modeling, which can be used for training or testing personnel, assessing potential risks when choosing a specific management strategy, working out certain practices and skills under conditions similar to a real project [9];

Obviously, to realize the possibility of comprehensive simulation, it is necessary to have tools that allow, on the one hand, to describe in details complex heterogeneous objects from different points of view, i.e. in fact, integrate within the framework of a unified system various types of knowledge, as well as various approaches to the interpretation of such descriptions. In addition, it is necessary to have the ability to easily modify the existing models, in particular, it should be easy to change the number and types of influencing factors, it should be easy to change the principles of behavior of objects of the environment and the simulated objects themselves, etc.

It is important to note that simulation modeling can be the basis for the use of other models and methods of artificial intelligence. For example, the possibility of a large number of simulation launches and the accumulation of certain information about these simulations can be the basis for their further analysis and application of machine learning methods.

The next stage in the development of production systems in general is the transition from CADs to more general PDM-systems (Product Data Management), and further to complex PLM-systems (Product Lifecycle Management) [10], as well as CALS-systems and technologies (Continuous Acquisition and Lifecycle Support).

The construction of such integrated information systems requires the unification (standardization) of heterogeneous information. To solve this problem, the ontological approach is currently widely used both in the development of software systems [11], [12], and in other areas [13], [14].

Thus, various kinds of intellectualization of design automation tools require solving the compatibility problem of various information representation and processing models, a specific list of which for different systems may differ significantly, since it depends on the design object, requirements for functionality of the tools, etc. In addition, the development of such funds, including those

associated with changes in the design object, requires a reduction in the laboriousness of modifying these tools, which is also currently a serious problem.

In this paper, it is proposed to solve these problems by using unified models for the representation and processing of information proposed in the *OSTIS Technology*, in particular, models of hybrid knowledge bases and hybrid problem solvers. Next, we consider in more detail the principles of building automation tools that implement these principles.

III. Architecture and features of the development of semantic computer systems

A. Architecture of semantic computer systems

Lets consider the features of the *ostis-systems* architecture, affecting the design process and the principles of appropriate automation tools constructing.

As a basis for knowledge representation in the framework of *OSTIS Technology*, a unified version of coding information of any kind based on semantic networks is used, named ***SC-code*** [1]. Elements of *SC-code* text (*sc-texts*) are named *sc-elements*, among which, in turn, are *sc-nodes*, *sc-arcs* and *sc-edges*. As part of the technology, several universal variants of visualization of *SC-code*, such as *SCg-code* (graphic variant), *SCn-code* (nonlinear hypertext variant), *SCs-code* (linear string variant).

Each *ostis-system* consists of a complete model of this system, described by means of *SC-code* (*sc-model of computer system*) and *sc-model interpretation platform*, which in general can be implemented both in software and in hardware [2]. This ensures complete platform independence of *ostis-systems*.

In turn, the *sc-model of computer system* is conventionally divided into *sc-model of knowledge base*, *sc-model of problem solver* and *sc-model of computer system interface* (including user interface, interface with the external environment and interface with other ostis-systems), as well as the model of abstract semantic memory (*sc-memory*), in which the *SC-code* constructs are stored, and, accordingly, all the listed *sc-models* (figure 1).

The principles of building of *sc-models of knowledge bases* and *sc-models of problem solvers* are discussed in more detail in the papers [4] and [5], respectively.

The ***sc-model of knowledge base*** is based on such basic principles as the distinguish of the hierarchical system of *subject domains* and *ontologies* (including the presentation level meta-ontology and the family of top-level ontologies that are part of each developed *ostis-system*), as well as the distinguish of *structures* (signs of entire fragments of the knowledge base), which can be subsequently described in the same knowledge base. The use of these and other principles provides such important properties as the ability to represent knowledge of various types in the knowledge base, ease of making changes to the knowledge base, including the possibility of expanding the set of knowledge types used, as well as the possibility of structuring the knowledge base according to an arbitrary set of features and the possibility of representation in the knowledge base of meta-knowledge of an arbitrary level.

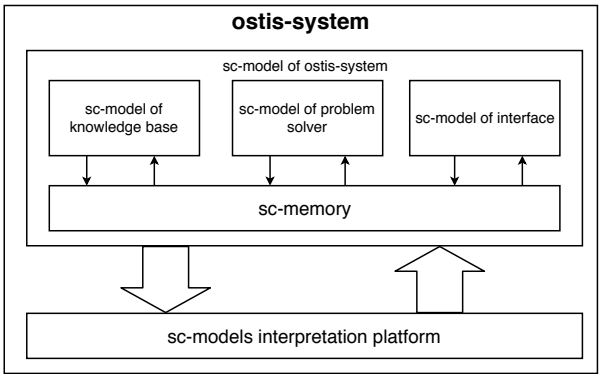

Figure 1. Ostis-system architecture

The ***sc-model of problem solver*** is based on the principle that the solver is treated as a hierarchical system of agents that react to situations and events in sc-memory (*sc-agents*) and interact with each other exclusively by specification of the information processes performed by the corresponding agents in the *sc-memory*. Such *sc-agents* can be *atomic*, i.e. those for which the program of their actions is specified and *non-atomic*, i.e. those that are decomposed into simpler *sc-agents*. Classes of functionally equivalent *sc-agents* are called *abstract sc-agents*. Each *abstract sc-agent* has a corresponding specification that contains, at a minimum, the initiating condition of *sc-agent* and the *sc-agent* implementation description depending on whether it is atomic or non-atomic *sc-agent*.

Further, speaking about the knowledge base, problem solver and user interface, we assume that we are talking about the *sc-model of knowledge base*, *sc-model of problem solver* and *sc-model of user interface*, respectively.

The sc-model separation into components is rather conditional, since an important architectural feature of *ostis-systems* is the fact that both the solver and the system interface are in fact part of its knowledge base. This is achieved through the following principles:

- all agents that are part of the solver (including the programs of agents), all the information processes they perform are described in the knowledge base by means of *SC-code*. This possibility, in turn, is achieved due to the presence within the framework of *OSTIS Technology* of programming language *SCP*, the programs of which are written using *SC-code*;
- the *ostis-system* interface is treated as a subsystem built according to the same principles, that is,

having its own sc-model of knowledge base and sc-model of problem solver, which in turn are based on the corresponding principles discussed above.

Thus, the most important feature of the development of *ostis-systems* is that **development of *ostis-system* actually comes down to development of its knowledge base**. When developing the components of the problem solver and the interface, their features are taken into account, however, the general mechanism for making any changes to the *ostis-system* becomes unified.

B. General typology of project actions of semantic computer systems developers

In the general case, when developing *ostis-systems* (as well as many other computer systems), the following types of project activities can be performed:

- synthesis (generation) of components and systems with specified properties
  - search for the closest components in components library;
  - adjustment of the specified (for example, found) component in order to obtain the specified property
  - assembly of large parts;
- integration of the developed component into the system for which the component is intended;
- analysis of the developed component or system
  - analysis of correctness (absence of errors and contradictions);
  - analysis for compliance with the required characteristics, including testing and test generation;
  - integrity analysis (completeness);
  - clearness analysis (absent of excesses);
  - value analysis;
  - evaluation of the project workload;
- specification (description, documentation) of the developed component or system;
- control of project discipline (adherence to work schedule);
- design management (assignment of performers and deadlines for specific project tasks);
- developer coordination;
- version control;
- analysis of the contribution of each developer to the overall result;
- stimulation of project activities.

IV. Existing approaches to the organization of the computer systems development

Despite the architectural features, each *ostis-system* is a computer system, and therefore, when designing *ostis-systems*, it is necessary to take into account current trends in the development of computer systems and the requirements for the corresponding automation tools.

In this section, we consider existing approaches to the organization of the development process of computer systems depending on the type of development object:

- general methodologies for developing computer systems, the object of development in general is any computer system, including the semantic computer system;
- means of automating the computer systems development, the object of development is a computer system built on the basis of traditional computer technologies. Many approaches implemented in such tools can also be used in the development of semantic computer systems; however, in a ready-made form, these tools are not focused on the development of such systems and do not take into account their features;
- tools for developing knowledge bases and ontologies, the object of development is the components of knowledge bases, first of all, ontologies;
- means of component development of intelligent systems, the object of development is intelligent computer systems.

A. Modern general methodologies for computer systems development

With the development of information technologies in the last decade, the Agile [15], [16] family of software development methodologies has gained the most popularity. Most Agile methodologies belong to the so-called lightweight methodologies and are contrasted with the classic heavyweight, such as, for example, the cascade (waterfall) model. It can be said that modern Agile methodologies actually integrate the best of the ideas underlying the more traditional methodologies (spiral, iterative, cascade, etc.), taking into account the peculiarities of the current stage of software systems development, first of all - the need for substantial more rapid adaptation ever-changing requirements, as well as the need for relevant workable versions of the system being developed. These features are fully valid for *ostis-systems*, however, due to their architectural features, many principles of Agile methodologies for such systems can be implemented easier than for traditional computer systems, which will be discussed in more detail below.

B. Existing automation tools for the development of computer systems

Most modern computer-aided design tools do not limit users to using any particular methodology (although there are tools that implement a specific methodology, for example, Trello).

In this section, we briefly review the main classes of tools aimed at solving some particular problems in the field of computer systems development.

1) Version control systems: Version control systems are currently used in the development of almost any software systems and give developers the following main features:

- save project versions with the ability to rollback to any previous version if necessary;
- fixing the authorship of each change, the date and time of the change, as well as the ability to specify the sense and reason for the change;
- access control and the possibility of making changes to the main project for different developers;
- many version control systems allow to create different versions of the same document, if necessary.

An overview of modern version control systems and their comparative analysis is given, for example, in [17].

2) Issue tracking systems, bug-tracking systems: Systems of this class allow developers to fix current project tasks (including errors correcting and other problems), assign performers and deadlines for tasks, indicate the status and priority of tasks. Modern systems of this class provide wide opportunities for discussing tasks, assessing the contribution and activity of developers, etc.

A comparative analysis of systems of this class is given in [18].

3) Project management systems: Project management systems are in many ways similar to bug tracking systems, but unlike them, as a rule, they are not focused on developing only software products. In addition, the key difference in developed project management systems is the availability of tools for estimating project development deadlines, development of plan implementation monitoring tools, visualization of project activities in the form of generally accepted diagrams, etc., that is, the emphasis in such systems is transferred to management and control design process.

A list of popular systems of this class with a brief description of their capabilities is given in [19].

4) Verification automation systems: Verification automation systems can be divided into the following classes:

- **continuous integration systems** that integrate with the version control system, build a project for each new version or according to a schedule, automatically perform a number of embedded tests, and if errors occur, immediately inform the developers;
- **test automation systems** that allow to automate part of the manual work in the process of testing of the developed product and its components.

Lists of systems of this class with specifications are given, for example, in [20], [21].

5) Project hosting systems: Project hosting systems give developers the opportunity to place repositories with the code of their projects in the cloud, and administer them. Each system of this kind is a complex system capable of working with at least one version control system, and also, as a rule, integrates the auxiliary systems of all the classes listed above (project management, tracking errors and tasks, automatic verification). Many systems of this kind also provide ample opportunity for project documentation.

A comparative review of the most popular systems of this class today is given, for example, in [22].

Thus, the systems of these classes solve problems that are relevant in the development of computer systems of any kind, including *ostis-systems*. Due to this the task of developing tools that would solve these problems when designing *ostis-systems* taking into account the specifics of such systems becomes urgent. The implementation of such tools on the basis of *OSTIS Technology* will significantly simplify the integration of subsystems that solve different tasks, which, in turn, will expand the functionality of such tools, which will be discussed in more detail below.

C. Knowledge base and ontology development methodologies

As mentioned earlier, the development of the *ostis-system* comes down to the development of its knowledge base, and therefore it is advisable to consider existing approaches to the development of knowledge bases, as well as appropriate tools. At the same time, in modern literature, when analyzing methods for knowledge bases development, the focus is on analyzing methodologies for developing ontologies, which are the basis of modern knowledge bases, including knowledge bases *ostis-systems*.

There are many papers devoted to the review of various approaches and methodologies for ontology design [23], [24].

In [25], a variant has been proposed for the classification of existing methodologies for the development of ontologies, based on the most essential features, which include:

- team development support;
- degree of dependence on the toolkit;
- type of ontology life cycle model used;
- possibility of formalization;
- ability to reuse knowledge base components;
- strategy for distinguishing of subject domain concepts;
- ability to support compatibility of developed ontologies.

Among the main methodologies that have been most developed to current date and have become basic in the field of creating ontologies of subject domains, the following can be singled out: Ushold and King's skeletal methodology [26], Gruninger and Fox methodology (TOVE) [27], METHONTOLOGY [28], [29], On-To-Knowledge (OTK) [30], KACTUS [31], DILIGENT [32], SENSUS [33] and UPON [34]. Their com-

parative characteristics in accordance with the above classification are given in [25].

Analysis of the reviewed methodologies shows that none of them is complete enough, and all proposed solutions are not unified. Most methodologies do not support the joint development of knowledge bases, the compatibility support for the knowledge bases being developed and, as a result, the support for the reuse of already developed knowledge bases and their components.

In addition, the overwhelming majority of knowledge base development methodologies describe the development process in general terms, not regulating the actions of participants at each stage of ontology development, not specifying the principles of matching new concepts with existing ones, the subjective influence of developers is high. Thus, the problem of compatibility of components of knowledge bases remains relevant when using even the most developed methodologies.

D. Knowledge base and ontology development tools and knowledge base development projects

Let us consider in more detail some of the most common knowledge base development tools available today.

Wiki-technology

Wiki-technology allows to accumulate knowledge that is presented in an interoperable form, providing navigation through knowledge. It is possible to use Wiki-Technology for projects of any scale and thematic focus (from open electronic encyclopedias, to reference systems of various enterprises and educational institutions) [35], [36].

Wiki-Technology provides its users tools for storing and structuring text, hypertext, files and multimedia. Wiki-Technology uses the MediaWiki [37] platform as a tool, which allows to perform information interaction, providing access to information resources to all participants in the system development process, organizing management and monitoring of the development [38]. The advantages of this technology include the simplicity of Wiki markup, communication capabilities that are realized through joint editing of pages, as well as through electronic discussions in the Wiki or additional media such as chat or forum, the design nature of the work, cooperation, the formation of a single product of joint activities meaningful interaction, knowledge sharing, evaluation and continuous improvement of work [35].

The influence of the Semantic Web on such projects is constantly increasing, as a result, Wiki-sites engines have been created that support the ontological representation of knowledge and semantic markup of resources using Semantic MediaWiki [39]. These tools allow you to include semantic annotations in Wiki markup in the form of OWL and RDF and explicitly separate structured and unstructured information [35].

In addition to these advantages, the Wiki as a technology has several disadvantages: duplication of information on different pages, the impossibility of structuring knowledge due to the lack of a hierarchy of hyperlinks and the lack of unification of the presentation of information, the lack of automatic verification. In addition, Wiki-Technology is currently designed to work only with structured natural-language texts, thus, based on this technology, it is not possible to build knowledge bases of intelligent systems, since informal text is unsuitable for automatic processing to the extent necessary for solving various problems, for example, logical inference problems.

However, many ideas of Wiki-technology can be adapted for the collective development of knowledge bases.

Software environments for ontology construction

Existing software for building knowledge bases (ontologies) are conditionally divided into three [40] groups:

1) Ontology creation tools. This class of tools supports the process of creating a knowledge base ”from scratch”. In addition to editing and browsing, tools provide support for ontology documentation, ontology import/export into various formats and languages, and ontology library management.

These include: Protégé [41], NeON [29], Co4 [42], Ontolingua [43], OntoEdit [30], OilEd [44], WebOnto [45] etc. A brief description of these and other tools can be found in [46], [47], [48].

2) Tools for displaying, aligning and combining of ontologies. This class of tools helps users find the similarities and differences between the original ontologies and create a resultant ontology that contains elements of the original ontologies. The author [47] divides them into subgroups according to the following features:

- to combine two ontologies in order to create the new one (PROMPT [49], Chimaera [50], OntoMerge [51]);
- to determine the conversion function from one ontology to another (OntoMorph [52]);
- to define the mapping between concepts in two ontologies, finding pairs of corresponding concepts (for example, OBSERVER [53], FCA-Merge [54]);

3) Ontology-based annotation tools. The most important condition for implementing the goals of the Semantic Web is the ability to annotate Web resources with meta-information. For this reason, recently ontology engineering tools include ontology-based annotation tools. These include: MnM [55], SHOE Knowledge Annotator [56], etc.

In the context of solving the tasks set in the framework of this work, it makes sense to consider in detail only the first class of tools.

In ontological engineering, any ontology is considered as the result of coordinated activities of a group of

specialists on a model of a certain field of knowledge. Based on this, with the development of methods and tools in the field of knowledge engineering, an increasing attention has been paid to the instrumental support of the process of collective development of ontologies, within which there are several basic tasks [57], [58]:

- management of interaction and communication between developers;
- access control to current results of joint design;
- copyright fixation for expert knowledge passed to the public;
- design error detection and error correction management;
- competitive change management.

Currently, to solve these problems, there are already several fairly well-developed approaches and appropriate tools. Among them are the following:

- Collaborative Protege [59];
- NeOn project [29];
- infrastructure for joint development of consistent knowledge bases Co4 [42].

The main disadvantages of the considered tools include:

- the lack of developed tools for automatic editing and verification of knowledge bases, including the assessment of completeness and redundancy;
- lack of a single mechanism for the collective creation of knowledge bases, including means of coordinating changes between developers of different levels of responsibility, a typology of developer roles;
- insufficient level of extensibility of development tools.

E. Analysis of the means of component development of intelligent systems

The use of ready-made components is the most important way to reduce the time and complexity of the development of computer systems, and reduce the professional requirements for their developers.

The issues of component design of intelligent systems and, in particular, knowledge bases and problem solvers are discussed in the works [60], [61], [62], [63]. When creating the first systems based on knowledge, it was assumed that these systems would ideally solve the problem of reusable components, however, developers faced a number of problems that are relevant to date [64].

Many researchers and developers determine the availability of ontological libraries as an important component of the Semantic Web [65] infrastructure. The first libraries and collections of ontologies were developed within such projects as Ontolingua server [43], DAML [44], Protégé ontology library [41], Ontaria ontology directory [66] and SchemaWeb [67]. Many of these projects are not currently supported, but they are being replaced by a new generation of ontology libraries [68].

However, as shown in [69], [70], the majority of ontologies developed based on the Semantic Web standards are not consistent with each other and, therefore, cannot be reused as components of knowledge bases, as was supposed by the developers of the Semantic Web standards.

The problems in the component design of knowledge bases include the following:

- many components use the developer's language (usually English) to identify concepts, and it is assumed that all users will use the same language. However, for many applications, this is unacceptable – developer-only identifiers should be hidden from end users, who should be able to choose a language for identifiers that they see [64];
- lack of unification in the principles of representing different types of knowledge within one knowledge base, and, as a result, the lack of unification in the principles of identifying and specifying reusable components leads to incompatibility of components developed in the framework of different projects [60];
- lack of search engine components that meet the specified criteria.

To date, a large number of knowledge bases have been developed in the different subject domains [71]. However, in most cases, each knowledge base is developed separately and independently from the others, in the absence of a unified formal basis for the presentation of knowledge, as well as unified principles for the formation of systems of concepts for the described subject domain. In this connection, the developed bases are, as a rule, incompatible with each other and not suitable for reuse.

In turn, in the development of problem solvers there are a large number of specific implementations, but the compatibility issues of different solvers for solving one problem are practically not considered in the literature.

There are a number of works that solve problems of accumulation and reuse of components of problem solvers [60], as well as the development of a unified platform for integrating various models of problem solving [72], but the problem of their compatibility is still relevant.

Thus, the problem of the development of common unified principles for the distinguishing and specification of reusable components of intelligent systems and the formation of a library of such compatible components is still relevant. In this case, reusable components of different levels of complexity can be distinguished - from specifications of single concepts and single knowledge processing agents to reusable ontologies, ontology systems, and problem solvers.

V. Requirements for the methodology and development tools for semantic computer systems

This section considers the requirements formulated on the basis of the analysis of modern methods and tools for developing computer systems, as well as modern approaches to the development of intelligent systems, and in particular, knowledge bases.

Given that the development of *ostis-system* comes down to the development of its knowledge base, the requirements for *ostis-systems* development automation tools include:

- refusal to edit the source code of the knowledge base (in any external languages) in favor of directly editing the knowledge base stored in the *sc-memory* by means of the appropriate editors. The main advantages of this approach are as follows:
  - the ability to automate the verification and editing of the knowledge base stored in memory;
  - the possibility of automating the process of integrating new knowledge (first of all, identifying and eliminating synonymous fragments) into the knowledge base being developed;
  - the ability to edit the knowledge base (as a result, the problem solver) directly during the operation of the system;
  - possibility during the collective development to allocate, if necessary, fragments of the knowledge base of arbitrary configuration and appeal to them in the process of discussion and agreement;
- there should not be any fundamental restrictions on the top level development methodology used, the nomenclature of the distinguished roles of developers, models of organization and management of the development process. At the same time, due to its openness, OSTIS Technology is focused on the development of primarily open-source projects, which also have a number of features from the point of view of the designing organization [16];
- possibility of joint development of a knowledge base by the development team (including distributed teams), including the possibility of discussing (agreeing on) and administering the changes, if necessary, the ability of third-party subject domain experts help in need to solve some contradictions. In addition, with an increase of the knowledge base size, it becomes important to organize the hierarchical administration of the knowledge base, with indicating the responsibilities of each administrator;
- the possibility, when experts have opposing points of view, in the process of agreeing on any fragments of the knowledge base to fixate and/or resolve these contradictions. This problem is particularly relevant in scientific projects where the truth of certain judgments can be confirmed or refuted for a long time;
- independence of the methodology and knowledge base development tools (except for external language editors) on which external language is used to develop the current knowledge base fragment (SCn, SCg, etc.), as well as on which identification language of sc-elements (English, Russian, etc.) is currently in use;
- availability of convenient and accessible tools of manual editing of the knowledge base using external languages;
- fixations of the entire history of changes in the knowledge base with the obligatory indication of authorship and the time of making each change, including both changes to its subject part (intended for the end user), and any changes associated with the development process, including commenting, approval or rejection of the proposed knowledge base changes, etc.;
- ensuring the integrity (consistency) of the knowledge base being developed at each point in time during its development, while in the early stages of development of automation tools, the degree of human participation in integrity ensuring can be significant and subsequently decrease. This requirement is largely due to the pursuit of the ideas of Agile-methodologies;
- reflexivity of the *ostis-system* development process, suggesting that the developed *ostis-system* itself becomes an active participant in the development process, that is, it can analyze current project tasks, processes aimed at its development and any related information.

In addition, when developing *ostis-systems*, the following requirements stay valid, which are valid when developing computer systems of any kind:

- availability of means for assessing the contribution of each participant to the development process, which allows calculating the amount of material remuneration of the development process participants, taking into account the overall activity of various participants when determining their roles and privileges within the hierarchy of developers, etc.;
- availability of tools and mechanisms to implement the process of material incentives for developers, in particular, investing in certain areas of development, micro-investment;
- availability of design management tools, including, at a minimum, means of current tasks fixation, their priorities, dependencies, deadlines and performers, tools of controlling the process of performing tasks, etc.;
- focus on the use of reusable components of dif-

ferent degrees of complexity, involving both the organization of the accumulation process, the specification and the search for components within the relevant library, and the organization of the process of integrating components from the library into the system being developed, as well as including new components into the library;

VI. The proposed approach to the organization of semantic computer systems development

Taking into account the formulated requirements, the following basic principles were used as the basis for the proposed approach to the development of *ostis-systems* and the means of automation of this process:

- tools for the *ostis-systems* development process automation are also implemented as *ostis-systems* and are built according to the same principles. This principle allows:
  - to ensure the modifiability of the tools themselves, as well as the hybridity of such tools, which is expressed in the possibility of combining within such tools different approaches (methodologies) to the development and verification of the knowledge base;
  - provide information support to developers and their learning directly in the process of working with automation tools due to the possibility of presenting in the knowledge base of such tools the information about the project, development methods, architecture and principles of operation of the tools themselves in a structured manner, as well as the ability to ask different questions;
  - the possibility of implementing tools for analyzing information about a project with the possibility of permanent expanding the functionality of such tools if necessary;
  - the possibility of implementing tools of various kinds of modeling with the possibility of permanent expanding the functionality of such tools if necessary;
- development process automation tools *ostis-systems* are embedded as a subsystem in the developed *ostis-system*, thus the knowledge bases and problem solvers of the developed system and automation tools are integrated. This ensures the reflexivity of the development process, that is, the possibility of the participation of the developed system in the development process itself. Due to this principle the following advantages appear:
  - eliminating the need to use the source code of the knowledge base and eliminating the stage of assembling and deployment of the system, as well as the need to restart the system to make changes. Thus, at any time there is a working, serviceable version of the *ostis-system*, which fully complies with the principles of Agile;
  - eliminating the need to document the system being developed (which is also consistent with the principles of Agile), since the knowledge base itself contains all the necessary information, which is accessed by the same tools as when solving any other problem;
  - when forming project tasks and discussing them, it becomes possible to appeal directly to fragments of the system's knowledge base, which have an arbitrary configuration, which makes it more flexible, for example, in the process of specifying problem parts of the knowledge base;
  - communication between developers is carried out on the same principle as the solution of problems by a team of agents, that is, all actions performed (including actions for the formation of natural language messages) are recorded in a common memory. At the same time, the authorship of the action, the object of the action, if any, and any other necessary information is indicated. This approach does not fully correspond to the principle of personal communication adopted in Agile, however, on the one hand, it does not exclude the possibility of personal communication of developers, and on the other hand, it makes it possible to develop a project by a distributed team of developers, including the case of open-source development;
  - it is possible not only to automatically detect errors and problems, but also to automatically generate tasks for correcting them within the framework of automation tools;
  - it becomes possible to record in the knowledge base not only the authorship of certain fragments, but also the whole process of agreeing and discussing the changes made, and, if necessary, even to have contradictory fragments in the knowledge base, the truth of each of which cannot be installed exactly. At the same time, the opinions of various experts who took part in the discussion, the arguments for and against, etc. are recorded. At the same time, solving problems in a knowledge base containing conflicting information remains possible by specifying the solution context for each problem, that is, that area of the knowledge base that is considered correct when solving a specific problem. Thus, the proposed approach not only makes the discussion of the changes **open** and **transparent**, but also makes it possible to record in the knowledge base **simultaneously**

any points of view on the same fragments, including contradictory.

The implementation of these principles implies the implementation of subsystems of the *ostis-systems* development automation tools, similar to all classes of existing automation tools for the development process of traditional computer systems. However, this task is rather time-consuming, and it would be inappropriate to make the possibility of *ostis-systems* development dependent on it. At the present stage of development, *OSTIS Technology* for solving many particular problems related to *ostis-systems* design, traditional tools can be used (some of mentioned advantages will be absent at this stage), while with the development of technology there will be a step-by-step transition from traditional tools to automation tools built on the basis of *OSTIS Technology*.

Besides the development automation tools *ostis-systems* embedded in the system being developed, ***IMS.ostis Metasystem*** [73] plays an important role in the development process.

IMS.ostis Metasystem is also an *ostis-system* and solves the following main tasks:

- informational support for *ostis-systems* developers, which assumes that the knowledge base of the metasystem in each current time is a complete formal description of the current version of *OSTIS Technology* (including a complete description of the metasystem itself, all the main models used in the technology, and also methods and tools for developing components of *ostis-systems*), as well as the availability of navigation tools in such a knowledge base;
- the accumulation of *ostis-systems* development experience and the implementation of the component approach, which is expressed by the presence of libraries of reusable components *ostis-systems*, as well as the means of component specification and search tools components based on their specifications.

To implement the principles discussed earlier, it is proposed to use an integrated ontological approach to the design of computer systems, discussed in [74]. This approach involves the construction of two ontological models:

- ontological model of the design object, in this case, *ostis-system*;
- ontological model of project activities aimed at the development of appropriate design objects;

Ontological models of *ostis-systems* and their components were considered in detail in the previously mentioned works [74], [4], [5].

Building an ontological model of a project activity aimed at *ostis-systems* development involves the development of *sc-models* of the following subject domains [4] and their corresponding ontologies (in SCn):

Subject domain of project activities
=> *particular subject domain**:
Subject domain of actions of knowledge bases sc-models developers
=> *particular subject domain**:
Subject domain of actions of problem solvers sc-models developers
<= *particular subject domain**:
Subject domain of actions and tasks

Subject domain of problem fragments of knowledge bases
=> *particular subject domain**:
- *Subject domain of incorrect fragments of knowledge bases*
- *Subject domain of incompleteness in the knowledge base*
- *Subject domain of information garbage*

All the listed *sc-models* of subject domains and corresponding ontologies are included in the relevant sections of the knowledge base of *IMS.ostis Metasystems*.

Next, we will consider in more detail the library of reusable components of *ostis-systems*, the *ostis-systems* development methodology, based on the family of subject domains and ontologies listed above, as well as the corresponding automation tools.

VII. Library of reusable components of semantic computer systems

Reuse of ready-made components is widely used in many industries related to the design of various types of systems, because it allows to reduce the complexity of development and its cost (by minimizing the amount of labor due to the absence of the need to develop any component), to improve the quality of the created content. The use of ready-made components assumes that the distributed component is verified and documented, and possible errors and limitations are eliminated or specified and known.

The basis for the implementation of the component approach within the *OSTIS Technology* is ***Library of reusable components of ostis-systems***, which is part of the *IMS.ostis Metasystem*.

It is important to note that since the library is part of the *IMS.ostis metasystem*, its replenishment is carried out in the same way as any other section of the knowledge base in accordance with the knowledge base editing mechanism discussed below.

General structure of *Library of reusable components of OSTIS*, presented in SCn-code:

Library of reusable components of OSTIS
= *Library of OSTIS*

= *reusable components of OSTIS*
= *reusable components of intelligent systems, build on OSTIS Technology*
<= *subdividing**:
{
- *Family of sc-models of computer systems interpretation platforms*
- *Library of reusable components of knowledge bases sc-models*
- *Library of standard components templates of computer systems sc-models*
- *Library of reusable components of problem solver sc-models*
- *Library of reusable components of user interfaces sc-models*
- *Library of typical subsystems of computer systems developed by OSTIS Technology*

}

Library of reusable components of OSTIS includes:

- set of such components;
- means of the specification of such components;
- tools of components search automation based on their specifications.

The *reusable component of OSTIS* is generally understood as a component of some ostis-system that can be used in another ostis-system. For this, at least two conditions must be met:

- it is technically possible to embed a component into a child ostis-system by either physically copying, transferring and embedding it into the designed system, or using a component hosted in the original system like a service, that is, without explicitly copying and transferring the component. The complexity of embedding depends, among other things, on the implementation of the component;
- use of the component in any ostis-systems, except for *IMS.ostis Metasystem*, is expedient, that is, a component cannot be a particular solution oriented to a narrow circle of tasks. It is worth noting, however, that in the general case almost every solution can be used in any other systems, the range of which is determined by the degree of generality and the domain dependence of such a solution.

From a formal point of view, each *reusable component of OSTIS* is a structure that contains all those (and only those) sc-elements that are necessary for the component to function in the child ostis-system and, accordingly, must be copied into it while including a component into one of such systems. The specific composition of this structure depends on the type of component and is specified for each type separately. In essence, this structure is a standard (or sample), which is copied when the corresponding component is included in the child system.

Each *reusable component of OSTIS* can be atomic or non-atomic, that is, it can consist of simpler self-contained components.

At any given time in the current state of the sc-memory, each reusable component can be fully represented, that is, all *sc-arcs of membership* that connect the *structure* corresponding to the component and all its elements are clearly present in memory, or it can be presented implicitly, for example, by setting the decomposition of this component into more particular ones.

A. Library of reusable knowledge base components

The main semantic classes of reusable components of knowledge bases stored in the library of components of knowledge bases include:

The main semantic classes of reusable components of knowledge bases stored in the library of components of knowledge bases include:

- semantic neighborhoods of various entities;
- ontologies of various subject domains;
- specifications of formal languages describing various subject domains;
- sections of the knowledge base of various semantic types (including non-atomic ones);
- knowledge base of entire subsystems that provide solutions to various problems;
- knowledge bases of applied system;
- and others.

Each reusable knowledge base component must be specified within the library. This specification includes the following minimum required information:

- information about the authorship of the component, i.e. the connection of the component with the sign of the author (individual, team, etc.);
- information about the atomicity or non-atomicity of the component;
- information about the semantic class of a component by specifying that the component belongs to a class of reusable components;
- information about the subject domain, a fragment of which is described in the component;
- description of the purpose of the component, its features;
- date the component was created and last modified;
- information about the dependent components for this, that is, such components that cannot be used separately from this. An example of such components are the components describing the ontologies of the subject domain of persons and the subject domain of historical personalities, since the first contains specifications of the concepts used to describe the objects of the second subject domain;

- information about the openness of the component and the possibilities of its use in various systems from the point of view of proprietary.

This list can be expanded if necessary.

An example of the specification of a reusable knowledge base component is shown in the figure 2.

Integration of the *reusable component of the knowledge base* into the system is reduced to merging of key nodes by identifiers and eliminating possible duplications and contradictions that could arise if the developer of the child system manually made any changes to its knowledge base.

To ensure the semantic compatibility of components of knowledge bases, it is necessary to:

- match the semantics of all used key nodes;
- match the main identifiers of the key nodes used in different components. After that, the integration of all components that make up the library in any combination is carried out automatically, without intervention by the developer, using the mechanisms proposed in [75].

Components automation tools include the following *sc-agents*:

- *agent of formation of a non-atomic component from the atomic components* – the task of this agent is to explicitly form a structure containing all the sc-elements that make up the indicated non-atomic component;
- *agent of dependency searching between components* – the task of this agent is to search for all components, without which the use of the specified component is impossible. In this case, the search is performed recursively, taking into account the dependence of other components;
- *agent of search for all non-atomic components*, which include the indicated component;
- *agent of component search*, within which the indicated concepts are described;
- *agent of component search by specification fragment*.

The most important component of the library is *Knowledge base kernel*, which is the basis for building the knowledge base of any system, since it contains the set of the top-level ontologies:

- *Subject domain of sc-elements*;
- *Subject domain of entities*;
- *Subject domain of sets*;
- *Subject domain of structures*;
- *Subject domain of knowledge*;
- *Subject domain of semantic neighborhoods*;
- *Subject domain of subject domains*;
- *Subject domain of ontologies*.

On the basis of the proposed knowledge base model, ontologies of subject domains describing the types of knowledge that are used in most intelligent systems were developed:

- *Subject domain of situations and events in sc-memory*;
- *Subject domain of relations and connections*;
- *Subject domain of parameters and values*;
- *Subject domain of logical formulas and logical ontologies*;
- *Subject domain of unified logical-semantic models of computer systems*;
- *Subject domain of numbers and number structures*;
- *Subject domain of actions and tasks*;
- *Subject domain of information constructions that do not belong to the SC-code*;
- *Subject domain of temporary entities*;
- *Subject domain of characters that are not elements of SC-code texts*;
- and others.

Together with the *Knowledge base kernel*, the ontologies of the specified subject domain make up *Extended knowledge base kernel*.

B. Library of reusable components of problem solvers

Classification of reusable components of problem solvers in SCn-code:

Library of reusable components of problem solvers
= reusable component of problem solvers
*<= subdividing**:
{
- *Library of reusable problem solvers*
- *Library of reusable atomic abstract sc-agents*
- *Library of reusable sc-text processing programs*

}

The *reusable abstract sc-agent* means the component corresponding to a certain *abstract sc-agent* that can be used in other systems, possibly as part of more complex *non-atomic abstract sc-agents*. The specified abstract sc-agent is included in the corresponding *structure* under the *key sc-element'* attribute. Each *reusable abstract sc-agent* must contain all the information necessary for the operation of the corresponding *sc-agent* in the child system.

Classification of reusable sc-agents in SCn-code:

Library of reusable abstract sc-agents
= reusable abstract sc-agent
*<= subdividing**:
{
- *Library of information search sc-agents*
- *Library of sc-agents of immersing integrable knowledge in the knowledge base*
- *Library of sc-agents for align ontology of integrable knowledge with the basic ontology of the current state of the knowledge base*

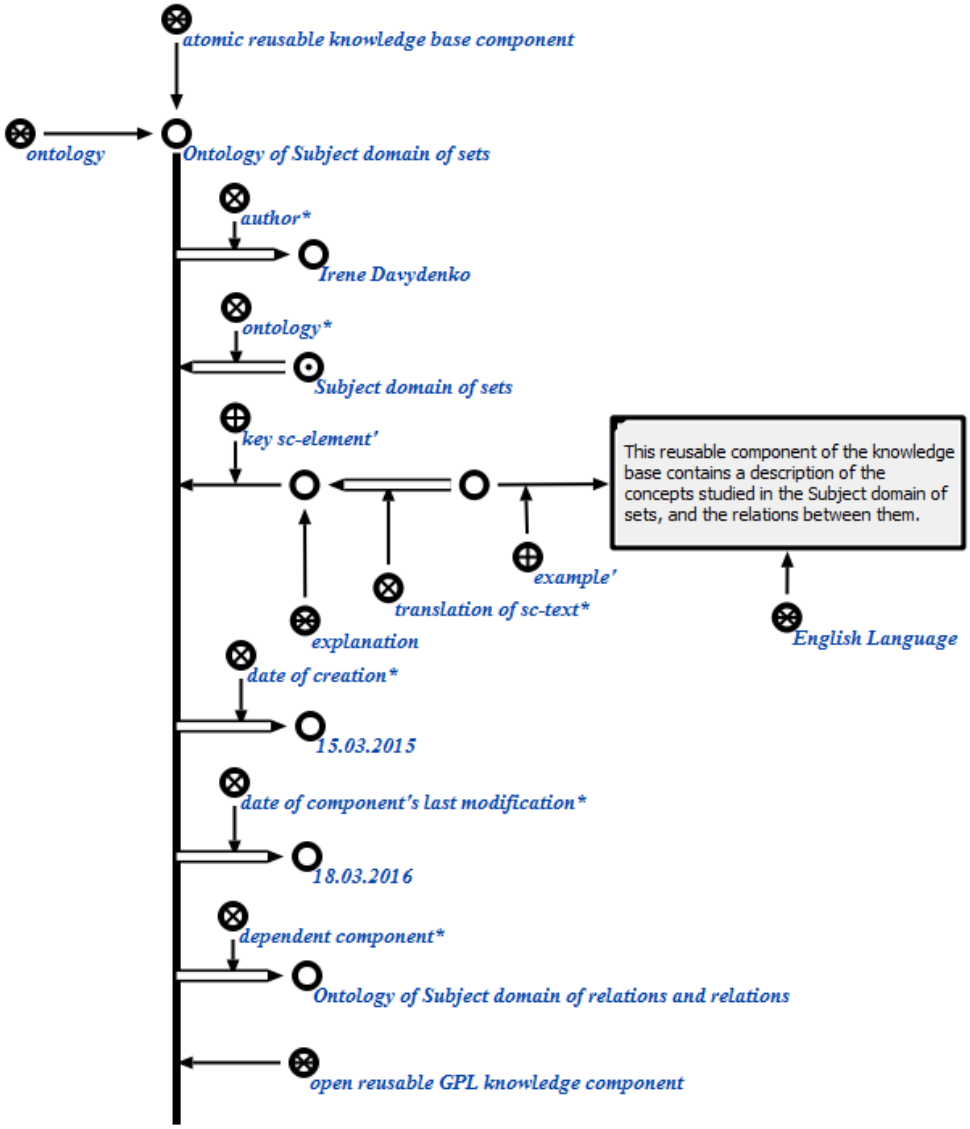

Figure 2. Example of the specification of a reusable knowledge base component

- *Library of sc-agents for planning explicitly defined tasks*
- *Library of sc-agents of logical inference*
- *Library of sc-models of high-level programming languages and their corresponding interpreters*
- *Library of sc-agents of knowledge base verification*
- *Library of sc-agents of knowledge base editing*
- *Library of sc-agents for knowledge developers activity automating*

}

For the convenience of working with the library of reusable components, tools for automating the search for components based on a given specification have also been developed, which are implemented as *non-atomic sc-agent*, which is decomposed into particular ones.

The following is the structure of such an agent in the SCn-code:

Automation tools of library of reusable abstract sc-agents

<= *decomposition of sc-agent**:

{

- *Abstract sc-agent of forming a non-atomic component from atomic components*
- *Abstract sc-agent of search for all non-atomic components of which a given atomic component*

is part
- *Abstract sc-agent search for all related components*
- *Abstract sc-agent of sc-agent search by initiation condition*
- *Abstract sc-agent of sc-agent search by the result of work*
- *Abstract sc-agent of scp-program search by input/output parameters*
- *Abstract sc-agent of sc-agents search for which the elements of a given set are the key sc-elements*

}

The *non-atomic component of problem solvers* is understood as such a component in which it is possible to select other components that can be used independently, separately from the source component. Most often, non-atomic sc-agents act as such non-atomic components, as part of which can be isolated self-sufficient sc-agents that can be used separately from the original non-atomic, or scp-program that are common to several agents and can be used not only as part of a non-atomic sc-agent. Thus, the task of the *Abstract sc-agent of forming a non-atomic component from the atomic* is the formation of a structure containing the complete sc-text of the non-atomic component, including the specifications of all sc-agents in its composition, as well as the texts of all the necessary scp-programs. The formation of such a structure is necessary in order to simplify the process of copying the specified component to other ostis-systems.

The *related component* is a component that is often used in the ostis-system simultaneously with some other component. Such a relationship between components is specified explicitly with the help of the *related component** relation. Examples of such components are some sc-agent and a user interface command that allows the user to initiate the execution of the specified agent with the given arguments. In this case, the sc-agent will function even without the presence of such a command in the system, however, to initiate it, it will be necessary to form the corresponding structure in the sc-memory manually.

Abstract sc-agent of sc-agents search for which the elements of a given set are the key sc-elements plays an important role when making changes in the knowledge base, in particular, when redefining any concepts. The specified sc-agent allows to identify those sc-agents that may require changes in the algorithm of work due to changes in the semantic interpretation of any concepts.

VIII. Methods of semantic computer systems development

As mentioned earlier, the development of the ostis-system comes down to the development of its knowledge base. In this section, we will take a closer look at the principles for the development of knowledge bases of ostis-systems, as well as some features specific to the development of ostis-system problem solvers.

In accordance with the requirements formulated above, the approach to the development of ostis-systems itself does not limit the use of any specific methods for the coordination of project activities. However, taking into account the current goals of OSTIS Technology, in particular, the need to develop the IMS.ostis Metasystem, currently the development methodology of ostis-systems and the corresponding tools implement an open-source project coordination mechanism borrowing the main ideas from

- traditional modern tools of collective development of such projects, for example [76]
- modern approach to the review of scientific articles adopted in the vast majority of scientific journals.

The most important features of the proposed methodology and tools are ensuring at each time point the integrity and consistency of the current version of the knowledge base directly during its operation, as well as the transparency and openness of the agreement mechanism, which are achieved through the previously presented principles.

The proposed method involves two main stages – the stage of creating the initial version of the developed ostis-system, whose knowledge base is synthesized from the components of the library of the reusable components of the ostis-systems knowledge base, and the stage of expanding and improving the knowledge base of the ostis-system being developed. The initial version of the ostis-system contains a set of knowledge and tools for problems solving, sufficient for the further development of the system.

The process of creating the starting version of the *ostis-system* can be divided into next main stages:

- selection and installation of *ostis-systems sc-models interpreting platform*;
- installation of *Knowledge bases kernel* from the library of reusable components of knowledge bases;
- installation of *Problem solver kernel* from the library of reusable components of problem solvers [5], that is, a set of basic reusable components of problem solvers necessary for the starting version of the ostis-system;
- installation of *Interface kernel* [77], i.e. a set of basic reusable components of sc-models of interfaces necessary for operation of the starting version of the ostis-system;
- installation of support system for collective development of knowledge bases.

After the basic configuration of the starting version of the ostis-system is assembled, the stage of developing the knowledge base begins, which is discussed in more detail below.

A. Structuring knowledge base from the point of view of the development process

To support the evolution of the ostis-system, it is necessary to distinguish sections of the knowledge base containing information on its development plans (the future of the system), current development processes, including the current processes for coordinating changes to the knowledge base, as well as information on completed knowledge base development processes in order to provide the ability to track and cancel changes to the knowledge base.

Thus, from the point of view of the development process, the knowledge base is conventionally divided into overlapping areas, describing the part of the knowledge base that is available for operation to the end user (*agreed part of the knowledge base*), a section containing information about the operation of the system, and a section containing information about the evolution of the system.

Figure 3 shows the structure of the knowledge base from the point of view of the development process.

The ***agreed part of the knowledge base*** is the part of the knowledge base that is agreed between all the participants in the development at the current time. The knowledge presented in this section of the knowledge base is available to the end users of the system in the operating mode of the system. The distinguishing of the agreed part of the knowledge base is necessary in order to be able to hide from the end user system information that is not directly related to the operation of an intelligent system.

In turn, *agreed part of the knowledge base* is divided into *subject part of the knowledge base*, *context of the subject part of the knowledge base within the Global Knowledge Base* and *computer system documentation* (figure 3).

By ***Global knowledge base*** we mean the global abstract semantic space of all knowledge accumulated by mankind to the current time [2].

Subject part of the knowledge base contains all information about the *subject domain* (or several interrelated subject domains within the same knowledge base) for which the developed system is intended to work. For example, the section describing Euclidean geometry in the geometry intelligent reference system.

The ***context of the subject part of the knowledge base within the Global knowledge base*** contains a specification of objects that are not directly studied in the subject part of the knowledge base of this system, but are related to it, i.e. used for description of concepts studied in the subject part of the knowledge base. For example, for the IMS.ostis Metasystem, these could be such concepts as *artificial intelligence* or *intelligent system*, for the system according to Euclidean geometry - historical information about Euclidean life, mathematics, etc.

The ***computer system documentation*** section contains documentation of the *ostis-system* itself, at a minimum, the specification of its knowledge base, problem solver and interface, as well as all the necessary manuals that provide the opportunity for learning in working with the system.

Section ***history and current processes of computer system operation*** includes the following sections:

- *computer system operation history*;
- *current processes of computer system operation*.

The ***computer system operation history*** section stores the history of the system's dialogue with its users.

The ***current processes of computer system operation*** stores the specifications of all actions performed by the *ostis-system* at the moment (which are *present entities*), as well as all temporary auxiliary constructions, generated by *sc-agents* in the process of work and not yet deleted. After performing these actions, their signs and specifications are transferred to the *computer system operation history* section.

The ***history, current processes and development plan of computer system*** section is decomposed into the following sections:

- *structure and organization of a computer system project*;
- *history of the computer system development*;
- *current development processes of computer system*;
- *computer system development plan*.

Section ***structure and organization of a computer system project*** describes the structure of a project aimed at the ostis-system development, including its subprojects and the roles of the developers responsible for each project.

The ***history of computer system development*** section contains specifications of project activities performed during system development (*past entities*), with the obligatory indication of the performers, the sequence and the result of each activity. The presence in the knowledge base of this kind of information will allow to rollback the changes made to the knowledge base, as well as to take into account completed design tasks when planning further work within the project.

The ***current development processes of computer system*** section contains specifications of approved and initiated project *actions* performed by the system developers at a given time (real entities), with the obligatory indication of the performers, the sequence and purpose of the implementation, and also all the information describing the proposals for editing the *subject part of the knowledge base* and *computer system operation history* and their discussion by administrators, managers and experts.

In the ***computer system development plan*** section there are specifications of project *actions* that are approved for execution, but have not yet been fulfilled for

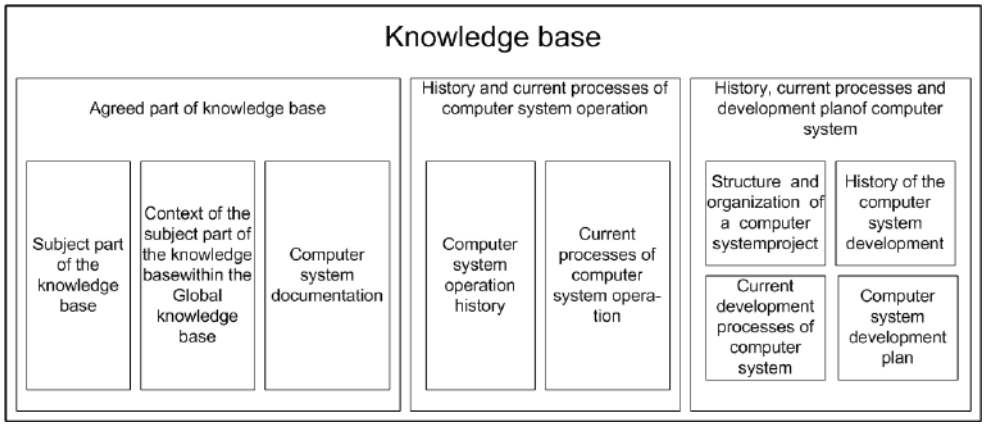

Figure 3. The structure of the knowledge base from the point of view of the development process

any reason, as well as all the information describing proposals for editing the section *history, current processes and development plan of computer system* and their discussion by administrators, managers and experts.

B. General mechanism for knowledge bases sc-models development

The process of a knowledge base development is a sequence of the following steps:

- Formation of the initial structure of the knowledge base, which involves:
  - the formation of the structure of the knowledge base sections corresponding to the above mentioned variant of structuring the knowledge base from the point of view of the development process;
  - identification of the described subject domains;
  - building a hierarchical system of the described subject domains;
  - building a hierarchy of knowledge base sections within the *subject part of the knowledge base*, which takes into account the hierarchy of subject domains constructed at the previous stage.
- Identifying knowledge base components that can be taken from a library of reusable knowledge base components and including them into the knowledge base that is being developed.
- Formation of project tasks for the development of missing fragments of the knowledge base and the assignment of tasks to developers.
- Development and coordination of knowledge base fragments, which, in turn, may later be included in the library of reusable knowledge base components.
- Verification and debugging of the knowledge base.

It should be noted that in the process of the knowledge base improving, stages 3–5 are performed cyclically.

Figure 4 shows a diagram reflecting the sequence of a knowledge base building steps according to the proposed methodology.

The basis of the methodology under consideration is a formal model of developer activity aimed at developing and modifying of knowledge bases, formal means for specifying proposals for editing a knowledge base, method for introducing changes to the knowledge base, formal means for specifying transition processes in the knowledge base, and formal means for specifying contradictions and incompleteness in the knowledge base.

To ensure the reflexivity of the intelligent system, in particular, the ability to automate the analysis of the history of the evolution of the knowledge base and generate plans for its development, all activities related to the development of the knowledge base are specified in this knowledge base by the same means as the subject part.

The process of creating and editing the knowledge base of the ostis-system is reduced to the formation of *proposals for editing* of a particular section of the knowledge base by developers (picture 5) and the subsequent consideration of these proposals by knowledge base *administrators*. In addition, it is assumed that, if necessary, *experts* can be involved in verifying incoming proposals for editing the knowledge base, and the development process is managed by *managers* of relevant knowledge base development projects. In this case, the formation of design tasks and their specification are also carried out using the mechanism of proposals for editing the relevant section of the knowledge base. Thus, all information related to the current processes of developing a knowledge base, history and plans for its development is stored in the same knowledge base as its subject part, that is, the part of the knowledge base accessible to the end user of the system. This approach provides wide opportunities to automate the process of knowledge bases

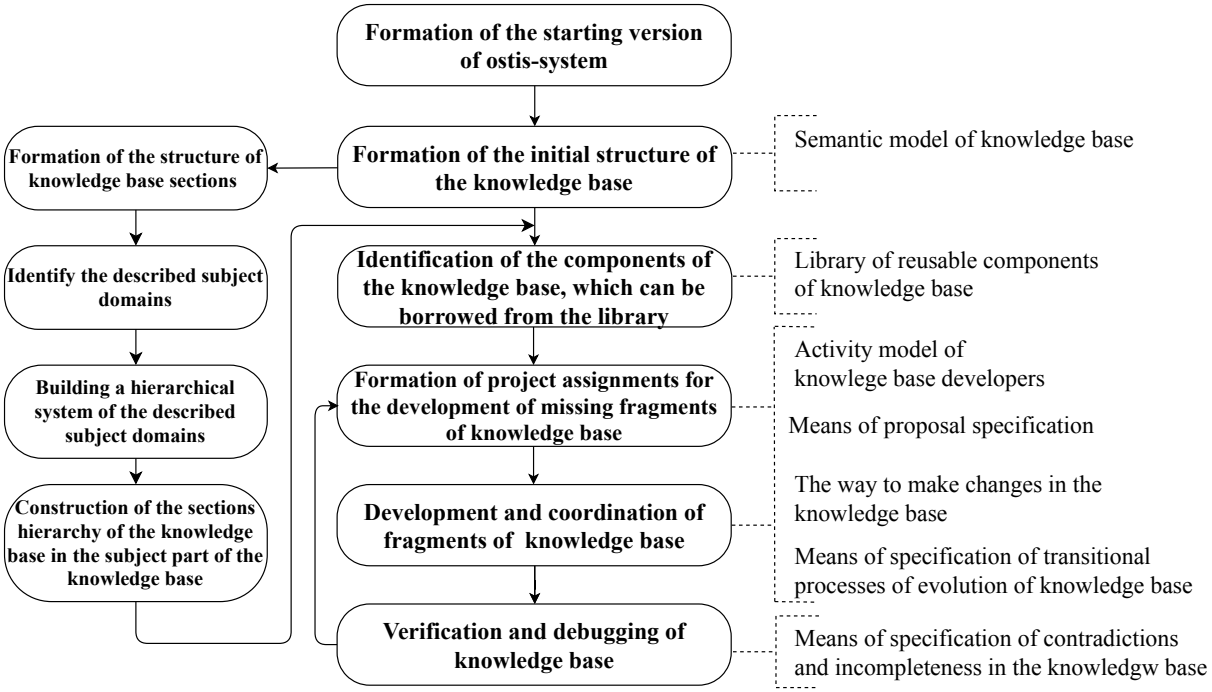

Figure 4. Methods of building and modifying of knowledge bases

creation, as well as subsequent analysis and improvement of the knowledge base.

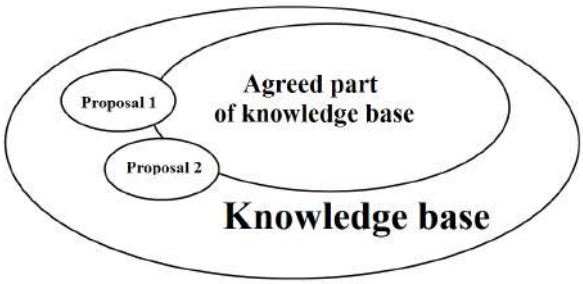

Figure 5. Illustration of the knowledge base editing process

Each *proposal for editing the knowledge base* is a structure containing sc-text that is proposed to be included in the agreed part of the knowledge base. The structure of such proposals may include signs of actions for editing the knowledge base, which are automatically initiated and executed by the relevant agents after the approval of the proposal.

Figure 6 shows the stages of developing a certain piece of the knowledge base, starting with the formulation of the project task and ending with the final approval or rejection of the proposal for editing the knowledge base.

The proposed methodology for developing a knowledge base is primarily focused on open-source projects for developing intelligent systems, where anyone can become a developer.

Next, we consider in detail the typology of developer roles and classes of actions they perform.

C. Typology of knowledge base developers

First of all, all users of any ostis-system are divided into *registered users* and *unregistered users*.

To describe this fact, the following relations are used in the knowledge base:

- ***unregistered user*** is a *binary relation* connecting *ostis-system* and *sc-element*, denoting *person* that did not pass the registration procedure in system;
- ***registered user*** is a *binary relation* connecting *ostis-system* and *sc-element*, denoting *person* that has passed the registration procedure in system.

An unregistered user has access to read the subject part of the ostis-system knowledge base. This type of users can work with the ostis-system in the operation mode, i.e. it can only set queries addressed to the subject part of the knowledge base (i.e., solve subject problems).

A registered user has access to read the entire knowledge base and make proposals to the entire knowledge base, can play the role of the end user of the ostis-system, that is, work in the operating mode, and also the role of its developer. At the same time, regardless of the role that a particular user performs, he can make proposals for editing any part of the knowledge base, which, depending

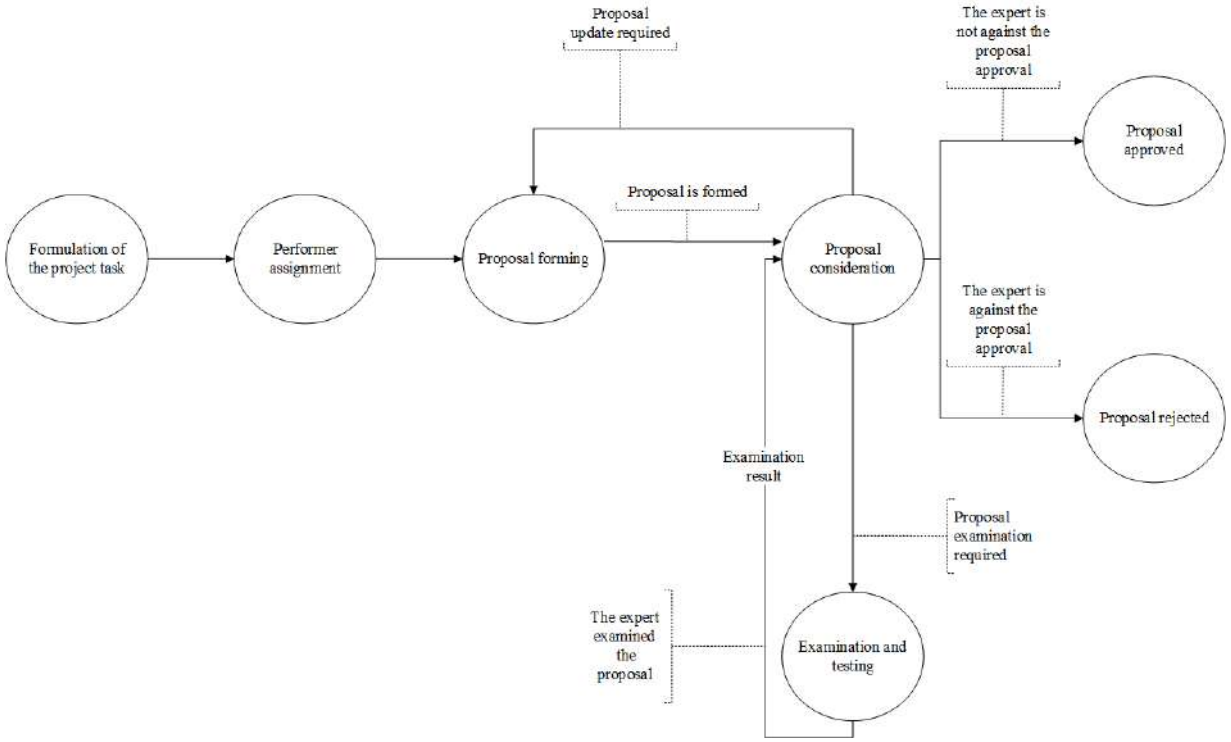

Figure 6. Illustration of the mechanism of making changes in the knowledge base

on its level, will either be automatically accepted or separately considered.

Among registered users there is a separate type of users - *developer*.

Developer is a binary relation connecting a project to develop a section of the knowledge base of the ostis-system (in the limit, the entire knowledge base) and a sc-element denoting a person who can be the developer of this section of the knowledge base, i.e. perform project tasks within this section.

In addition to operating the ostis-system, the *developer* can make proposals for changing any part of the knowledge base, leave comments on the such proposals. Among the developers, such roles as *administrator*, *manager* and *expert* are distinguished.

Administrator is a binary relation connecting a project to develop the knowledge base section of the ostis-system (in general, the entire knowledge base) and the sc-element denoting the person who is the administrator of this knowledge base section.

Tasks of *administrator* are:

- control of the integrity and consistency of the entire knowledge base;
- define access levels for other users;
- a decision regarding the acceptance or rejection of proposals in various parts of the knowledge base, including, if necessary, sending them for expertise;
- making changes in various parts of the knowledge base by using the appropriate editing commands (in this case, changes are automatically made out as proposals and entered into the section of the development history of the ostis-system).

If it is necessary to develop a knowledge base of a big size, a hierarchy of developers can be introduced corresponding to the hierarchy of sections of the knowledge base being developed. In this case, the approval of a proposal by the administrator of the lower level section does not lead to the integration of the proposal into the appropriate section, but requires consideration by the higher level administrators. The final decision is made by the administrator of the entire knowledge base. Figure 7 shows a fragment of the knowledge base that describes the hierarchy of knowledge administrators.

Manager is a binary relation connecting a project to develop the knowledge base section of the ostis-system (in general, the entire knowledge base) and the sc-element denoting the person who is the manager of this knowledge base section.

Tasks of the *manager* are:

- planning the amount of work on the development of the knowledge base;
- detailed elaboration of project tasks for subtasks, formulation of project tasks, assignment of performers of project tasks;

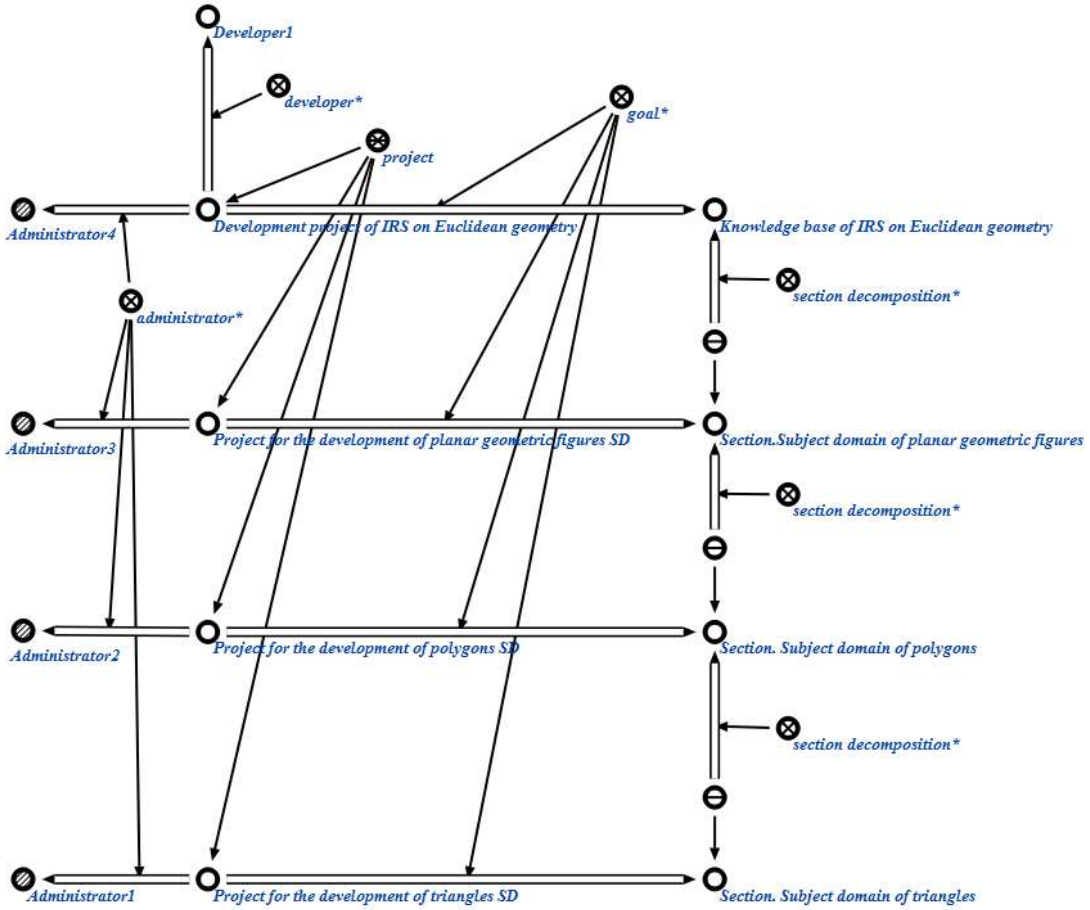

Figure 7. Knowledge base administrators hierarchy

- setting priorities and deadlines for tasks completing;
- control the timing of project tasks.

Manager makes changes to the part of the relevant section that describes the project tasks using the appropriate editing commands, and the changes are automatically presented as proposals and entered into the section describing the development plan of the ostis-system. Thus, the *manager* is the *administrator* of the specified section.

Expert is a binary relation connecting a project to develop a section of the knowledge base of the ostis-system (in general, the entire knowledge base) and an sc-element denoting a person who is an expert of this knowledge base section.

Tasks of the *expert* are:

- verification of the results of the project tasks;
- if necessary, the expert can leave comments on any fragment of the knowledge base regarding its correctness. All comments fall into the section describing the plan for the development of a computer system.

In addition, any participant in the development process has the opportunity to leave a natural language commentary to any fragment or element of the knowledge base, thus, any issues related to the specified fragment or element of the knowledge base can be discussed. Such comments fall into the knowledge base section *current processes of computer system development*.

An example of using these relations to indicate the roles of developers in a project to create a knowledge base in the SCg is presented in the figure 8:

D. Ontology of actions of knowledge base developers

In the process of developing the knowledge base of the ostis-system, each of the users involved in the development process uses a specific set of commands corresponding to the knowledge base editing mechanism described above. Each such command corresponds to a certain class *actions in sc-memory* [5]. All such actions are combined into a common class *actions of the knowledge base developer*.

For the purpose of subsequent automation, some classes of actions of the knowledge base developer are

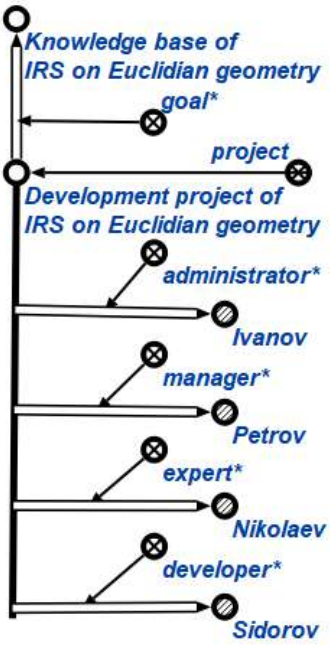

Figure 8. The roles of developers in the knowledge base

formally specified and detailed to the level of elementary transformations in the system memory.

To identify classes of actions, the names of the following form will be used: *action. «action class name»*.

The hierarchy of the actions of the knowledge bases sc-models developers in the SCn-code, taking into account the roles considered and the corresponding responsibilities (the actions of the manager are not shown, since they repeat the actions of the administrator, but are applicable only for certain sections):

knowledge bases sc-models developer action
⊃ *action of knowledge base ordinary developer*
  ⊃ *action. build a new fragment for inclusion in the knowledge base*
  ⊃ *action. modify the proposal for editing the knowledge base*
  ⊃ *action. make a proposal for editing the knowledge base*
    ⊃ *action. form a project task proposal*
    ⊃ *action. form a project task performer proposal*
⊃ *action of knowledge base administrator*
  ⊃ *action. consider a proposal for editing the knowledge base*
  ⊃ *action. approve a proposal for editing the knowledge base*
  ⊃ *action. reject a proposal for editing the knowledge base*
  ⊃ *action. create a task for the verification of the proposal*
  ⊃ *action. approve the result of the proposal verification*
  ⊃ *action. reject the result of the proposal verification*
⊃ *action of knowledge base manager*
⊃ *action of knowledge base expert*
  ⊃ *action. verify the specified structure*
  ⊃ *action. approve verifying proposal*
  ⊃ *action. reject verifying proposal*
  ⊃ *action. create a task for consideration of the proposal verification result*

For the specification of *knowledge bases sc-models developer action* and *structures* describing the proposal for editing the knowledge base, relations such as *proposal**, *approved**, *rejected**, *new version**.

An example of a specification of a proposal for editing a knowledge base using the above relations and classes of actions in the SCg-code is presented in the figure 9:

E. Typical mistakes and difficulties in the development of knowledge bases of semantic computer systems

As mentioned earlier, one of the important tasks of *IMS.ostis Metasystem* is the information support for the developers of sc-models of ostis-systems, which also involves learning of developers using typical examples and exercises. To solve this problem, a section of the knowledge base of the Metasystem IMS.ostis was developed, describing typical errors and difficulties in developing of knowledge bases sc-models. In this section we consider some of the most common examples of such problems.

1) It is necessary to distinguish:
- Syntactic typology of *sc-elements* (*sc-node*, *sc-edge*, *sc-arc*);
- Semantic typology of *sc-elements*.

That is, *membership pair* $\neq$ *sc-arc of membership*

membership pair
⊃ *sc-arc of membership*

2) It is necessary to distinguish:
- *variable sc-arc of membership*;
- *constant sc-arc of membership*.

Sometimes, for example, *constant sc-arcs of membership* can go out of *constant sc-nodes*, but are included in *variables sc-elements* (figure 10).

3) It is necessary to distinguish:
- *permanent (stationary) sc-arcs of membership*;
- *temporal (situative) sc-arcs of membership*.

4) It is necessary to distinguish:
- the entity being described;
- abstract sign (internal sign, sc-sign) of the described entity;
- external sign (identifier, designation, name) of the described entity;
- specific occurrence of an external sign of the described entity in a specific sign structure;

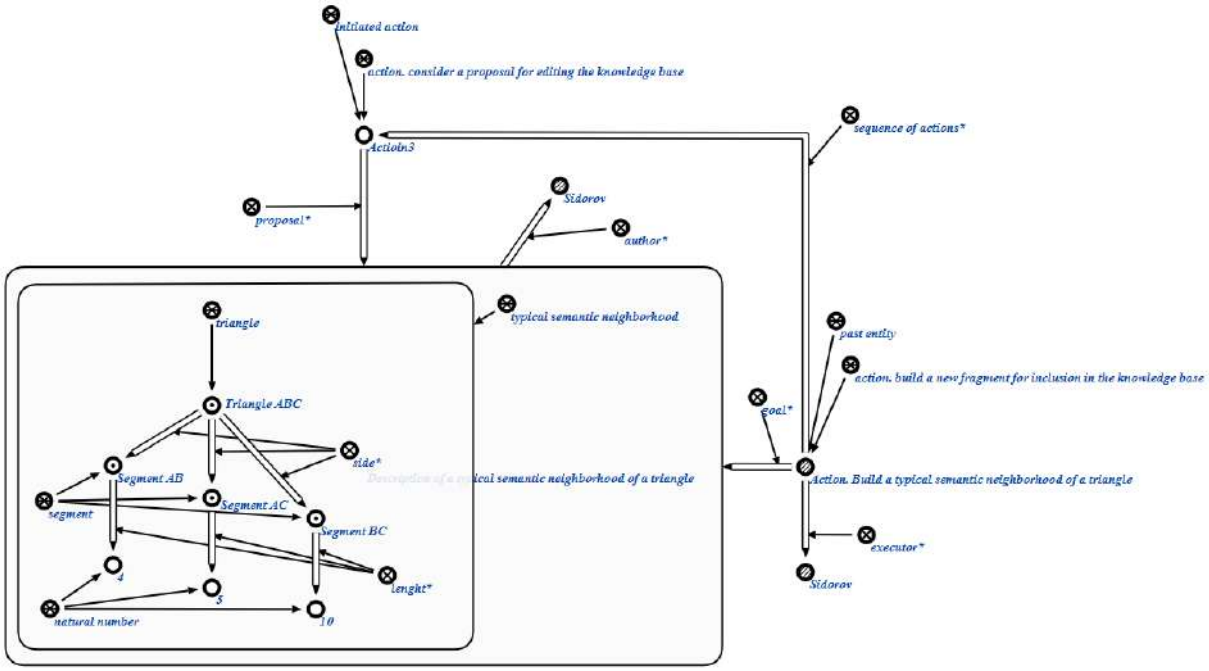

Figure 9. Specification of the proposal for editing the knowledge base

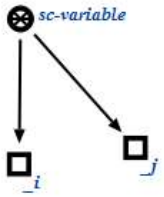

Figure 10. Incidence of constant sc-arcs and variable sc-nodes

- unambiguous specification of the described entity, represented in sc-memory or in any external language.

5) It is necessary to distinguish:

- set of *sc-elements* itself, which can be a described entity;
- *sc-element*, which is the sign of the corresponding (denoted by it) set of *sc-elements*;
- *sc-text*, which describes the connection of the sign of some set of *sc-elements* with all sc-elements that are members of this set.

6) It is necessary to distinguish:

- concept;
- natural language text (text file), which is one of the wording of the concept definition;
- natural language text that is an another formulation of the concept definition;
- text, which is the natural language formulation of a statement, which could be a concept definition, but is not (a statement of a defining type);
- sc-node denoting a statement, which is the concept definition, presented in the SC-code;
- sc-node denoting the entire sc-construction, which is the concept definition, presented in the SC-code.

7) It is necessary to distinguish:

- concept of set;
- concept of a set of sc-elements (semantically normalized set, sc-set).

8) It is necessary to distinguish:

- sc-sign *atomic logical formula*;
- sc-sign *non-atomic logical formula*;
- sc-sign of the full sc-text of the logical formula.

For an atomic logical formula, the sc-sign of this formula coincides with the sign of its full text.

9) It is necessary to distinguish:

- concrete number as a sign of the corresponding abstract entity;
- unambiguous specification of this number, for example, in one or another number system;
- string of digits, which is the external identifier (name) of this number corresponding to a particular number system.

10) It is necessary to distinguish:

- constant sc-element that is a sign of a specific

known (identified), uniquely defined, specified entity – *known sc-constant*;
- variable sc-element that is a sign of an arbitrary entity from some additionally defined (using logical statements) set of entities – *sc-variable*;
- constant sc-element, which is a sign of a specific entity, but not currently known – *unknown sc-constant*.

11) It is necessary to distinguish:
- *sc-constant* which is a member of a given set;
- *sc-variable*, any value of which is a member of a given set;
- *sc-variable*, which itself is a member of a given set (for example, a structure).

12) It is necessary to distinguish:
- sign of some specific (constant) entity class. At the same time, the elements (instances) of a class can be signs of other classes, signs of variables, signs of specific temporary entities, and signs of specific permanent entities;
- sign of some constant element (instance) of the above class;
- sign of a specific subset (subclass) of the specified class;
- sign of some arbitrary (variable) entity, possible values of which can only be signs of elements of the considered class of entities;
- sign of some arbitrary (variable) entity, one of the values of which is the sign of the entity class itself.

13) It is necessary to distinguish a section describing the subject domain and the subject domain being described

14) It should be remembered that for binary oriented relations there is no semantic need to introduce inverse relations, i.e. semantically, all links of each binary relation are also links and its inverse relation and vice versa (sin*=arcsin*, be a subset*=be a superset*)

15) It is necessary to distinguish:
- sign of *non-role relation*;
- sign of *role relation* corresponding to a given *non-role relation*;
- signs of *relation domains* (a domain is a set of those and only those entities that, in the tuples of a given relation, perform the specified role).

16) It is necessary to distinguish the connection of *membership* and *inclusion** (figure 11).

17) It is necessary to distinguish:
- case when the element *ei* is included in the set *si*, while performing multiple roles at the same time (figure 12)
- case when the element *ei* is included in the set *si* multiple times. Moreover, within the framework of different occurrences, the specified element may perform different roles (figure 13)

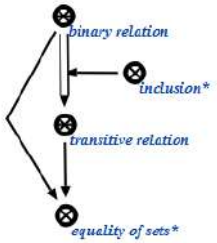

Figure 11. Membership and inclusion

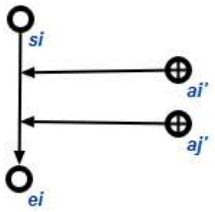

Figure 12. Multiple element roles in the set

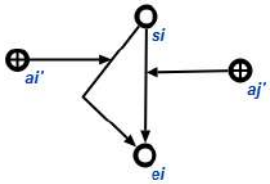

Figure 13. Multiple occurrences of an element in a set with different roles

18) It is necessary to distinguish the sc-element, denoting some described entity and a singleton, the only elements of which is the sign of this entity (figure 14)

19) It is necessary to distinguish the relation between classes and the relation between instances of these classes (figures 15, 16)

F. Methods for sc-models of problem solvers development

The proposed methods for problem solvers constructing and modifying includes several stages. Figure 17 presents a list of such stages, indicating the sequence of their execution.

The considered methods can be applied both in the development of hybrid solvers and in the development of simpler solvers, since from a formal point of view all of them are treated as a non-atomic abstract sc-agent.

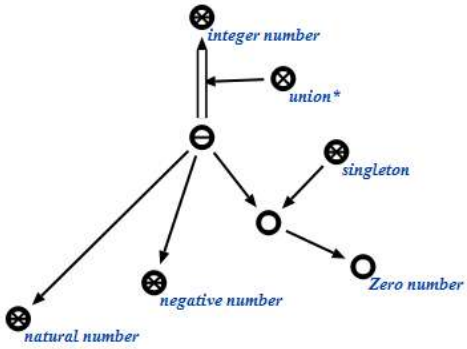

Figure 14. The difference of the sign of the entity and the singleton containing this sign

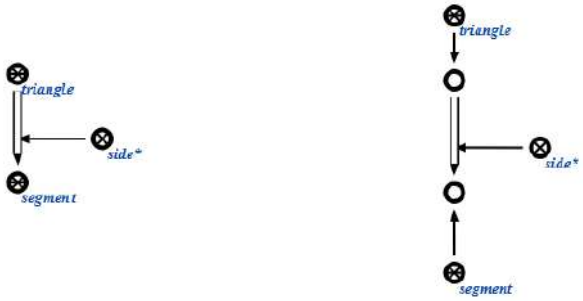

Figure 15. Incorrect (left) and correct (right) example of using the concepts of a triangle and a segment

Stage1. Requirements formation and problem solver specification

At this stage, it is necessary to clearly identify the problems that should be solved by the problem solver, consider the intended ways of solving them and, based on this analysis, determine the place of the future solver in the general hierarchy of solvers. The importance of this stage lies in the fact that, with proper classification,

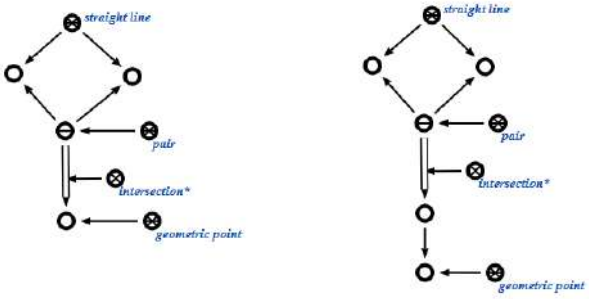

Figure 16. Incorrect (left) and correct (right) example of using the concept of intersection*

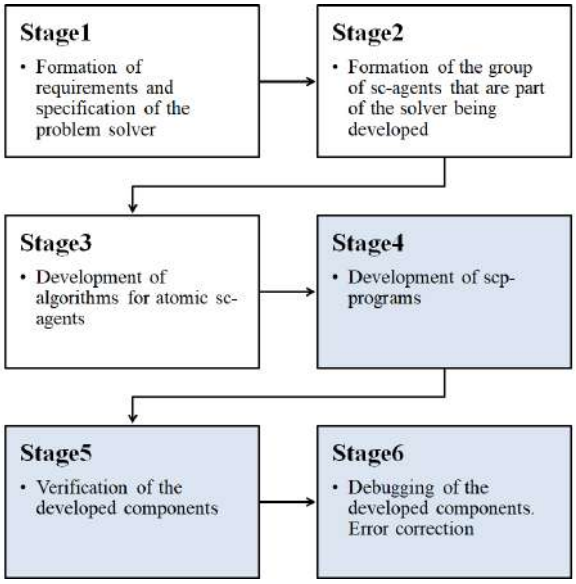

Figure 17. Stages of the process of problem solvers constructing and modifying

there is a possibility that there is already an implemented version of the required solver in the component library. Otherwise, however, the developer has the opportunity to include the developed solver into the component library for later use. These facts are due to the fact that the structure of the library of problems solvers components is based on the semantic classification of such solvers and, accordingly, of their components.

With an insufficiently precise specification and classification of the solver being developed, it is more likely that a suitable solver will not be found in the component library, even if it is there, and the newly developed solver cannot be included in the library. Thus, the idea of reuse of already developed components will be broken, which will significantly increase the cost of such a solver development.

Stage2. Formation of a sc-agents collective that are part of the developed solver

In the case when it is not possible to find a ready-made solver in the library that meets all requirements, it is necessary to distinguish and specify all the components of such a solver.

The result of this stage is a list of fully specified *sc-agents*, which will be part of the developed solver, with their hierarchy up to *atomic sc-agents*. Within this stage, it is very important to design a group of agents in such a way as to maximize the use of reusable components already presented in the library, and in case of the necessary component absence, be able to include it in the library after implementation. Depending on the complexity of the solver being developed, such components can include both *atomic sc-agents* and whole

teams of sc-agents (*non-atomic sc-agents*).

When developing the list of agents (including their specifications) it is necessary to follow a number of principles:

- each sc-agent being developed should be as independent as possible, that is, the set of key nodes of this sc-agent should not include concepts that are directly related to the subject domain under consideration. The exceptions are concepts from general subject domains that are interdisciplinary in nature (for example, the *inclusion** relation or the *action* concept). This rule can also be violated if the sc-agent is auxiliary and is focused on processing a particular class of objects (for example, sc-agents that perform arithmetic calculations can work directly with specific relations *addition** and *multiplication**, etc.). All the sc-agent information needed to solve the problem must be extracted from the semantic neighborhood of the corresponding initiated action. Obviously, the sc-agent, developed with these requirements, can be used to design a larger number of osti-systems than if it was implemented with a focus on a particular subject domain. After development and debugging is completed, such a sc-agent should be included in the *Library of reusable abstract sc-agents*;
- it is important to distinguish the concept of *sc-agent* and *agent program* (including agent scp-program). The interaction of sc-agents is carried out exclusively through the specification of information processes in common memory, each sc-agent responds to a certain class of events in sc-memory. Thus, each sc-agent corresponds to a condition of initiation and one agent program that starts automatically when an appropriate condition of initiation occurs in the sc-memory. In this context, various subprograms can be called as many times as necessary. However, it is important to distinguish the initiation of the sc-agent, which occurs when the corresponding construction appears in the sc-memory, and the subprogram call by another program, which implies an explicit specifying of the called subprogram and the list of its parameters;
- each sc-agent should independently verify the completeness of its own initiating condition in the current state of sc-memory. In the process of problem solving, a situation may arise when several sc-agents reacted to the appearance of the same structure. In this case, the execution continues only those of them, the condition of initiation of which is fully consistent with the situation. The remaining sc-agents in this case stop execution and return to the standby mode. The implementation of this principle is achieved by carefully specifying the specifications of the developed sc-agents. In the general case, the initiation conditions for several sc-agents may coincide, for example, in the case when the same task can be solved in different ways and it is not known in advance which of them will lead to the desired result;
- it is necessary to remember that a non-atomic sc-agent from the point of view of other sc-agents that are not part of it must function as an integral sc-agent (perform logically atomic actions), which imposes certain requirements on the specifications of the atomic sc-agents included in its composition: as a minimum, it is necessary that at least one atomic sc-agent is present in the composition of a non-atomic sc-agent, the initiation condition of which completely coincides with the initiation condition of this non-atomic sc-agent;
- if necessary, the implementation of a new sc-agent should be guided by the following principles for atomic abstract sc-agents design:
  - •• the designed sc-agent should be as independent as possible from the subject domain, which will enable it to be used in the development of solvers for the maximum possible number of osti-systems in the future. At the same time, universality implies not only minimizing the number of key nodes of the sc-agent, but also distinguishing the class of actions performed by this sc-agent in such a way that it makes sense to include this sc-agent into the *Library of reusable abstract sc-agents* and use it when developing solvers of other ostis-systems. One should not artificially link a set of actions into one sc-agent and, conversely, dismember one self-sufficient action on sub-actions: this will cause difficulties in understanding how sc-agents work by developers and will not allow using sc-agent in some systems (for example, in learning systems which should explain the decision-making way to the user);
  - •• the act of activity of each sc-agent (the action performed by this sc-agent) must be logically consistent and complete. It should be remembered that all sc-agents interact exclusively through common memory and avoid situations in which the initiation of one sc-agent is performed by explicitly generating a known initiation condition by another sc-agent (i.e., in fact, explicitly direct calling one sc-agent by another);
  - •• it makes sense to separate into sc-agents those relatively large fragments of the implementation of a certain general algorithm that can be executed independently of each other;
- when combining sc-agents into teams, it is recommended to design them in such a way that they can

be used not only as part of the non-atomic abstract sc-agent considered. If this is not possible and some sc-agents, being separated from the team, lose their meaning, it is necessary to indicate this fact when documenting the sc-agents;
- the actual initiator of the sc-agent launch via common memory (the author of the corresponding construction) can be either the system user directly or another sc-agent, which should not be reflected in the work of the sc-agent itself.

Stage3. Development of algorithms for atomic sc-agents

Within the framework of this stage, it is necessary to think over the algorithm of each developed *atomic sc-agent*. The development of the algorithm implies the distinguishing of logically consistent fragments in it, which can be implemented as separate *scp-programs*, including those executed in parallel. Thus, there is a need to speak not only about the *Library of reusable abstract sc-agents*, but also about *Library of reusable programs for sc-texts processing* in various programming languages, including *Library of reusable scp-programs*. Due to this, part of the scp-programs that implement the algorithm of the operation of a certain sc-agent can be taken from the corresponding library.

It is important to remember that if *sc-agent* generates any temporary structures in memory during the work, then at the completion of the work it is necessary to delete all information, the use of which in the system is no longer advisable (to remove information garbage). The exceptions are situations when such information is necessary for several *sc-agents* to solve one problem, but after solving a problem, the information becomes useless or redundant and requires removal. In this case, a situation may arise when none of the *sc-agents* is able to remove the garbage. In this case, there is a need to talk about the inclusion of specialized *sc-agents* into the solver, whose task is to identify and destroy information garbage.

Stage4. Implementation of scp-programs

The final stage of development is the implementation of previously specified *scp-programs* or, if necessary, programs implemented at the platform level.

Stage5. Verification of the developed components

The verification of the developed components can be carried out both manually and using the specified tools that make up the automation system for constructing and modifying of problem solvers built on the base of OSTIS Technology.

Stage 6. Debugging of developed components. Error correction

The debugging phase of the developed components, in turn, can also be divided into more specific stages:

- debugging of individual scp-programs or programs implemented at the platform level;
- debugging of individual atomic sc-agents;
- debugging of non-atomic sc-agents included in the problem solver;
- debugging of the entire problem solver.

Note that *Stage5* and *Stage6* can be executed in parallel and are repeated until the developed components meet the necessary requirements.

G. Ontology of the activity of problem solvers developers

In the framework of the proposed approach, the methods for constructing and modifying problem solvers is based on the formal ontology of the activities of such solvers developers.

It is important to note that, according to the model mentioned earlier, the problem solver is a *abstract sc-agent*, in connection with which the development of a solver comes down to the development of such an agent.

A fragment of a formal ontology of activity aimed at constructing and modifying problem solvers in the SCn-code looks as follows (for convenience of reading, the relations defining the order of actions are omitted):

action. develop an osti-system problem solver
= *action. develop an abstract sc-agent*
<= *subdividing**:
{
- *action. develop an atomic abstract sc-agent*
- *action. develop non-atomic abstract sc-agent*

}
=> *abstract subaction**:
- *action. specify an abstract sc-agent*
- *action. find an abstract sc-agent in the library that satisfies the given specification*
- *action. verify sc-agent*
- *action. debug sc-agent*

action. develop a platform-independent atomic abstract sc-agent
=> *abstract subaction**:
- *action. decompose a platform-independent atomic abstract sc-agent into scp-programs*
- *action. develop an scp-program*

action. develop a non-atomic abstract sc-agent
=> *abstract subaction**:
- *action. decompose a non-atomic abstract sc-agent into particular*
- *action. develop an abstract sc-agent*

action. develop an scp-program
=> *abstract subaction**:
- *action. specify scp-program*
- *action. find in the library an scp-program that satisfies the given specification*
- *action. implement the specified scp-program*
- *action. verify scp-program*
- *action. debug scp-program*

action. verify sc-agent
<= *subdividing**:
{
- *action. verify atomic sc-agent*
- *action. verify non-atomic sc-agent*
}

action. debug sc-agent
<= *subdividing**:
{
- *action. debug atomic sc-agent*
- *action. debug non-atomic sc-agent*
}

The presence of such a formal ontology allows, firstly, to partially automate the process of constructing and modifying solvers, and secondly, to increase the effectiveness of information support for developers, since this ontology is included in the knowledge base of the IMS.ostis Metasystem.

IX. Automation tools for the development of semantic computer systems

A. Architecture of automation tools for the development of knowledge bases sc-models

To reduce the complexity of the process of developing knowledge bases and reduce the requirements for developers in the framework of OSTIS Technology, tools have been developed to automate the processes of knowledge bases development and information support for such knowledge bases developers.

The information support tools for developers are implemented in the form of the previously mentioned *IMS.ostis Metasystems* [73].

Tools for automating knowledge base development processes are implemented in the form of *system for the collective knowledge bases development support* (SKBD). An important aspect of the knowledge bases development support is support of activities of knowledge base developers directly during the operation of the system being developed, which is possible due to the fact that the support system for the collective development of knowledge bases is embedded as a subsystem into each developed system.

Figure 4 shows the stages of the process of developing a knowledge base in accordance with the methods described above. Actions performed by developers in the first two stages cannot be fully formalized, and therefore their implementation cannot be fully automated.

Thus, in the framework of the *system for the collective knowledge bases development support*, the actions of developers carried out in the last three stages are automated. These actions are performed cyclically throughout the entire life cycle of the system being developed (figure 4).

The architecture *system for the collective knowledge bases development support* is presented in the figure 18. As can be seen from the figure, the system is an ostis-system and interacts with the IMS.ostis Metasystem, which includes a library of reusable components, which allows, on the one hand, to take components available in the library in accordance with the proposed methods of knowledge bases developing, on the other hand – to provide the opportunity to replenish the library with new components obtained in the process of knowledge bases development.

Let us consider in more detail the composition of each system component.

B. The knowledge base of system for the collective knowledge bases development support

The knowledge base of *system for the collective knowledge bases development support* includes sections containing all the knowledge necessary to support the process of the knowledge base developing and evolving.

Such knowledge includes:

- set of *top-level ontologies* necessary for the functioning of the SKBD itself and being the basis for building knowledge bases of the systems being developed. These ontologies are part of the previously considered *Knowledge base kernel*;
- formal *ontology of the subject domain of developers' activities aimed at knowledge bases developing and modifying*, including a description of the typology of roles of developers of knowledge bases, classification of developer actions, as well as formal means of specifying proposals for editing a knowledge base. The concepts included in this ontology were discussed above;
- ontology of the subject domain of problem structures in knowledge bases, i.e., those structures that describe incomplete, incorrect or redundant information in the knowledge base;
- means for specifying changes and transients in the knowledge base.

1) Problem structures in knowledge bases and their typology: One of the tasks of the *system for the collective knowledge bases development support* is the identification of problem structures in the knowledge base with the aim of correcting them.

Search and elimination of incompleteness, incorrectness and information garbage is carried out on the basis of:

- *ontologies of completeness*, which formally set the requirements for the completeness of the specified subject domains in the sc-memory;
- ontologies, within which classes of constructions are specified, representing incorrectness and informational garbage in the respective subject domains.

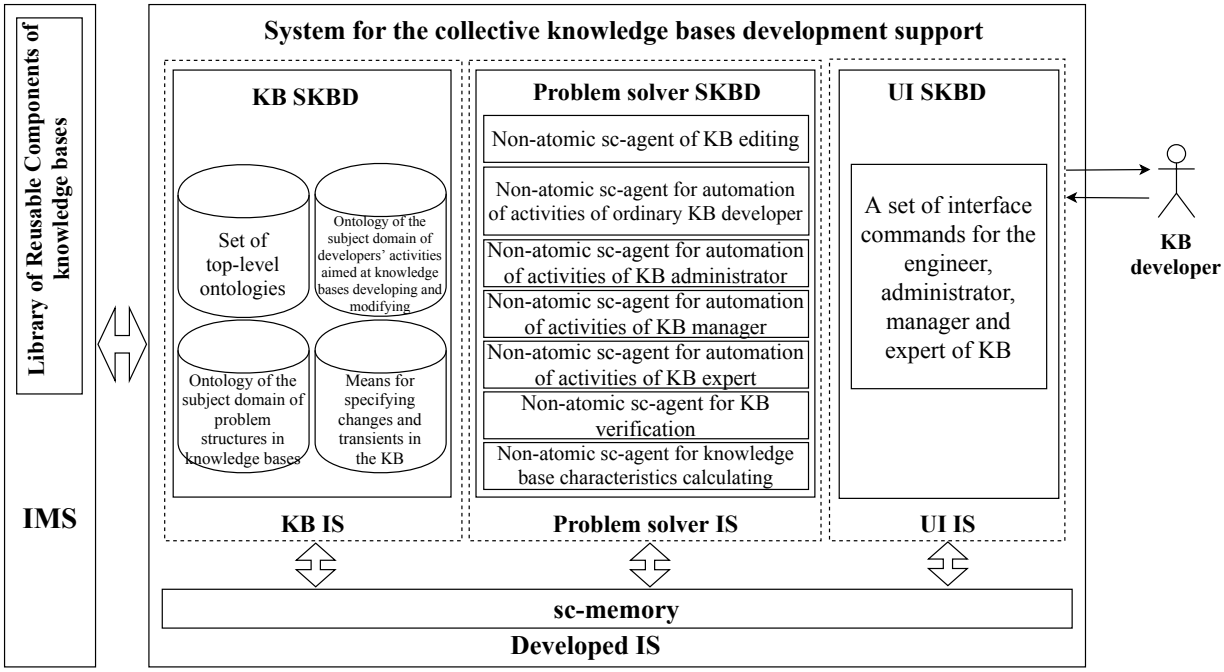

Figure 18. Architecture of system for the collective knowledge bases development support

The selection of classes of problem structures in the knowledge base allows us to specify such structures for knowledge base developers and for their automatic processing by agents.

In addition, the specification of the problem structures in the knowledge base allows the system to analyze its own knowledge base for correctness, completeness and redundancy, evaluate acquired knowledge and skills, which ensures the property of reflexivity of the intellingent system.

Consider the typology of such structures:

- *incorrect structure*;
- *structure describing the incompleteness in knowledge base*;
- *informational garbage*.

Under the *incorrect structure* we mean a structure containing a fragment of the knowledge base, in which in any way revealed any incorrectness. Additional concretization of the fact of incorrectness can be carried out by adding this structure to a particular class of incorrect structures or by specifying additional relations specifying this structure, for example, the *contradiction* relation.

The *structure describing the incompleteness in the knowledge base* means a structure containing a fragment of a knowledge base that lacks any information that is necessary (or at least desirable) for an unambiguous and complete understanding of the meaning of this fragment.

By *informational garbage* is meant a structure containing a fragment of the knowledge base, which for some reason has become unnecessary and requires removal. The formation of a structure describing such a fragment of the knowledge base, and, accordingly, the removal of its contents can be performed both by the sc-agent that generated the fragment and by the specialized sc-agents of garbage collection.

The following are the selected classes of *incorrect structures*:

- *structure that contradicts the property of uniqueness* (a special case of this class of structures is the class *Cantor set contains a repeating element*);
- *cycle within order relation* – failure to comply with the antisymmetry property for the order relation;
- *mismatch of elements tuple with relationship domains*;
- *power mismatch of relation arity*;
- *elements of a single subdividing have a non-empty intersection*;
- *mismatch of a fragment of the knowledge base with a logical statement*;
- and more.

As an example, consider the detection in the knowledge base of the contradictions associated with the indication of the angle in a right-angled triangle. As a result of fragment analysis, two contradictions arose:

- contradiction with the definition of a right triangle (figure 19);
- a contradiction with the theorem on the sum of the angles of a triangle (figure 20).

Among the situations that describe the incompleteness of the knowledge base, the following can be distinguished:

- *specified the main identifiers of a given entity for some, but not all external languages*;
- *system identifier is specified for the given entity, but the main identifiers for all external languages are not specified*;
- *the definition or explanation for the concept of the subject domain is not specified*;
- *no constants are used in the definition*;
- *the key sc-element of the semantic neighborhood is not specified*;
- *maximum studied object class is not specified for subject domain*;
- *relation domains are not specified*;
- *no unit or scale for the measured parameter is indicated*;
- *concept is not related to any subject domain*.

To identify incompleteness in the knowledge base, rules are used that are recorded within the framework of the corresponding ontologies in the knowledge base. The figure 21 shows an example of such a rule, according to which each relation must have a definition domain.

The listed classes of problem structures are specified in the *ontology of subject domain of the problem parts of the knowledge base*. The specified list of classes can be expanded and supplemented.

2) Means of specifying changes and transitions in the knowledge base: In the course of its evolution, the knowledge base undergoes significant changes, in particular, it is necessary to make changes that affect the conceptual structure of the subject domains described in the knowledge base. Among these types of changes the most problematic are the following:

- in the knowledge base **you need to override the already introduced and used concept**.
- in the knowledge base **an alternative concept appears, which excludes the use of another** concept associated with it.

To solve problems in the above situations, the following classes of concepts are introduced that are part of the *ontology of situations and events in sc-memory*:

- *main concept*;
- *non-main concept*;
- *concept moving from main to non-main*;
- *concept moving from non-main to main*.

The figure 22 shows an example of the specification of the transition process in the knowledge base with the indication of the planned completion dates of this process and the rule on the basis of which the transition is made.

As mentioned earlier, the *history of computer system development* section is used to store the history of changes in the knowledge base in the process of its evolution. Figure 23 shows an example of a structure that uses means for specifying changes made in the knowledge base.

It should be noted that in this case, actions are specified not only for editing the knowledge base, but also actions for coordinating the changes made to it. All performed actions, as well as their specifications, are included into the section *history of computer system development*.

This mechanism for changes fixation in the knowledge base is the basis for managing versions of the knowledge base. In the framework of the proposed approach, it is assumed that, if necessary, a rollback of the changes made before any action in the history is required to perform in reverse order a number of actions, which are inverse to the actions that follow the specified action in the history. At the same time, actions performed in this way are also added to the change history in the order of execution.

C. Problem solver and user interface of system for the collective knowledge bases development support

Problem solver of system for the collective knowledge bases development support is a team of knowledge processing agents, each of which automates actions belonging to any of the classes of actions of the knowledge bases developers discussed above.

The structure of the considered solver in SCn-code:

Problem solver of system for the collective knowledge bases development support
<= *abstract sc-agent decomposition**:
{
- *Non-atomic sc-agent of knowledge bases editing*
- *Non-atomic sc-agent for automation of activities of ordinary knowledge base developer*
- *Non-atomic sc-agent for automation of activities of knowledge base administrator*
- *Non-atomic sc-agent for automation of activities of knowledge base manager*
- *Non-atomic sc-agent for automation of activities of knowledge base expert*
- *Non-atomic sc-agent for knowledge base characteristics calculating*

}

Traditionally, when working collectively with a shared resource, in this case, a knowledge base, conflicts may arise, for example, several developers try to enter conflicting or duplicate information in the knowledge base, try to simultaneously change the same piece of knowledge base. The final decision is the responsibility of the knowledge base administrator.

User interface of system for the collective knowledge bases development support is presented by a set of interface commands that allow developers to initiate

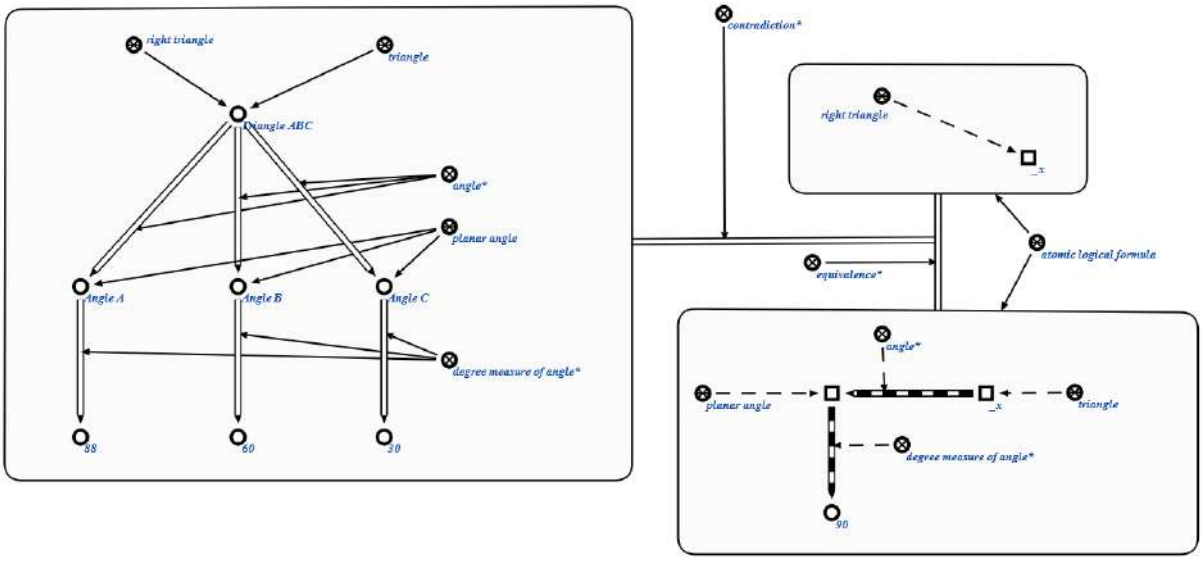

Figure 19. Example of the description of the contradiction with the definition in the knowledge base

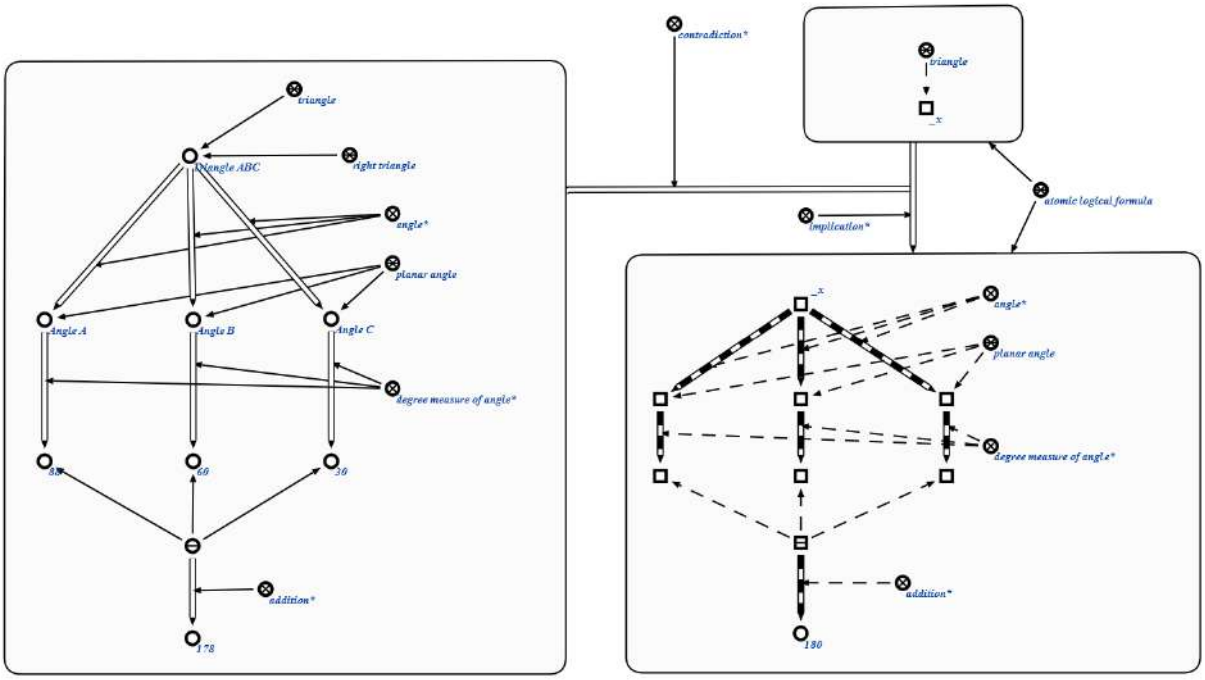

Figure 20. Example of the description of the contradiction with the theorem in the knowledge base

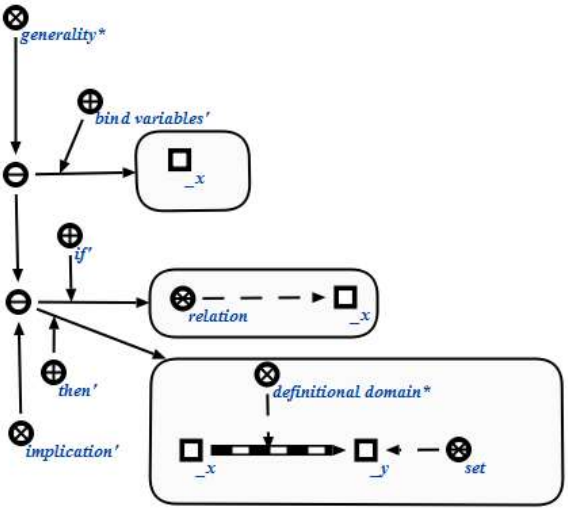

Figure 21. Example of incompleteness specification in the knowledge base

the activity of the desired agent that is part of this system [77], as well as a set of editors which allow editing knowledge base fragments taking into account the mechanism discussed earlier.

In the current version of the tools, there are two editors that support the ability to edit in the SCg (figure 24) and SCn (figure 25) languages.

D. Tools of automating the development of problem solvers sc-models

Among the tasks solved by *tools of automating the development of problem solvers sc-models* are technical support for problem solver developers, including ensuring the correct and efficient implementation of the steps provided by the above methods. These tools are also implemented as an osti-system, which can be used both in the local version and as a subsystem for the automation system for the development of knowledge bases.

In turn, within the framework of the system under consideration, two subsystems are conventionally distinguished: the subsystem of automation of the process of constructing and modifying of knowledge processing agents and the subsystem of automating the process of constructing and modifying of scp-programs.

Graphically, the structure of the system under consideration and its subsystems can be represented as follows (figure 26).

An important stage in the development of software systems is the debugging of the developed components. In the case of problem solvers based on OSTIS Technology, two fundamentally different levels of debugging are distinguished:

- debugging at *sc-agents* level;
- debugging at *scp-program* level.

In the case of debugging at the sc-agents level, the act of execution of each agent is considered indivisible and cannot be interrupted. In this case, both atomic sc-agents and non-atomic ones can be debugged. The initiation of one or another agent, including one that is not part of an atomic one, is done by creating appropriate constructions in sc-memory, thus, debugging can be done at different levels of detailing of agents, even atomic ones.

Taking into account the fact that the model of agents interaction used within the framework of OSTIS Technology uses a universal variant of interaction of agents through common memory, the considered agent design support system can serve as a basis for agent modeling systems that use other communication principles, for example, direct message exchange between agents.

Debugging at the level of scp-programs is carried out similarly to the existing modern approaches to debugging procedural programs and suggests the possibility of setting breakpoints, step-by-step program execution, etc.

The considered system for automating the constructing and modifying of problem solvers, accordingly, its sc-model, is divided into two more specific ones:

System for automating the constructing and modifying of problems solvers using OSTIS Technology
<= *basic decomposition**:
{
- *System for automating the constructing and modifying of knowledge processing agents*
- *System for automating the constructing and modifying of scp-programs*

}

In turn, these subsystems are decomposed in accordance with the general principles of ostis-systems building into sc-models of the knowledge base, problem solver and user interface. Next, we consider in more detail the components listed.

The knowledge base of the system for automating the constructing and modifying of problem solvers includes, in addition to the Knowledge base kernel and kernel extensions, sc-models of knowledge bases provided at the level of OSTIS Technology and models of subject domains of scp-programs and scp-interpreter as well the description of key concepts related to verification and debugging of scp-programs such as *breakpoint*, *incorrectness in the scp-program*, *error in the scp-program* and others.

The problem solver of the system for automating the constructing and modifying of knowledge processing agents has the following structure:

Problem solver of system for for automating the constructing and modifying of knowledge processing agents
<= *decomposition of sc-agent**:

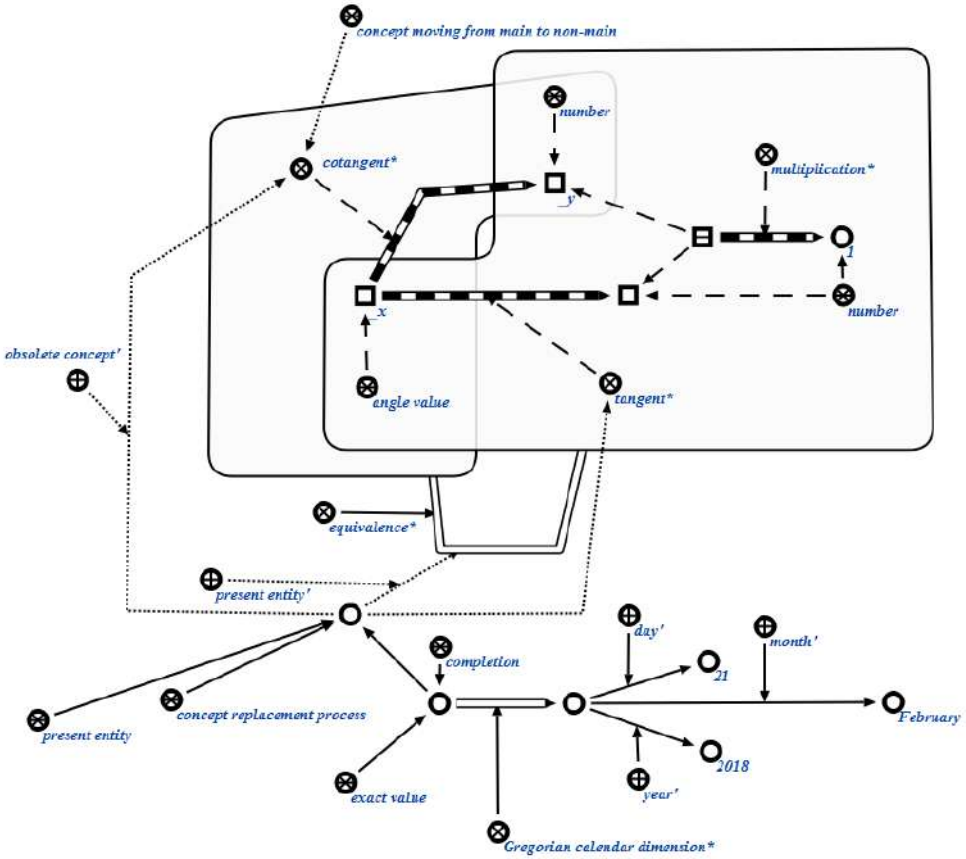

Figure 22. Transition process specification in knowledge base

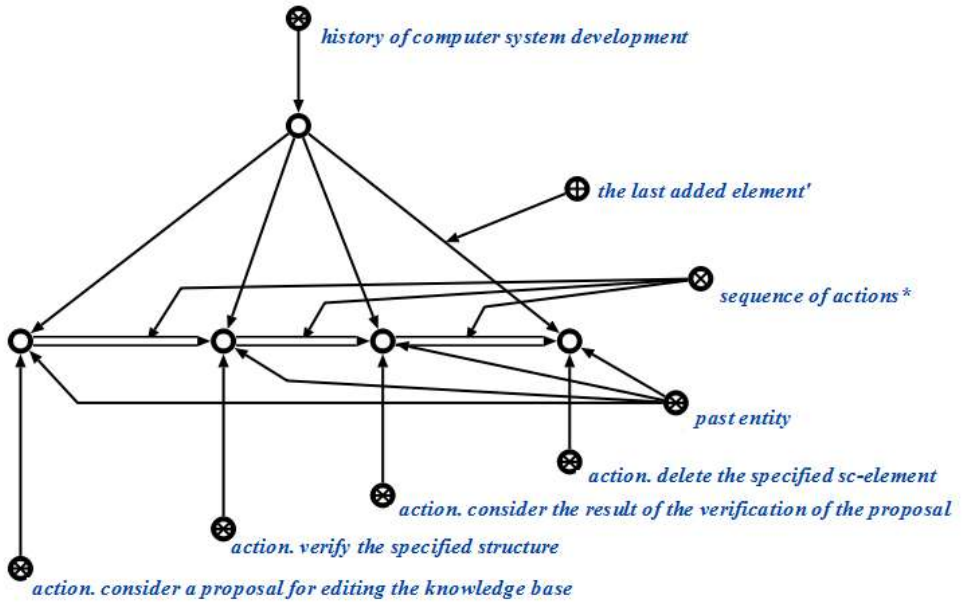

Figure 23. Means of specifying changes made in the knowledge base

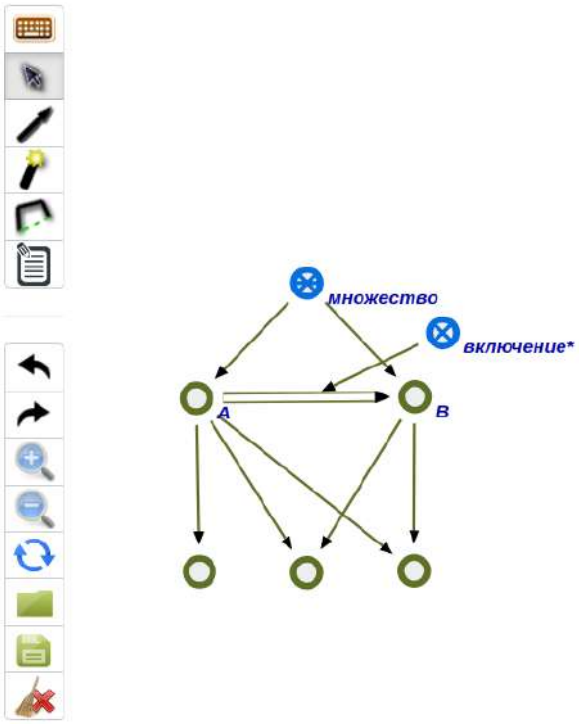

Figure 24. SCg-editor example

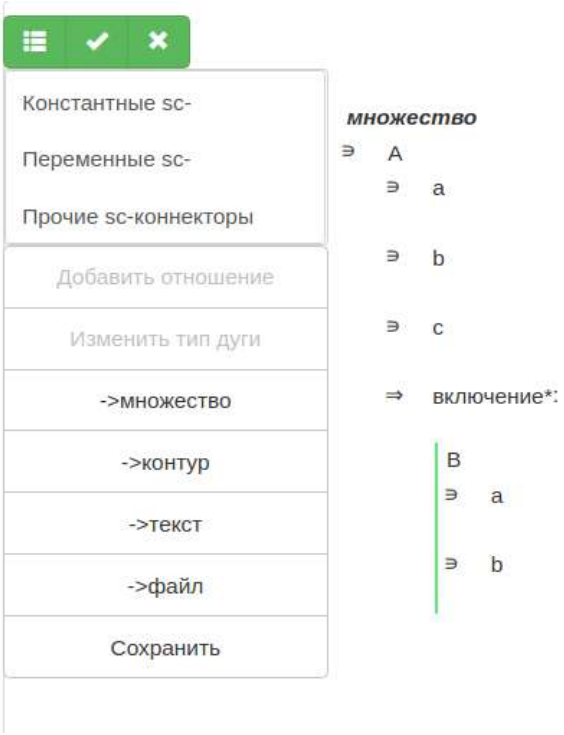

Figure 25. SCn-editor example

{
- *Abstract sc-agent of sc-agents verification*
  <= decomposition of sc-agent:*
  {
  - *Abstract sc-agent of sc-agents specification verification*
  - *Abstract sc-agent for checking a nonatomic sc-agent for the consistency of its specification to the specifications of particular sc-agents in its composition*

  }
- *Abstract sc-agent of sc-agents teams debugging*
  <= decomposition of sc-agent:*
  {
  - *Abstract sc-agent of search for all running processes corresponding to a given sc-agent*
  - *Abstract sc-agent of initiation of a given sc-agent on the given arguments*
  - *Abstract sc-agent of activation of a given sc-agent*
  - *Abstract sc-agent of deactivation of a given sc-agent*
  - *Abstract sc-agent of setting a lock of a given type for a given process on a given sc-element*
  - *Abstract sc-agent of unlocking of all locks of a given process*
  - *Abstract sc-agent of unlocking of all locks of a given sc-element*

  }

}

In turn, the problem solver of the automation system for automating the constructing and modifying of scp-programs has the following structure:

Task solver of the automation system for automating the constructing and modifying of scp-programs
<= decomposition of sc-agent:*
{
- *Abstract sc-agent of scp-programs verification*
- *Abstract sc-agent of scp-programs debugging*
  <= decomposition of sc-agent:*
  {
  - *Abstract sc-agent of launch of a given scp-program for a given set of input data*
  - *Abstract sc-agent of launch of a given scp-program for a given set of input data in step-by-step mode*
  - *Abstract sc-agent of search of all scp-operators in the scp-program*
  - *Abstract sc-agent of search of all breakpoints within the scp-process*
  - *Abstract sc-agent of adding breakpoint in scp-program*
  - *Abstract sc-agent of removing breakpoint*

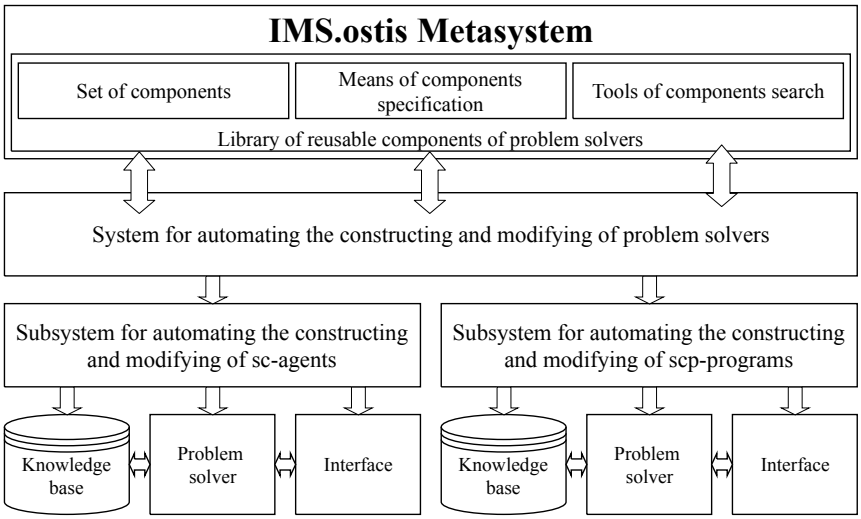

Figure 26. The structure of the system for automating the constructing and modifying of problem solvers

from scp-program

- *Abstract sc-agent of adding breakpoint in scp-process*
- *Abstract sc-agent of removing breakpoint from scp-process*
- *Abstract sc-agent to continue the execution of the scp-process on one step*
- *Abstract sc-agent to continue the execution of the scp-process to a breakpoint or end*
- *Abstract sc-agent for viewing information about the scp-process*
- *Abstract sc-agent for viewing information about the scp-operator*

}

}

Since the objects of the design of the described automation system are the components of the problem solvers, in particular, agents and knowledge processing programs presented in the SC-code, such a system can use the basic means of external representation of the texts of the SC-code, for example, SCn or SCg languages.

In order to visually simplify the process of verifying and debugging the components of the solver, an approach is used that assumes that only the minimum necessary set of sc-elements is displayed to the system user at a time. For example, when debugging a scp-process, it suffices to display the scp-operators and connections between them. If necessary, the user can manually request and view the specification of the desired scp-operator at the time of the break. This approach is embedded in the algorithms of all agents of the described system.

Thus, at present, the user interface of the system for automating the constructing and modifying of problem solvers is represented by a set of interface commands that allow the user to initiate the activity of the necessary agent that is part of this system.

X. Means of project task specification

To represent the project tasks and their specifications in the knowledge base of the system for automating the development of knowledge bases, the first version of the sc-model of the *Subject domain of project actions* and its ontology were developed. This subject domains is particular for the *Subject domain of actions and tasks* [73] and, thus, inherits from it many general concepts, such as *action*, *action class*, *task* , *decomposition of action**, *subaction**, *performer** and others.

In the framework of the *Subject domain of project actions*, concepts are studied that are directly related to project activities, in particular, with classes of project activities, priorities of project activities and dependencies between them. The corresponding ontology can become the basis for the formalization of existing methods and standards in the field of organization of project and intelligent activity [78], [79] and was developed taking into account existing standards in this field.

It should be noted that in the framework of the proposed approach to the formalization of activities *task* is treated as a specification (semantic neighborhood within the framework of the knowledge base) of a certain *action*. Thus, each action can be assigned a certain task, containing the conditions in which the specified action is or should be performed. In this connection, it turned out to be inexpedient to introduce separately the classification of actions and the classification of tasks, and the choice was made in favor of the classification of actions, since the concept of action can be used in a wider context, which is not necessarily related to the project activity. Thus, in describing project activities, the concepts *action* and *action class* will be used, and it is

understood that, if necessary, a project action can always be put in correspondence with a project task.

Examples of the specification of concepts studied in the *Subject domain of project actions* in the SCn-code:

project action
⊂ *action*

dependent action*
∈ *binary relation*:
=> *first domain**:
project action
=> *second domain**:
project action

Tuples of ***dependent action**** relation connect together some project action and another project action, which cannot be completed until the original project action is successfully executed. It is assumed that the original action is not a subaction for the dependent action.

action priority*
∈ *binary relation*:
=> *first domain**:
project action
=> *second domain**:
project action

Tuples of ***action priority**** relation connect two project actions, the first of which is of higher priority for some reason. Most often it is assumed that both actions are the actions of some general action.

In turn, for the development of ostis-systems, additional classes of project activities were allocated, taking into account the specifics of developing sc-models of knowledge bases and sc-models of problem solvers. The specified classes of actions are studied in the framework of the *Subject domain of actions of knowledge bases sc-models developers*.

Fragment of the typology of project actions of the ostis-systems sc-models developers in the SCn-code:

action. build a new fragment for inclusion in the knowledge base
⊃ *action. build subject domain sc-model*
=> *abstract subaction**:
- *action. build a structural specification of subject domain*
- *action. build a terminological ontology of subject domain*
- *action. build a set-theoretic ontology of subject domain*
- *action. build a logical ontology of subject domain*

⊃ *action. build a semantic neighborhood of a given entity*
⊃ *action. develop an example of the given concept use*

An example of the specification of project activities for the development of the IMS.ostis Metasystem knowledge base:

Section. Development plan of the IMS.ostis Metasystem
∋ *key sc-element'*:
- *Action. develop exercises for the formalization of basic knowledge*
- *Action. develop a family of introductory sections on OSTIS Technology*
- *Action. build sc-model of the subject domain of artificial neural networks*
  => *subaction**
  - *Action. build a structural specification of the subject domain of artificial neural networks*
  - *Action. build a terminological ontology of the subject domain of artificial neural networks*
  - *Action. build a set-theoretic ontology of the subject domain of artificial neural networks*
  - *Action. build a logical ontology of the subject domain of artificial neural networks*

XI. Means of specification of participants in the development of semantic computer systems

One of the important advantages of the approach to the organization of the development process, in which all the information about executing, executed and planned actions, participants in the process, etc., is recorded in the knowledge base, is the possibility of creating professional "portraits" of developers which can be further analyzed and taken into account when solving, for example, such tasks as:

- evaluation of the developer's total contribution to the development results for a certain period, including in material terms;
- evaluation of the experience and competence of the developer in solving tasks of certain classes for
  - •• planning available resources and optimizing the assignment of tasks in terms of their implementation;
  - •• determining the need to train certain developers in any areas, the improving of certain skills of a specific developer;
  - •• assignment of authority and determining the value of a particular developer's opinion when making any collective decisions;
- the ability to automate the above processes.

An example of a fragment of a knowledge base in the SCg-code describing a specific participant in the development process (figure 27):

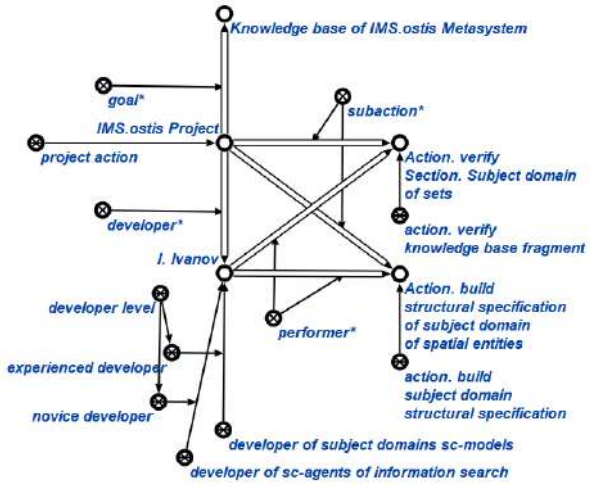

Figure 27. An example of the description of a developer professional portrait

XII. Mechanisms for assessing the complexity of project tasks and the contribution of the developers of semantic computer systems and their components

Evaluation of the developer's experience and the construction of his professional portrait are closely related to the mechanisms for assessing the contribution of each particular developer to the system being developed. At the same time, the assessment of the contribution, expressed in any conventional quantitative units, will allow both to evaluate the developer's experience and directly provide his material remuneration and solve a number of other tasks related to the assessment of the participation of developers in the development of a specific system. Thus, it is important to move from continuous improvement of methods and tools of assessing the complexity of project tasks in developing such systems, as well as assessing the quality, timeliness and value of project developer's results to improving methods and tools of project activities stimulating.

In the development of modern computer systems, the assessment of the complexity of the tasks being solved, as a rule, manually on the basis of accumulated experience and is expressed in man-hours. Real remuneration is formed depending on the qualifications of the developer and the total number of man-hours corresponding to the tasks solved for a certain period of time.

When assessing the contribution of the developers of osti-systems, it is possible to take into account the semantics of the fragments being developed, which, on the one hand, will allow **automate the process of assessing contribution** for each developer, and on the other, allow **to make such an assessment more objective**.

In addition, the fact that all project activities are described in the knowledge base of the designed system and, accordingly, can be analyzed by the system itself, provides additional opportunities for automating the process of evaluating the contribution of the developer.

The assessment of the contribution in the general case may depend on the following factors:

- directly the amount of the changes made (the number of concepts, the number of sc-arcs, the number of operators in programs, etc.) and the amount of work performed (in man-hours or other conventional units of labor intensity);
- the complexity of the changes (in general, a fragment of the knowledge base describing a formal logical statement is considered more complicated than, for example, the description of a simple set-theoretic connection between concepts);
- quality of the changes made, which is assessed not only in terms of the correctness of the changes, but also their completeness, compatibility with other fragments of the system, etc. As was shown earlier, the assessment of the quality of fragments of knowledge bases of ostis-systems (including the specifications of knowledge-processing agents and their corresponding programs) can be largely automated;
- importance (purposefulness, expediency, priority) of the task (accomplishment of the task with a priority higher in terms of achieving current goals, is rated higher than solving a useful, but not very priority task).

One of the problems that arise in assessing the contribution of the developer, and in assessing his professional experience, is the problem that the author is sometimes from the point of view of the system (the person who directly formed the proposal to edit the knowledge base) and the real author of knowledge introduced into the system (an expert who does not use the technical tools of the knowledge base editing) may turn out to be different people, and the formal authorship will be attributed not to the expert, but to his technical assistant. To solve this problem, in addition to the obvious option, which involves simplifying technical tools and adapting them to subject domain experts, an option is proposed where the authorship of the expert is clearly specified manually and considered in the same way as any other proposal for editing the knowledge base.

XIII. Conclusion

The paper discusses the principles of developing new generation semantic computer systems based on the Open Semantic Technology for Intelligent System Design (OSTIS Technology), justifies the advantages of transition from traditional computer systems to semantic computer systems from the point of view of their design process, and considers the advantages of developing design automation tools as semantic computer systems.

The main conclusions on the work include the following:

- the accumulated experience of organizing and automating the development of modern computer systems (synthesis, assembly, analysis, testing, diagnostics, etc.) is a rich basis for creating models, methods and means of organizing and computer support for project activities that should be directed:
  - •• on the consistency of project actions of all developers (compatibility of project results);
  - •• on the reduction of the time of transition from the current workable version to the next also consistent version not due to the intensification of the developers' activities, but due to:
    - ••• increasing the consistency of their actions;
    - ••• increase the level of **valuation**, the purposefulness of the results of each developer from the point of view of the earliest construction of the next consistent workable version of the developed system, which has qualitative advantages over the previous version;
- the features of the development objects themselves (semantic computer systems) due to the possibility of their consideration at the semantic level create favorable prerequisites for effective analysis:
  - •• consistency of project activities;
  - •• quality of project results (consistency, completeness, clearness);
  - •• valuation (purposefulness) of project results;
  - •• scope of completed design work;

  This, in turn, creates prerequisites for creating methods and tools of effectively stimulating project activities, which can also be used in open source projects that assume free entry into the development team;
- development of semantic computer systems in the presence of a satisfactory version of the implementation of a universal interpreter of semantic models is reduced to the development of the relevant sections of knowledge base. Therefore, the intelligent system to support the collective design of knowledge bases of semantic computer systems has a special place in the complex of design tools for semantic computer systems.

ACKNOWLEDGMENT

This work was supported by the BRFFR-RFFR (No. F18R-220).

REFERENCES

[1] V. V. Golenkov and N. A. Guljakina, "Proekt otkrytoj semanticheskoj tehnologii komponentnogo proektirovanija intellektual'nyh sistem. chast' 1: Principy sozdanija [project of open semantic technology for component design of intelligent systems. part 1: Creation principles]," *Ontologija proektirovanija [Ontology of design]*, no. 1, pp. 42–64, 2014.

[2] V. Golenkov, N. Guliakina, N. Grakova, I. Davydenko, V. Nikulenka, A. Eremeev, and V. Tarassov, "From training intelligent systems to training their development tools," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2018, pp. 81–98.

[3] A. Kolesnikov, *Gibridnye intellektual'nye sistemy: Teoriya i tekhnologiya razrabotki [Hybrid intelligent systems: theory and technology of development]*, A. M. Yashin, Ed. SPb.: Izd-vo SPbGTU, 2001.

[4] I. Davydenko, "Semantic models, method and tools of knowledge bases coordinated development based on reusable components," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2018, pp. 99–118.

[5] D. Shunkevich, "Agent-oriented models, method and tools of compatible problem solvers development for intelligent systems," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2018, pp. 119–132.

[6] (2018, Feb.) Autodesk generative design. [Online]. Available: https://www.autodesk.com/solutions/generative-design

[7] (2018) Autodesk dreamcatcher. [Online]. Available: https://autodeskresearch.com/projects/dreamcatcher

[8] S. M. Duffy and A. H. B. Duffy, "Sharing the learning activity using intelligent cad," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 10, no. 2, p. 83–100, 1996.

[9] L. S. Cardona-Meza and G. Olivar-Tost, "Modeling and simulation of project management through the PMBOK standard using complex networks," *Complexity*, vol. 2017, pp. 1–12, 2017.

[10] J. Stark, *Product Lifecycle Management (Volume 1): 21st Century Paradigm for Product Realisation*, 3rd ed. Switzerland: Springer, Cham, 2015.

[11] T. Dillon, E. Chang, and P. Wongthongtham, "Ontology-based software engineering- software engineering 2.0." *Australian Software Engineering Conference, IEEE Computer Society*, pp. 13–23, 2008.

[12] A. Emdad, "Use of ontologies in software engineering," 2008, pp. 145–150.

[13] A. N. Andrichenko, "Tendencii i sostojanie v oblasti upravlenija spravochnymi dannymi v mashinostroenii [trends and status in the field of reference data management in mechanical engineering]," *Ontologija proektirovanija [Ontology of design]*, no. 2(4), pp. 25–35, 2012.

[14] A. Fedotova, I. Davydenko, and A. Pfortner, "Design intelligent lifecycle management systems based on applying of semantic technologies," vol. 1. Switzerland , Springer International Publishing,, 2016, pp. 251–260.

[15] (2018) Agile manifest. [Online]. Available: http://agilemanifesto.org/iso/ru/manifesto.html

[16] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," *Agile Software Development*, vol. 1, pp. 31–59, 2002.

[17] (2018) Best version control systems. [Online]. Available: https://www.g2crowd.com/categories/version-control-systems

[18] (2018) Comparison of issue-tracking systems. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems

[19] (2018) Project management software. [Online]. Available: https://www.capterra.com/project-management-software

[20] (2018) Comparison of gui testing tools. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_GUI_testing_tools

[21] (2018) List of web testing tools. [Online]. Available: https://en.wikipedia.org/wiki/List_of_web_testing_tools

[22] (2018) Best version control hosting software. [Online]. Available: https://www.g2crowd.com/categories/version-control-hosting

[23] R. Iqbal, A. Murad, and A. Mustapha, "An analysis of ontology engineering methodologies : a lit. rev." *Research J. of Appl. Sciences, Engineering a. Technology*, vol. 6, no. 16, pp. 48–62, 2013.

[24] P. Sainter, K. Oldham, and A. Larkin, "Achieving benefits from knowledge-based engineering systems in the longer term as well as in the short term," *Semantic Scholar*, retrieved from https://pdfs.semanticscholar.org/0edc/b90ca6601192e10d7eb2fd03147fb72e1f9d.pdf.

[25] A. A. Slobodjuk, S. I. Matorin, and S. N. Chetverikov, "O podhode k sozdaniju ontologij na osnove sistemnoobektnyh modelej predmetnoj oblasti [on the approach to the creation of ontologies based on system-object domain models]," *Nauch. vedomosti Belgor. gos. un-ta. Ser.: Jekonomika. Informatika [Scientific journ. of Bel. state univ. Ser .: Economy. Computer science]*, no. 22, pp. 186–193, 2013.

[26] M. Uschold and M. Grueninger, "Ontologies: principles, methods and applications," *The Knowledge Engineering Rev*, vol. 11, no. 2, pp. 93–136, 1996.

[27] M. Gruninger and M. Fox, "Methodology for the design and evaluation of ontologies [Electronic resource]," *Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95*, mode of access: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=70152DBAFA34B86E22661C3E3F7C978F?doi=10.1.1.44.8723\&rep=rep1\&type=pdf. — Date of access: 30.10.2016.

[28] A. Gomez-Perez, M. Fernandez-Lopez, and A. de Vicente, "Towards a method to conceptualize domain ontologies," in *ECAI 96 : proc. of the 12th Europ. Conf. on Artificial Intelligence, Budapest, 11–16 Aug. 1996*, Europ. Coordinating Comm. for Artificial Intelligence ; ed. W. Wahlster. Budapest, 1996, pp. 41–51.

[29] A. Gomez-Perez and M. Suarez-Figueroa, "Scenarios for building ontology networks within the neon methodology," in *Proceedings of the Fifth International Conference on Knowledge Capture (K-CAP 2009)*, Assoc. for Computing Machinery, Spec. Interest Group on Artificial Intelligence. New York, 2009, pp. 183–184.

[30] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke, "Ontoedit: collaborative ontology development for the semantic web," in *The Semantic Web (ISWC 2002) : proc. of the first Intern. semantic web conf., Sardinia, 9–12 June 2002*, I. Horrocks and J. Hendler, Eds. Berlin, 2002, pp. 221–235.

[31] A. Bernaras, I. Laresgoiti, and J. Corera, "Building and reusing ontologies for electrical network applications," in *ECAI 96 : proc. of 12th Europ. Conf. on Artificial Intelligence, Budapest, 11–16 Aug. 1996*, Europ. Coordinating Comm. for Artificial Intelligence. Chichester, 1996, pp. 298–302.

[32] H. S. Pinto, S. Staab, and C. Tempich, "Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies," in *European Conference on Artificial Intelligence (ECAI'04) : proc. of the 16th Conference, Valencia, Spain – August 22–27, 2004*. Amsterdam, 2004, pp. 393–397.

[33] B. Swartout, R. Patil, K. Knight, and T. Russ, "Toward distributed use of large-scale ontologies [Electronic resource]," *Semantic Scholar*, mode of access: https://www.semanticscholar.org/paper/Toward-Distributed-Use-of-Large-Scale-Ontologies-Swartout-Patil/07b64a07e6d56a91f2471975a3922e3fcd9ff2d7?tab=abstract. — Date of access: 26.08.2016.

[34] A. D. Nicola, M. Missikoff, and R. Navigli, "A proposal for a unified process for ontology building: Upon," in *Database and expert systems applications : proc. of the 16th intern. conf. (DEXA 2005), Copenhagen, 22–26 Aug. 2005*, K. Andersen, J. Debenham, and R. Wagner, Eds. New York ; Berlin , Springer, 1997, pp. 655–664.

[35] J. V. Rogushina, "Semanticheskie wiki-resursy i ih ispol'zovanie dlja postroenija personificirovannyh ontologij [semantic wiki-resources and their use for building personalized ontologies]," in *Proceedings of the 10th International conference of programming (UkrPROG'2016), Kyiv, 24–25 May 2016*, P. A. I. Sergienko, Ed. Kyiv, 2016, pp. 188–195.

[36] M. Raman, "Wiki technology as a «free» collaborative tool within an organizational setting," *Inform. Systems Management*, vol. 23, no. 4, pp. 59–66, 2006.

[37] "MediaWiki [Electronic resource]," mode of access: https://www.mediawiki.org.. — Date of access: 26.11.2016.

[38] A. J. Gladun and J. V. Rogushina, "Wiki-tehnologii," *Telekom. Kommunikacii i seti*, no. 5, p. 58, 2008.

[39] Semantic MediaWiki [Electronic resource]. Mode of access: https://www.semantic-mediawiki.org. — Date of access: 21.10.2016.

[40] I. S. Chistjakova, "Inzhenerija ontologij [ontology engineering]," *Inzhenerija programm. zabezpechennja [Software engineering]*, no. 20, pp. 51–66, 2014.

[41] "Protege [Electronic resource]," mode of access: http://protege.stanford.edu. — Date of access: 27.05.2016.

[42] J. Euzenat, "Corporate memory through cooperative creation of knowledge bases and hyper-documents [Electronic resource]," *Knowledge Acquisition for Knowledge-Based Systems : Proc. of 10th Workshop, EKAW'97*, mode of access: http://ksi.cpsc.ucalgary.ca/KAW/KAW96/euzenat/euzenat96b.html. — Date of access: 30.10.2016.

[43] "Ontolingua [Electronic resource]," *Knowledge systems, al laboratory*, mode of access: http://www.ksl.stanford.edu/software/ontolingua. — Date of access: 29.05.2016.

[44] "Building DAML+OIL ontologies [Electronic resource]," *OilEd*, mode of access: http://oiled.semanticweb.org/building. — Date of access: 24.05.2016.

[45] "Projects [Electronic resource]," *Knowledge Media Institute*, mode of access: http://kmi.open.ac.uk/projects. — Date of access: 22.05.2016.

[46] N. M. Borgest, "Rol' ontologii v proektirovanii informacionnyh sistem [role of ontology in information system design]," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2014, pp. 155–160.

[47] O. M. Ovdej and G. J. Proskudina, "Obzor instrumentov inzhenerii ontologij [ontology engineering tools survey]," *Jelektron. b-ki. [Electronic libraries]*, vol. 7, no. 4, 2004, available at: http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2004/part4/op. — accessed: 30.11.2016.

[48] E. Alatrish, "Comparison some of ontology," *Management Inform. Systems*, vol. 8, no. 2, pp. 18–24, 2013.

[49] N. Noy, M. Klein, S. Kunnatur, A. Chugh, and S. Falconer, "PROMPT [Electronic resource]," *Protege Wiki*, mode of access: https://protegewiki.stanford.edu/wiki/PROMPT. — Date of access: 23.11.2016.

[50] "Chimaera [Electronic resource]," *Knowledge systems, al laboratory*, mode of access: http://www.ksl.stanford.edu/software/chimaera/. — Date of access: 20.05.2016.

[51] "OntoMerge: ontology translation by merging ontologies [Electronic resource]," *Computer Science*, mode of access: http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html. — Date of access: 19.10.2016.

[52] H. Chalupsky, "OntoMorph: a translation system for symbolic knowledge [Electronic resource]," mode of access: https://www.isi.edu/~hans/ontomorph/presentation/ontomorph.html. — Date of access: 19.05.2016.

[53] "Ontology based system enhanced with relationships for vocabulary heterogeneity resolution [Electronic resource]," *SID Research Group*, mode of access: http://sid.cps.unizar.es/OBSERVER. — Date of access: 16.05.2016.

[54] G. Stumme and A. Maedche, "Fca-merge: bottom-up merging of ontologies," in *IJCAI-01 : proc. of the seventeenth Intern. joint conf. on artificial intelligence, Seattle, 4–10 Aug. 2001*, B. Nebel, Ed., Amer. Assoc. for Artificial Intelligence. San Francisco, 2001, pp. 225–230.

[55] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna, "MnM: ontology driven semi-automatic and automatic support for semantic markup," in *Knowledge engineering and knowledge management: ontologies and the semantic web : proc. of the 13th intern. conf., Siguenza, 1–4 Oct. 2002*, A. Gomez-Perez and V. R. Benjaminsns, Eds. Berlin ; New York, 2002, pp. 379–391.

[56] J. Heflin and J. Hendler, "A portrait of the semantic web in action," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 54–59, 04 2001.

[57] I. V. Efimenko and V. F. Horoshevskij, *Ontologicheskoe modelirovanie jekonomiki predprijatij i otraslej sovremennoj Rossii : v 2 t.* M. : Nac. issled. un-t «Vyssh. shk. jekonomiki», 2011, vol. 1 : Ontologicheskoe modelirovanie: podkhody, modeli, metody, sredstva, resheniya [Ontological modeling: approaches, models, methods, tools, solutions], (Preprint / Nac. issled. un-t «Vyssh. shk. jekonomiki» ; Serija: Matematicheskie metody analiza reshenij v jekonomike, biznese i politike ; WP7/2011/08).

[58] A. F. Tuzovskij, I. A. Zaikin, and V. Z. Jampol'skij, "Sistema kollektivnoj podderzhki ontologicheskih modelej [system of collective support of ontological models]," *Izv. Tom. politehn. un-ta [News of Tomsk polytechnic univ.]*, no. 5, pp. 89–94, 2011.

[59] T. Tudorache, N. Noy, S. Tu, and M. Musen, "Supporting collaborative ontology development in protege," in *The semantic web (ISWC 2008) : proc. of the 7th Intern. semantic web conf., Karlsruhe, 26–30 Oct. 2008*, D. H. [et al.], Ed. Karlsruhe, 2008, pp. 17–32.

[60] V. V. Gribova, A. S. Kleshchev, F. M. Moskalenko, V. A. Timchenko, L. Fedorishchev, and E. A. Shalfeeva, "Platforma dlya razrabotki oblachnykh intellektual'nykh servisov [platform for intelligent cloud services development]," in *XV Natsional'naya konferentsiya po iskusstvennomu intellektu s mezhdunarodnym uchastiem (KII-2016), Smolensk, 3–7 oktyabrya 2016 g. : trudy : v 3 t. [XV National Conference on Artificial Intelligence with International Participation (CAI-2016), Smolensk, October 3–7, 2016: procs: 3 v.]*, vol. 1, Ros. assots. iskusstv. intellekta, Feder. issled. tsentr «Informatika i upravlenie» Ros. akad. nauk [Russ. assoc. art. intelligence, Feder. research Center «Informatics and Management» Rus. Acad. of science]. Smolensk, 2016, pp. 24–33.

[61] A. N. Borisov, "Postroenie intellektual'nyh sistem, osnovannyh na znanijah, s povtornym ispol'zovaniem komponentov [building of intelligent knowledge-based systems with reusable components]," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk : BSUIR, 2014, pp. 97–102.

[62] L. S. Globa and R. L. Novogrudskaja, "Modeli i metody integracii informacionnyh i vychislitel'nyh resursov [models and methods for the integration of information and computing resources]," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2012, pp. 447–452.

[63] G. B. Zagorul'ko and J. A. Zagorul'ko, "Podhod k organizacii kompleksnoj podderzhki processa razrabotki intellektual'nyh sppr v slaboformalizovannyh predmetnyh oblastjah [approach to the organization of complex support of intelligent dms development in weakly-structured subject domains]," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2016, pp. 61–64.

[64] L. S. Bolotova, *Sistemy iskusstvennogo intellekta: modeli i tehnologii, osnovannye na znanijah : uchebnik [Artificial Intelligence Systems: Knowledge-Based Models and Technologies: Textbook]*. Moskow , Finances and Statistics, 2012.

[65] (2016, Aug.) RDF 1.1 semantics. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-mt-20140225

[66] (2016, May) Ontaria: easy access to the semantic web. [Online]. Available: https://www.w3.org/2004/ontaria/

[67] (2016, Aug.) Schema.org. [Online]. Available: http://schema.org

[68] M. D'Aquina and N. F. Noyb, "Where to publish and find ontologies? A survey of ontology libraries [Electronic resource]," *Web Semantics*, vol. 11, 2012, mode of access: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3293483/pdf/nihms-324445.pdf. — Date of access: 20.05.2016.

[69] C. Debruyne, P. D. Leenheer, and R. Meersman, "A method and tool for fact type reuse in the dogma ontology framework," in *On the move to meaningful Internet systems: OTM 2009 : in 2 pt.*, ser. 2 pt. London ; Berlin , Springer, 2009, pp. 1147–1164.

[70] P. D. Leenheer and C. Debruyne, "Dogma-mess: A tool for fact-oriented collaborative ontology evolution," in *On the move to meaningful Internet systems: OTM 2008 Workshops*, R. Meersman, Z. Tari, and P. Herrero, Eds. Heidelberg , Springer, 2009, pp. 797–806.

[71] M. Bergman, "Knowledge-based artificial intelligence [Electronic resource]," *AI3: Adaptive Information*, mode of access: http://www.mkbergman.com/1816/knowledge-based-artificial-intelligence. — Date of access: 14.10.2016.

[72] A. Filippov, V. Moshkin, D. Shalaev, and N. Yarushkina, "Edinaya ontologicheskaya platforma intellektual'nogo analiza dannykh [unified ontological data mining platform]," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2016, pp. 77–82.

[73] (2018, Nov.) IMS metasystem. [Online]. Available: http://ims.ostis.net/

[74] V. Golenkov, "Ontology-based design of intelligent systems," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2017, pp. 37–56.

[75] V. P. Ivashenko, "Modeli i algoritmy integratsii znanii na osnove odnorodnykh semanticheskikh setei [models and algorithms for the knowledge integration based on homogeneous semantic networks]," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, vol. 2, pp. 111–132, 2015.

[76] (2018, Nov.) Github. [Online]. Available: https://github.com

[77] D. Koronchik, "Semanticheskie modeli mul'timodal'nykh pol'zovatel'skikh interfeisov i semanticheskaya tekhnologiya ikh proektirovaniya [semantic models of multimodal user interfaces and semantic technology for their design]," in *Otkrytye semanticheskie tehnologii proektirovanija intellektual'nyh sistem [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed., BSUIR. Minsk , BSUIR, 2012, pp. 339–346.

[78] (2018) Pmbok® guide and standards. [Online]. Available: https://www.pmi.org/pmbok-guide-standards

[79] A. S. Kleshhjov and E. A. Shalfeeva, "Ontologija zadach intellektual'noj dejatel'nosti [ontology of the tasks of intelligent activity]," *Ontologija proektirovanija [Ontology of design]*, vol. 5, no. 2(16), pp. 179–205, 2015.

ПРИНЦИПЫ ОРГАНИЗАЦИИ И АВТОМАТИЗАЦИИ ПРОЦЕССА РАЗРАБОТКИ СЕМАНТИЧЕСКИХ КОМПЬЮТЕРНЫХ СИСТЕМ

Голенков В.В., Шункевич Д.В.,
Давыденко И.Т., Гракова Н.В.

Работа посвящена принципам разработки семантических компьютерных систем нового поколения на основе Открытой семантической технологии проектирования интеллектуальных систем (Технологии OSTIS). Обоснованы преимущества перехода от традиционных компьютерных систем к семантическим компьютерным системам с точки зрения процесса их проектирования, а также рассмотрены преимущества реализации средств автоматизации проектной деятельности как семантических компьютерных систем.

Received 09.01.19