Agent-oriented models, method and tools of compatible problem solvers development for intelligent systems

Shunkevich D. V.
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
shunkevichdv@gmail.com

***Abstract*—The article is devoted to the development of agent-oriented models, a method and means for developing compatible solvers of problems of intelligent systems capable to solve complex problems. The requirements for such solvers, the model of problem solver that satisfy the requirements, as well as the method and tools for developing and modifying such solvers are considered.**

The main problem considered in the work is the problem of low consistency of the principles underlying the implementation of various problem solving models. As a consequence, it is difficult to simultaneously use different models for solving problems within the same system when solving the same complex problem, it is practically impossible to reuse the technical solutions implemented in a certain system, in fact, there are no integrated methods and tools for problem solvers development.

As a basis for problem solvers design, it is proposed to use the multi-agent approach. The process of any problem solving is proposed to decompose into logical atomic actions, which will ensure compatibility and modifiability of the solvers. The solver is proposed to be considered as a hierarchical system consisting of several interconnected levels, which allows to provide the possibility of independent designing, debugging and verification of components at different levels.

***Keywords*—semantic technologies, ostis-system, problem solver, multi-agent system, intelligent agent, knowledge base**

I. Introduction

One of the key components of each intelligent system is the problem solver, which provides the ability to solve various problems, related both to the basic functionality of the system, and to the such a system efficiency ensuring, as well as the development automation of the system itself. The problem solver, ensuring the fulfillment of all listed functions, will be called the **integrated problem solver**.

The capabilities of the problem solver largely determine the functionality of the intelligent system as a whole, the ability to answer on non-trivial user questions and solve various problems in a certain subject domain.

The composition of the solver of each particular system depends on its purpose, the classes of problems being solved, the subject domain and a lot of other factors. Expanding of the scope of intelligent systems applications requires such systems to be able to solve **complex problems**, that is, problems that require the application of a variety of different knowledge representation models and different knowledge processing models.

Examples of such tasks are:

- The problem of the natural language texts understanding, both printed and hand-written, understanding of speech messages, semantic analysis of images. In each of the listed cases, it is necessary first to performs the syntactic analysis of the processed file, remove the insignificant fragments, then classify the significant fragments, correlate them with the concepts known to the system, identify those fragments, that the system can not recognize, eliminate duplication of information, etc.;
- The problem of automating adaptive learning of students, suggesting that the system itself can solve various problems from a certain subject domain, and also manage the learning process, create tasks for students and monitor their implementation;
- The problem of intelligent robots behavior planning, including both understanding of various kinds of external information, and the various decision making, using both reliable and plausible methods.
- The problem of complex and rapidly evolving automation of various enterprises;
- and others.

At present, there are a lot of problem solving models of various kinds, including the variety of types of logics (clear, fuzzy, inductive, deductive, temporal, etc.), neural networks and genetic algorithms, various strategies of problem solution search ways (depth-first search, breadth-first search, etc.), various programming languages, both declarative and imperative.

The ability of different models using for problems solving within a single system will allow to decompose the complex problem into subproblems, each of which can be solved by one of the methods, known to the system. Thanks to a combination of different problem solving models, the number of problem classes, which such a system will be able to solve, will be significantly larger than the total number of problem classes, which can be solved by several systems, each of which implements only one of the problem solving models being integrated.

Modern intelligent systems, oriented to the simultaneous use of different types of knowledge and various problem solving models, are built on the principle of hybrid computer

systems [1], [2]. Such an approach allows to solve complex problems, however, during hybrid systems design process, it becomes necessary to ensure the interaction interface for various problem solving models. That substantially increases the overhead in the such systems construction. As a result, such systems, as a rule, have a complex monolithic architecture, the introduction of any changes in which requires considerable work. To solve this problem, it is necessary to ensure the compatibility of various problem solvers.

At the same time, the urgent problem is the intelligent system teaching to new knowledge and skills and adapting it to permanently changing requirements. It is necessary to attach new resources, including new approaches to solving of problems of various classes. At the same time, unlike most modern approaches to computer systems learning (machine learning) [3], where the class of tasks being solved is actually fixed (does not change during the learning process) and only the method of solving is optimized, in this case we are talking about the expansion of the number of classes of problems solved by the system, and in the general case - unlimited expansion.

An important way to reduce the complexity of the process of changing the functionality of intelligent systems is the accumulation of libraries of reusable components of solvers that will significantly reduce both the terms of development and modification of solvers, and the level of professional requirements for their developers. At the same time unification of various models of problem solving on a common formal basis will allow to form not only traditional libraries of standard subprograms, but also libraries of entire solvers that implement one or another problems solving model.

Let us consider in detail one of the examples of complex problems listed above, which is related to the enterprise automation.

Within the integrated system of the enterprise automation, the following automation levels can be conventionally distinguished:

- automation of the actual production process at all stages, from the receipt and evaluation of raw materials to packaging and goods delivery to the end user;
- automation of production process management, that is, automation of making changes in the production process, for example, changes in batch quantities, nomenclature or properties of the manufactured product, etc;
- automation of production process control, which involves the use of various methods of the current situation analysis, as well as mechanisms for identifying, classifying and eliminating of emergencies, up to complete elimination of the emergency situation without operator intervention.

Figure 1 describes a schematic diagram of the integrated system of the enterprise automation, showing the application of which problem solving models is actual in each of the subsystems of such a system.

In the figure 2 the conventional scheme of the contingencies handling subsystem is shown in more detail.

This example shows that design of such a complex automation system is impossible without ensuring the consistent use of different types of knowledge and problem solving models within the same system when solving the same complex problem. In addition, the problem of such a system support in a state corresponding to the current production technologies level, supplementing it with more advanced models and methods of problem solving becomes urgent. It is obvious that such a system reconfiguration should be carried out directly during the system operation, and not require a complete stop of the entire production or its individual parts at every time. Thus, it can be said that such a system should be learnable, that is being able to acquire not only new knowledge, but also new skills.

The foregoing allows us to formulate requirements for the problem solver of an intelligent system, which is able to solve complex problems:

- at each point in time the solver must ensure the solution of problems from the specified class for a specified time, and the result of the problem solution must satisfy certain known requirements. In other words, as in the case of modern computer systems, the correctness of the problem solving results at the system development stage should be verified by special methods, including such modern approaches as unit-testing, «black box» testing and others [4].
- the solver should be easily **modifiable**, that is, the complexity of making changes to an already developed solver should be minimal. Ways to increase the modifiability are the ensurance of the introduced changes localness, as well as the availability of ready-made reusable components that can be included into the solver if necessary. At the same time, changes must be made directly during the system operation, and the overhead of new components integrating or replacing existing ones should be minimal.
- in order for the intellectual system to be able to analyze and optimize the existing problem solver, to integrate new components into it (even by ssystem itself), to evaluate the importance of certain components and their applicability for particular problem solving, the specification of the solver should be defined with system understandable language, for example, with the same means as the processed knowledge. The ability of an intelligent system to analyze (verify, correct, optimize) its own components will be called **reflexivity**.
- an additional requirement to the integrated problem solver in relation to the solver in general is its **completeness** (integrity, complexity), that is, such a solver must provide all the functionality of the system, i.e. ensure the solution of all problems, both related to the direct designation of the system, and ensuring the effectiveness of the system operation.

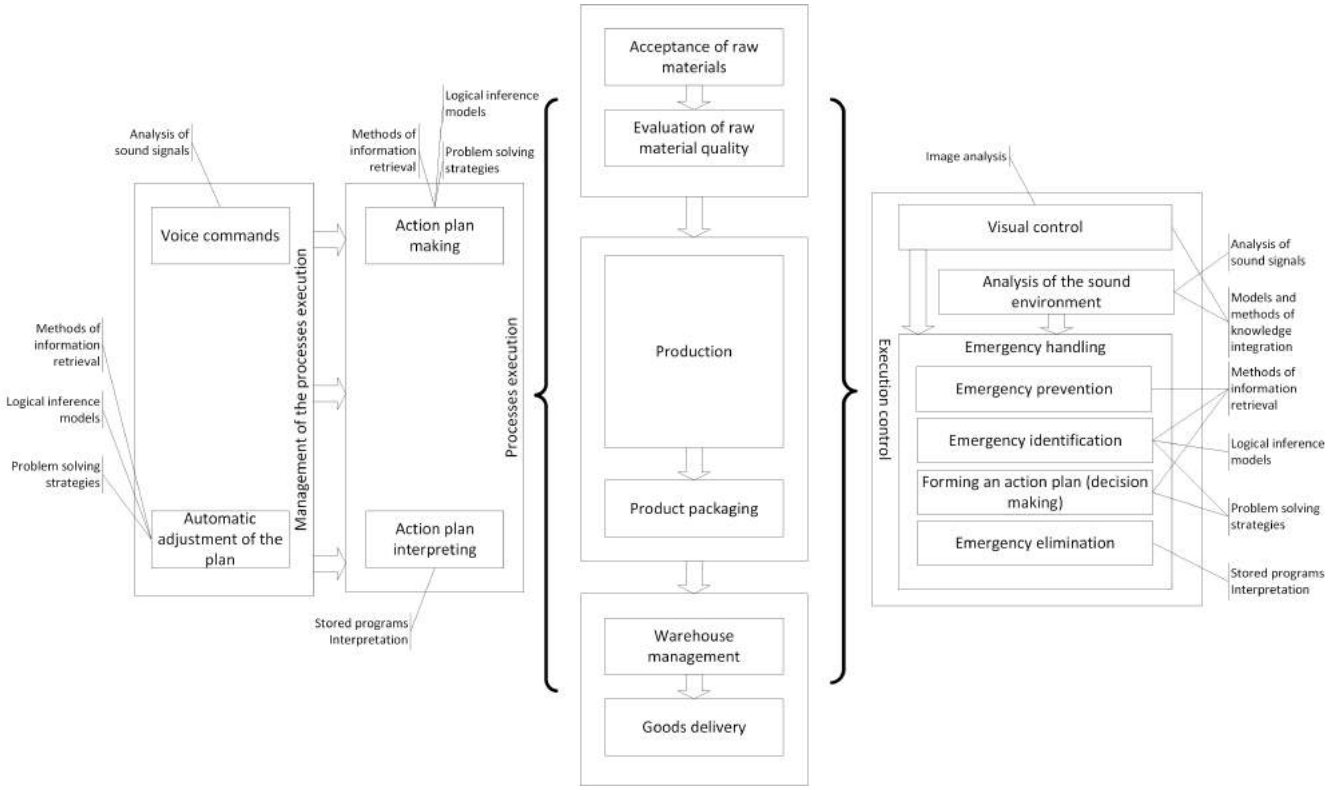

Figure 1. Simplified scheme of integrated system of the enterprise automation

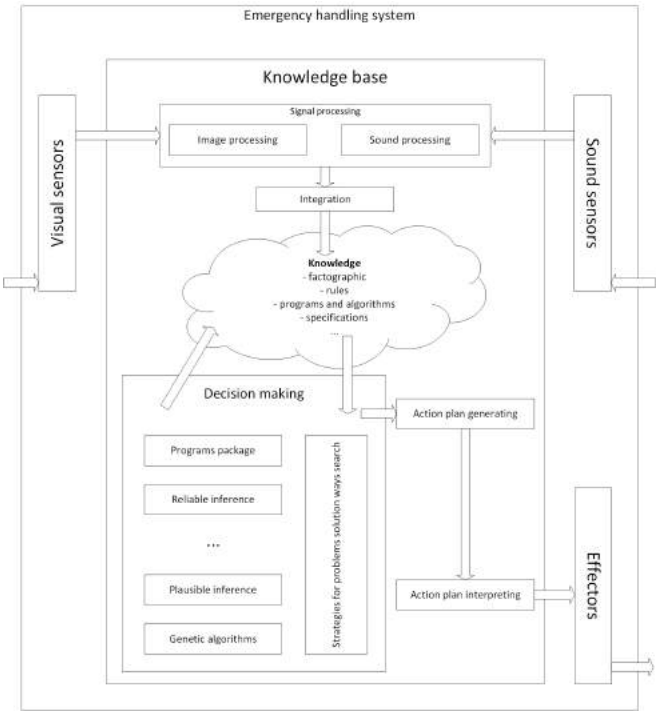

Figure 2. The simplified scheme of the contingencies handling subsystem

A. *Problems in the development of problem solvers*

Despite the fact that there are currently a lot of problem solving models, many of which are implemented and successfully used in practice in various systems [5], [6], [7], [8], the problem of low consistency of the principles underlying the implementation of different models of this kind, leads to the fact that:

- it is difficult to simultaneously use different models of problem solving within the same system for the same complex problem solving, it is practically impossible to combine different models to solve a problem for which there is no a priori algorithm;
- it is practically impossible to use technical solutions implemented in one system in other systems, i.e. the possibility of the component approach implementation in the problem solvers design is very limited. As a consequence, there is a large number of duplications of similar solutions in different systems;
- in fact, there are no complex methods and tools for problem solvers construction that would provide the ability to design, implement and debug solvers of various types.

The consequences of these problems are:

- high complexity of each solver development, increase of their development time, and as a result - the increase of the corresponding intelligent systems development and support cost;

- high complexity of making changes to the already developed solvers, i.e. there is no possibility or it is very difficult to add new components to the already developed solver and to make changes to existing components during the system operation process;
- a high level of professional requirements to the developers of solvers;

B. The proposed approach

The development of a problem solvers model that meets the above requirements, as well as the corresponding method and tools for their construction and modification, is proposed to be implemented within the *OSTIS Technology* [9]. As a formal basis for representation of knowledge within this technology approach, a unified semantic network with set-theoretic interpretation is used. This representation model is called ***SC-code***. Elements of such a semantic network are called sc-nodes and sc-connectors (sc-arcs, sc-edges). The model of an entity, described by means of SC-code will be called the semantic model of the specified entity or simply a *sc-model*. Computer systems, based on OSTIS Technology are called ***ostis-systems***. Each ostis-system consists of a sc-models interpreting platform and the sc-model of this system, which does not depend on the platform of its interpretation.

Orientation to OSTIS Technology and SC-code is due to their following advantages and peculiarities:

- SC-code is oriented to the **sense** representation of knowledge, which allows to generalize the problems solving models and thus significantly reduce the variety of different models, which is largely due to the representation forms of certain knowledge types, rather than their meaning;
- SC-code, unlike other widely used knowledge representation languages, allows to represent in a unified form any kinds of knowledge, including logical statements and programs. This fact makes it possible to unify also the problem solution models on the basis of knowledge presented in this form and to ensure integration of various problem solving models on the basis of such formalism;
- the associativity and structural reconfigurability of the semantic memory (sc-memory), in which the SC-code constructions are stored, makes it possible to provide modifiable models for solving problems, presented on its basis.

In addition, the work involves a number of solutions developed within the technology:

- a graph procedural programming language SCP, programs of which are also written using SC-code and which will be used as the base programming language within the proposed approach;
- models of knowledge base structuring and models of various types of knowledge representation, built on the basis of SC-code, presented in [10];
- the method of consistent construction and modification of knowledge bases, presented in the work [11];
- means for automatic editing and verification of various knowledge types, presented in the same work;
- an implementation version of the computer systems sc-models interpreting platform, considered in the work [12].

At the base of the proposed approach to the previously formulated problems solution are the following principles:

- as a basis for problem solvers design, it is proposed to use a multi-agent approach that will allow to build parallel asynchronous systems, having a distributed architecture, as well as to increase the modifiability and performance of the solvers developed.
- the process of any problem solution is proposed to decompose into logically atomic actions, which will provide compatibility and modifiability of the solvers too.
- the solver is suggested to be considered as a hierarchical system consisting of several interconnected levels. This approach allows to provide the possibility of designing, debugging and verifying components at different levels independently of other levels.
- in order to ensure the reflexivity of the designed intelligent systems, it is proposed to record all information about the solver and the problems it solves with the means of SC-code in the same knowledge base as the subject knowledge of the system. In general, this information includes: (1) the specification of the solver's agents, including the full texts of agent programs (in SCP language); (2) specification of all information processes performed by agents in the semantic memory, including - constructions that ensure synchronization of parallel processes; (3) specification of all problems on solution of which the specified information processes are directed;
- when designing the solver as a hierarchical system, it is suggested to use at each level the component approach, which allows to significantly reduce the development time and improve the reliability of solvers by using well-debugged components. To implement this approach, it is proposed to develop within the IMS metasystem [13] a library of solvers components of various levels, as well as a method for solvers constructing and modifying, which takes into account the existence of such a library.
- it is proposed to build automation and information support tools for solvers developers using OSTIS Technology, that is, including using models, methods and tools offered in this work. Such an approach will allow to ensure high rates of development of these tools, as well as significantly improve the effectiveness of information support tools, allowing to build these tools as part of the intelligent metasystem IMS, that is, as a kind of intellectual subsystem.

Orientation to the multi-agent approach as a basis for the modifiable solvers constructing is due to the following main advantages of this approach [14]:

- autonomy (independence) of agents within such a system, which allows to localize the changes introduced into the

solver during its evolution, and reduce the corresponding labor costs;
- Decentralization of processing, i.e. the absence of a single monitoring center, which also allows to localize the changes introduced into the solver.

The most common and widely used definition of intelligent agent is given in [15].

Existing approaches to the construction of multi-agent systems are discussed in detail in the papers [16], [17], [18], [19], also a specialized journal Autonomous Agents and Multi-Agent Systems [20] is devoted to multi-agent systems.

In the general case, in order to construct a concrete multi-agent system, it is necessary to clarify the following components:

- a model of the agent itself, which is part of such a system, including the classification of such agents and a set of concepts that characterize each agent within the system. Currently, the most popular is the BDI (belief-desire-intention) model, in which it is intended to describe the «beliefs», «desires» and «intentions» of each agent of the system in appropriate languages.
- a model of the environment within agents are located, on the events in which they react and within which they can perform some transformations. A survey on the varieties of environments for multi-agent systems is given in [21].
- an agent communication model that specifies the agent interaction language (the structure and classification of messages) and the way messages are exchanged between agents. Currently, there are a number of standards describing agent interaction languages, for example, KQML [22] and ACL [23].
- a model of agents coordination, regulating the principles of their activities, including mechanisms for resolving possible conflicts. Currently, the most works in the field of multi-agent systems is aimed specifically at mechanisms for coordinating agents, including the allocation of a higher level agents (meta-agents) [24], various socio-psychological models [25], [26], ontology-based behavior [17] and others [27], [28].

The main disadvantages of most popular modern tools for building multi-agent systems [29], [30], [31], [32], [33], [2], [34], [35] include the following:

- the rigid orientation of most tools to the BDI model leads to significant overhead costs associated with the need to express a particular practical task in the BDI concept system. At the same time, the orientation toward the BDI model implicitly provokes the artificial separation of languages, describing the BDI components themselves and the agent's knowledge about the external environment. That leads to the lack of unification of the representation and, correspondingly, to additional overhead costs.
- most modern means of multi-agent systems construction are oriented to the representation of agent's knowledge using highly specialized languages, often not intended to represent knowledge in a broad sense. Here we mean both the agent's knowledge of himself (for example, in accordance with the BDI model) and knowledge about the external environment. In some approaches, an ontology is first constructed, which, however, often uses tools with low expressiveness that are not designed for ontologies building [32], [33]. Ultimately, this approach leads to a strong limitation of the capabilities of the developed multi-agent systems and their incompatibility.
- the absolute majority of modern tools assume that agents interact through messaging directly from the agent to the agent. This approach has a significant disadvantage due to the fact that in this case each agent of the system should have sufficient information about other agents in the system, which leads to additional resource costs. In addition, adding or removing one or more agents leads to the need to notify other agents about that change. This problem is solved by organizing agents' communication on the «blackboard» principle [36], which assumes that messages are placed in some common area for all agents, and each agent in general case may not know to which of the agents the message is addressed and from which agent the message was received. However, this approach does not exclude the problem associated with the need to develop a specialized agent interaction language that is not generally associated with a language that describes the agent's knowledge about the problems to be solved and about the environment.
- a lot of means of multi-agent systems construction are designed in such a way that the logical level of agents' interaction is rigidly tied to the physical level of the multi-agent system implementation. For example, when sending messages from an agent to an agent, the developer of a multi-agent system needs, in addition to the semantically significant information, to specify the ip-address of the computer on which the receiving agent is located, the encoding with which the message text was encoded and other technical information, which depends on the features of the concrete tools implementation.
- in most approaches, the environment with which agents interact is specified separately by the developer for each multi-agent system, which, on the one hand, expands the possibilities of corresponding means using, but on the other hand leads to significant overhead and incompatibility of such multi-agent systems. In addition, in some cases, the developer also has to take into account the specifics of the technical implementation of development tools in terms of their docking with the intended environment, which, for example, can be a local or global network.

Within this work, the listed disadvantages are supposed to be eliminated by using the following principles:

- communication of agents is suggested to be implemented on the basis of the «blackboard» principle, however, unlike the classical approach, in the role of messages there are specifications in the general semantic memory

of the actions (processes) performed by agents and aimed at solving any problems, and the role of communication environment is played by this semantic memory itself. This approach allows to:

  - exclude the need to develop a specialized language for messaging;
  - ensure the «impersonality» of communication, i.e. each agent generally does not know which other agents are in the system, who has formulated the request and to whom this or that request is addressed. Thus, adding or removing agents to the system does not lead to changes in other agents, which ensures the modifiability of the entire system;
  - agents, including the end user, get the opportunity to formulate tasks in the *declarative way*, i.e. do not declare for each problem the way to solve it. Thus, the agent does not need to know in advance how the system will solve a particular problem, it is enough only to specify the final result;

  It should be noted that this approach allows, if necessary, to organize messaging between agents directly, and, thus, can be the basis for modeling multi-agent systems that implement other ways of interaction between agents.
- in the role of the external environment for agents is the same semantic memory, in which problems are formulated and through which agents interact. This approach ensures unification of the environment for different agent systems, which in turn ensures their compatibility.
- the specification of each agent is described by means of SC-code in the same semantic memory, which allows:
  - minimize the number of specialized means required to specify agents, including language and tools;
  - on the one hand - to minimize the necessary specification of the agent in the general case, which includes the condition of its initiation and the program that describes the algorithm of the agent, on the other hand - to provide the possibility of unlimited expansion of such specification for each specific case, including the possibility of implementing the BDI model and others;
- synchronization of the agents activities is supposed to be carried out at the level of the processes performed by them, aimed at solving certain problems in the semantic memory. Thus, each agent is treated as an abstract processor, which is able to solve the tasks of a particular class. With this approach, it is necessary to solve the problem of ensuring the interaction of parallel asynchronous processes in the common semantic memory, for the solution of which the solutions used in traditional linear memory can be adapted.
- each information process at any time has associative access to the necessary fragments of the knowledge base stored in the semantic memory, except of fragments blocked by other processes in accordance with the synchronization mechanism discussed below. Thus, on the one hand, the need to store information about the external environment by each agent is excluded, on the other hand, each agent, like in classical multi-agent systems, has only a part of all the information necessary to solve the problem.

  It is important to note that in the general case it is impossible to predict a priori which knowledge, models and methods of problem solving will be needed for the system to solve a specific problem. In this regard, it is necessary to ensure, on the one hand, the ability to access all the necessary fragments of the knowledge base (in the limit, to the entire knowledge base), on the other hand, to be able to localize the area of the problem solution search, for example, within a single subject domain [11]. Each agent has a set of key elements (usually concepts) that it uses as starting points for associative search within the knowledge base. A set of such elements for each agent is specified at the stages of a multi-agent system design in accordance with the method considered below. Reducing the number of key elements of the agent makes it more universal, but it reduces the effectiveness of its work due to the need to perform additional search operations.

Next, consider the model of knowledge processing and the model of the problem solver itself in accordance with the listed principles.

II. General model of knowledge processing in OSTIS-systems

The model of processing the knowledge stored in semantic memory can be conditionally divided into two components (figure 3):

- the model of information processes performed in such a memory, including the classification of such processes, the mechanisms for their execution regulating, the means for various conflicts solving, including associated with the parallel execution of such processes, means of specifying the state of information processes (executing, delayed, planned, etc.);
- the model of the problem solver, which is treated as an abstract processor that performs the specified information processes, and, accordingly, the model of which is constructed taking into account the model of information processes in the semantic memory. The solver consists of a platform-independent part (solver program) that includes a model of the operational semantics of the SCP language (scp-interpreter model) and a platform-dependent part that includes the implementation of the scp-interpreter. In addition, some components of the solver program can, if necessary, be implemented within the sc-model interpretation platform, for example, to improve the solver performance.

This approach to the knowledge processing organization makes it possible to ensure the independence of the ostis-system sc-model (including the solver program) from the interpretation platform of such models. Thus, the development of

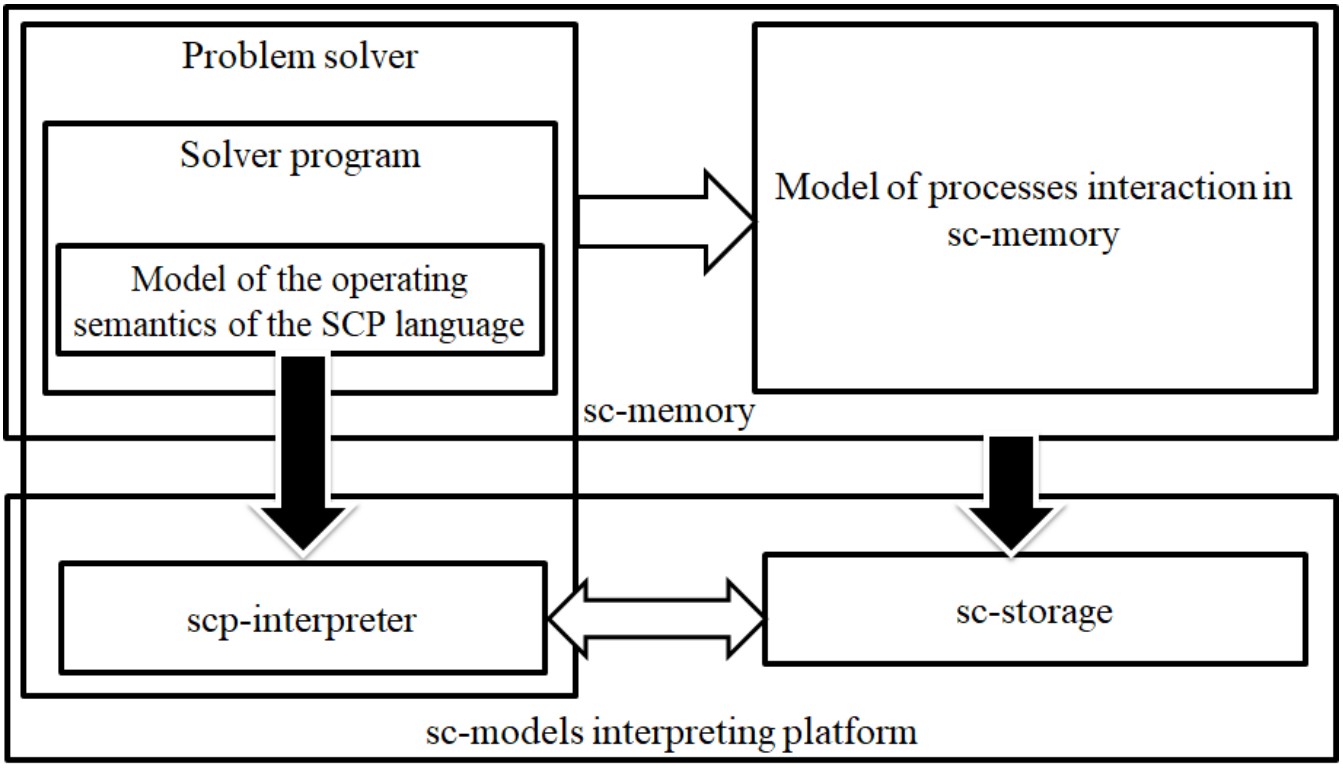

Figure 3. General scheme of knowledge processing organization in ostis-systems

a solver for a particular ostis-system is reduced to developing a program for this solver, i.e. the collective of the solver's agents and the programs corresponding to them. If the agent programs are implemented in the SCP language, such a solver program can be transferred from one implementation of the sc-models interpretation platform to another, including the hardware platform, without any changes.

In its turn, the formal model of the operational semantics of the SCP language is, in fact, a technical task for the implementation of the sc-model interpretation platform, both in software version and in hardware. The complete specification of the denotational and operational semantics of the SCP language is described in the corresponding sections of the IMS metasystem [13].

The development of a certain entity sc-model (including a program, an agent and a solver) supposes a formal refinement of the concepts system that is used to describe this entity in the knowledge base of the ostis-system. To achieve this goal, in accordance with the principles of OSTIS Technology, it is necessary to develop the sc-models of one or several interconnected *subject domains* and the corresponding *ontologies* [10].

The implementation of the approach proposed in this work requires the construction of several subject domains (SD) sc-models connected with each other, as shown in the figure 4.

Next, the most important concepts that are *researched concepts* [10] within the specified subject domains will be discussed in more detail.

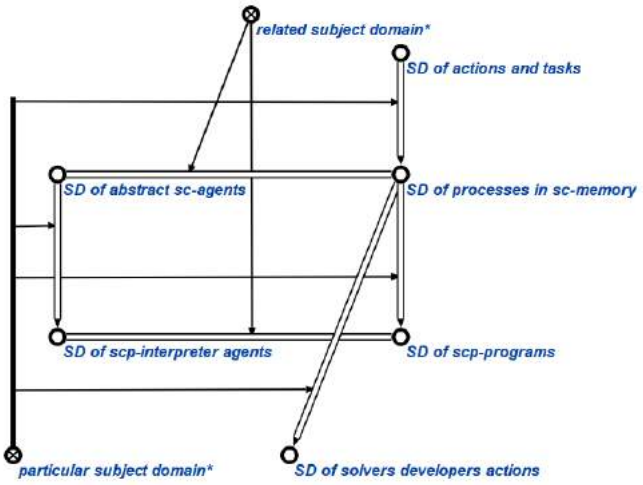

Figure 4. Hierarchy of subject domains

A. Means of specification of problems being solved

As it was said before, the principle of communication of sc-agents within the proposed approach is based on the specification by sc-agents of all the actions performed by them in semantic memory, i.e. in the formal description in the knowledge base of all problems being solved.

In the proposed approach, the problem is treated as a specification of some action, which in general can include such information as: the subject and the object of the performed action; the timing of the problem, its priority, dependent problems, etc .; detailed specification of the specified action execution process by its decomposition into sub-actions (procedural formulation of the problem); specification of the action goal (declarative formulation of the problem), etc.

The figure 5 shows an example of an information task in SCg language.

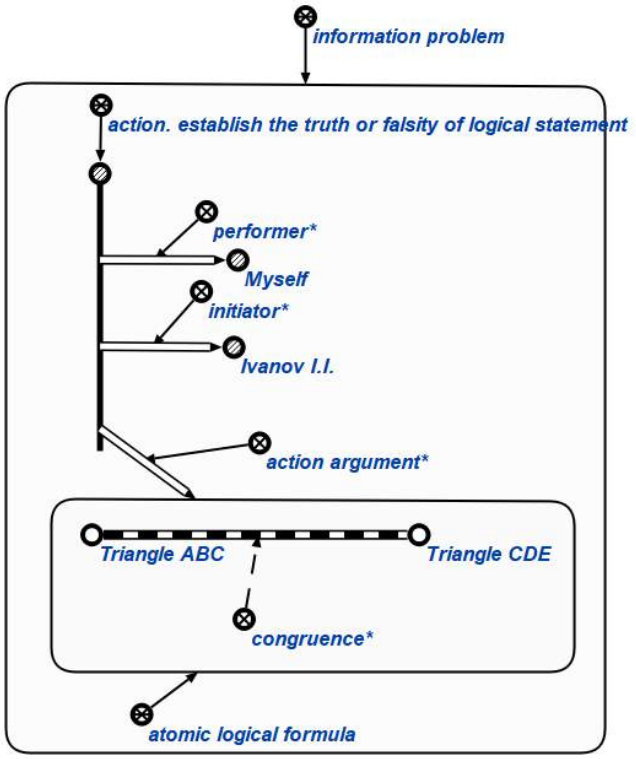

Figure 5. The problem of establishing the truth or falsity of a statement

The model of the activity performed by agents while solving problems in semantic memory is defined as follows:

$$M_A = \{A_C, A_{CM}, A_R, A_{CS}\}, \quad (1)$$

where A_C – set of classes of actions performed by different subjects;

A_{CM} – set of classes of actions performed by agents in semantic memory, $A_{CM} \subset A_C$;

A_R – set of relations specifying actions belonging to classes from A_C;

A_{CS} – a set of specification classes of actions belonging to classes from A_C, such as a task, a protocol for an action executing, etc.;

In more detail, the means for specifying the actions performed in semantic memory are discussed in [37].

One of the basic principles underlying the proposed approach to the construction of solvers is the principle of dividing the process of any problem solving in an intelligent system by *logically atomic actions*.

We will assume that each *action* belonging to some particular *class of logically atomic actions* has two necessary properties:

- The execution of an action does not depend on whether the specified action is part of the decomposition of a some general action. When this action is carried out, it should also not take into account the fact that the action precedes or follows any other action;
- the specified action should be a logically complete act of transformation, for example, in semantic memory. Such an action is essentially a transaction, i.e. the result of this transformation is the new state of the system being converted, and the action to be performed must either be executed completely, or not executed at all, partial execution is not allowed.

At the same time, logical atomicity does not prevent decomposition of the executed action into partial ones, each of which in turn will also belong to some class of logically atomic actions.

B. The concept of sc-agent

The only kind of entities that perform transformations in *sc-memory* is *sc-agents*. We will call an *sc-agent* some *subject* which is able to execute *actions in sc-memory*, belonging to some particular *class of logically atomic actions*.

Formally, the sc-agent model is defined as follows:

$$M_S = \{S_C, S_R\}, \quad (2)$$

where S_C – a set of classes (types) of sc agents; S_R – set of relations defined on the set of sc-agents.

Logical atomicity of sc-agent actions assumes that each sc-agent responds to the corresponding class of events occurring in the sc-memory, and performs a certain transformation of the sc-text (SC-code text) located in the semantic neighborhood of the event being processed. In this case, each sc-agent in general does not have information about what other sc-agents are currently present in the system and interacts with other sc-agents solely through the formation of certain constructions (usually the action specifications) in the common sc-memory. Such a message can be, for example, a question addressed to other sc-agents in the system (not known in advance what exactly) or the answer to the question formed by other sc-agents (again, it is not known exactly what). Thus, each sc-agent controls only the knowledge base fragment in the context of the problem solved by the agent at any given time, the state of the rest of the knowledge base is unpredictable for the sc-agent in the general case.

To ensure the availability of such a multi-agent system, it is necessary that each sc-agent in its composition specifies in the sc-memory all the results of its actions. It is assumed that after solving a certain problem:

- all the intermediate constructs generated in the solution process and having no meaning outside this process must be deleted;

- all processes (actions) in sc-memory, aimed at solving the same problem, should be terminated, except when it is supposed to receive several independent responses to the same questions.

It is important to note that the end-user of the ostis-system in terms of knowledge processing also acts as an sc-agent, forming the messages in the sc-memory by performing the elementary actions provided by the user interface. In the same way the ostis-system interacts with other systems and the environment in general. All information gets in and out the ostis-system exclusively through the appropriate sc-agents of the interface.

Let's list some advantages of the offered approach to the organization of knowledge processing in sc-memory:

- because of processing is performed by agents that exchange messages only via common memory, adding a new agent or excluding (deactivating) one or more existing agents usually does not result in changes to other agents, since agents do not exchange messages directly;
- the agents are initiated in a decentralized manner and, most often, independently of each other. Thus, even a significant increase of the agents number within the same system does not lead to its productivity reduce;
- agent specifications and, as will be shown below, their programs can be written in the same language as the processed knowledge, which significantly reduces the list of specialized means designed to develop such agents and their groups, and simplifies system development by using more universal components;

Since it is supposed that copies of the same *sc-agent* or functionally equivalent *sc-agents* can work in different ostis-systems, being physically different sc-agents, it is advisable to consider properties and classification of not sc-agents, but classes of functionally equivalent sc-agents, which we will call *abstract sc-agents*. Thus, an *abstract sc-agent* is a class of functionally equivalent *sc-agents*, different instances (i.e. elements) of which can be implemented in different ways.

Each *abstract sc-agent* has a corresponding specification, which specifies the key sc-elements of the specified abstract sc-agent, as well as a description of the initiating condition for this *sc-agent*, i.e. class of those *situations* in *sc-memory* that initiate the activity of this *sc-agent*. In addition, for each abstract sc-agent, the variant of its implementation is specified. From the implementation point of view, two classes of *abstract sc-agents* can be distinguished:

- *non-atomic abstract sc-agent*, which is decomposed into a group of simpler *abstract sc-agents*, each of which in turn can be both *atomic abstract sc-agent*, and *non-atomic abstract sc-agent*. However, in some version of *abstract sc-agent decomposition**, the child *non-atomic abstract sc-agent* can become an *atomic abstract sc-agent*, and be implemented accordingly.
- *atomic abstract sc-agent* is an abstract sc-agent, for which the platform of its implementation is specified, i.e. there is a corresponding connection of the *sc-agent program** relation.

In turn, *atomic abstract sc-agents* are subdivided into *platform-independent abstract sc-agents* and *platform-dependent abstract sc-agents*.

The *platform-independent abstract sc-agents* are *atomic abstract sc-agents* implemented in *SCP language*.

When describing *platform-independent abstract sc-agents*, platform independence is understood from the point of view of OSTIS Technology, i.e. implementation in the SCP language, because *atomic sc-agents* implemented in the specified language can be easily transferred from one sc-models interpretation platform to another.

The *platform-dependent abstract sc-agents* are *atomic abstract sc-agents*, implemented below the level of sc-models, i.e. not in the *SCP language*, but in some other language of the program description.

An example of an atomic abstract sc-agent specification including the agent program, its key sc-elements, and the initiation condition is given in the figure 6.

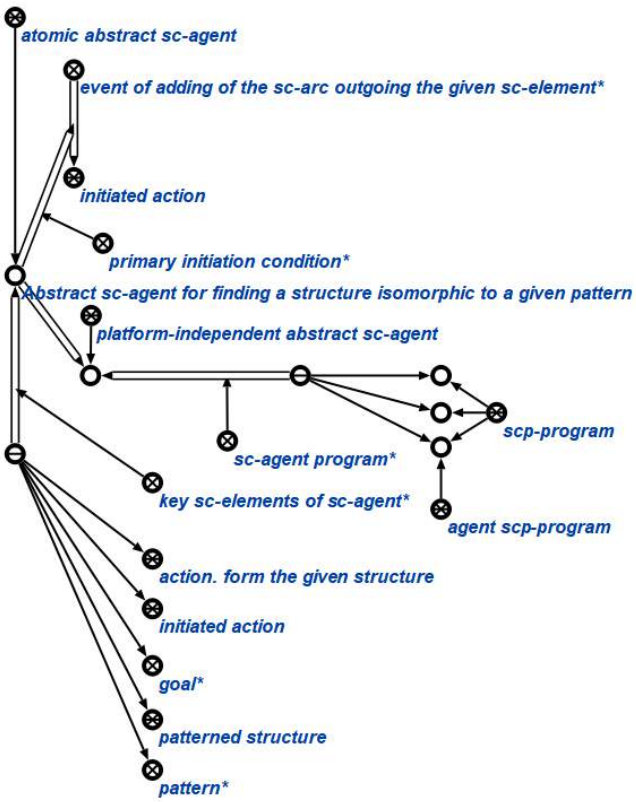

Figure 6. Atomic abstract sc-agent of search for structures isomorphic to a given pattern

The construction of non-atomic sc-agents allows to ensure the hierarchy of the designed multi-agent system and the ability to consider it at different levels of detail, which, in turn, provides the convenience of such a system designing and debugging through the ability to design and debug components

of varying complexity independently. In addition, the allocation of non-atomic sc-agents is the basis for the formation of a hierarchical library of problem solvers reusable components, which will include components of different complexity levels, including even entire solvers.

C. Basic model of knowledge processing

The *SCP language*, developed within the OSTIS Technology, is proposed as the base development language for the programs describing the activity of sc-agents within *sc-memory*. *SCP language* is focused on processing of unified semantic networks represented with *SC-code*.

The basic *sc-text* processing model includes:

- model of *scp-programs* subject domain, which includes all the texts of *scp-programs*, and in which the classification of these programs operators and the means of their specification are researched.
- model of subject domain of the scp-programs interpretation agents (also called *Abstract scp-machine*), which is part of the sc-model interpretation platform. The term *abstract* in this case, as in the case of the *abstract sc-agent*, shows that a semantic model of the scp-interpreter is being developed, including the specification of each agent in its composition, which can later be implemented within the any sc-model interpretation platform, including hardware.

The main features of the SCP language include the following:

- texts of scp-programs are written using SC-code, as well as processed information;
- each *scp-program* is a generalized structure in sc-memory, each time a *scp-program* is called, an independent *scp-process* is created on its basis.

The main advantages of the SCP language, due to these features:

- simultaneously several independent processes can be executed in the common memory, and processes corresponding to the same *scp-program* can be executed on different servers, in case of distributed implementation of the sc-model interpretation platform.
- *SCP language* allows concurrent asynchronous subroutine calls (creating subprocesses within scp-processes), as well as concurrently execute of scp-operators within a single *scp-process*;
- since *scp-programs* are written using *SC-code*, the transfer of the *sc-agent* implemented with the SCP language from one system to another supposes a simple transfer of the knowledge base fragment, without any additional operations, which depends on the sc-model interpretation platform;
- the fact that sc-agents' specifications and their programs can be written in the same language as the knowledge they are processing significantly reduces the list of specialized tools, designed to build and modify problem solvers and simplifies their development by using more universal components;
- the fact that a unique *scp-process* is created for the *scp-program* interpreting allows to optimize the execution plan as much as possible before its implementation and even directly during the execution without potential danger to break the general algorithm of the entire program. Moreover, this approach to programs designing and interpreting allows to talk about the possibility of creating self-reconfigurable programs;

III. Semantic model of problem solver

Using the concepts discussed above, we will say that the *sc-model of the integrated problem solver* is a non-atomic abstract sc-agent, which is the result of combining of all abstract sc-agents within a particular ostis-system into one. In other words, the *sc-model of the integrated problem solver* is the collective of all sc-agents within a given ostis-system, considered as a single whole.

Formally, the semantic model of the integrated problem solver is given as follows:

$$M_{IPS} = \{AG_{NA}, AG_A, AG_R\}, \tag{3}$$

where AG_{NA} – a set of non-atomic abstract sc-agents within the solver;

AG_A – a set of atomic abstract sc-agents within the solver;

AG_R – a set of concepts specifying abstract sc-agents within the solver, including those describing the decomposition of non-atomic agents into atomic ones;

There are several basic levels of detail for the problem solver:

- level of the solver itself;
- level of non-atomic sc-agents within the solver, including particular solvers;
- level of atomic sc-agents;
- level of scp-programs or programs implemented at the level of *sc-model interpretation platforms*.

Such a level hierarchy, firstly, provides the possibility of step-by-step design of task solvers with a gradual increase of the detail level from the upper to the lower, and secondly, the possibility of designing, debugging and verifying components at different levels independently from other levels, which significantly simplifies the task of solvers construction and modification due to lower overhead costs.

In addition, the proposed approach to the construction of a solver model allows to provide modifiability of solvers and the possibility of consistent use of different problem solving models within a single solver.

IV. Semantic model of parallel processes interaction in the common semantic memory

Taking into account the models considered earlier, a formal model of information processes interaction in semantic memory is constructed, which is defined as follows:

$$M_{IPM} = \{M_A, M_S, M_{SYNC}, M_{SCP}\}, \tag{4}$$

where M_A – model of the activity performed by agents in semantic memory;

M_S – model of sc-agent that performs transformations in semantic memory;

M_{SYNC} – model of processes execution synchronization in the semantic memory;

M_{SCP} – a model of a basic programming language, oriented to the processing of unified semantic networks, which, in turn, is defined as:

$$M_{SCP} = \{M_P, M_I\}, \tag{5}$$

where M_P – model of basic programming language program; M_I – model of basic programming language programs interpreter;

To synchronize the execution of *processes in sc-memory*, the lock mechanism is used. The *lock** relation connects the signs of the *processes in the sc-memory* to the signs of the situational *structures* that contain sc-elements that are locked for the duration of this process or for some part of this period. Each such *structure* belongs to any of the *lock types*.

In the current version, three *lock types* are distinguished to synchronize the processes execution in the sc-memory:

- *full lock*;
- *lock on any change*;
- *lock on delete*.

In turn, from the point of view of synchronization tools, three classes of *sc-agents* can be distinguished:

- textit sc-agents of scp-programs interpreting, which are implemented at the sc-models interpretation platform level and one of tasks of which is to provide the described synchronization mechanism. In turn, the principles of synchronization of agents of this class are more trivial than in the case of *program sc agents*, and are described separately.
- textitprogram sc-agents, providing the main functionality of the system, that is, its ability to solve certain tasks and working in accordance with the considering mechanism.
- *sc-metaagents*, task of which is to coordinate the activity of *program sc-agents*, in particular, to solve the deadlocks problem.

For more details on the locks mechanism in semantic memory, see [38], [39].

Thus, each sc-agent can generally correspond to several concurrently running processes in the sc-memory, the interaction of which is regulated by the described locks mechanism.

An example of a description of locks in semantic memory that correspond to several processes in this memory is shown in the figure 7.

V. Method for problem solvers constructing and modifying

As mentioned above, all platform-independent components of problem solvers can be represented using SC-code. In this case, we are talking about the specifications of sc-agents, and the full texts of scp-programs that describe the algorithms of these agents.

Thus, the construction of the ostis-system problem solver is reduced to the development of a special kind of knowledge base fragment of such a system. In this regard, when constructing and modifying solvers, all existing automation tools for the knowledge bases construction and modifying using OSTIS Technology can be used, considered, in particular, in the work [11].

Due to that, when constructing a method and means for solvers developing, it is necessary to clarify only some aspects of the development that are specific to the problem solvers based on the model considered earlier. It is important to note that, according to the model presented earlier, the task solver is an *abstract sc-agent*, and therefore the development of the solver is reduced to the such an agent development.

To develop problem solvers on the basis of the solver model considered, a method is proposed that assumes the use of formal ontology of such solvers developers activity for solvers development and is oriented to the use of reusable solver components at each level of the structural hierarchy of the solver being developed, which makes it possible to reduce the complexity of their development.

The proposed method includes several stages (figure 8):

The formally proposed method for solvers constructing and modifying is defined as follows:

$$M_{PA} = \{PA_C, PA_R\}, \tag{6}$$

where PA_C – a set of classes of actions performed by the solver developers;

PA_R – set of relations specifying these actions, including relations, specifying the order of actions implementation and decomposition of some actions into sub-actions.

The main advantage of the proposed method is its orientation to the ontology of the solver developers activity, which, on the one hand, will allow to automate this activity, and on the other hand, will allow to present the specifications of this activity within the knowledge base of the IMS metasystem and thus provide information support to the solvers developers.

VI. Tools for problem solvers constructing and modifying

To implement the proposed method, tools were developed to automate the process of problem solvers construction and modifying (TAPCM). The tools include a system for automating the process of solvers constructing and modifying, which in turn is conditionally divided into an automation system for the agents development and automation system for scp-programs development, as well as the library of solvers reusable components as part of the IMS metasystem for ostis-systems design support. Schematically, the architecture of the tools is shown in the figure 9:

In its turn, the library includes:

- the set of problem solvers components;
- means for specifying the problem solvers components;
- search tools for problem solver components based on their specification;

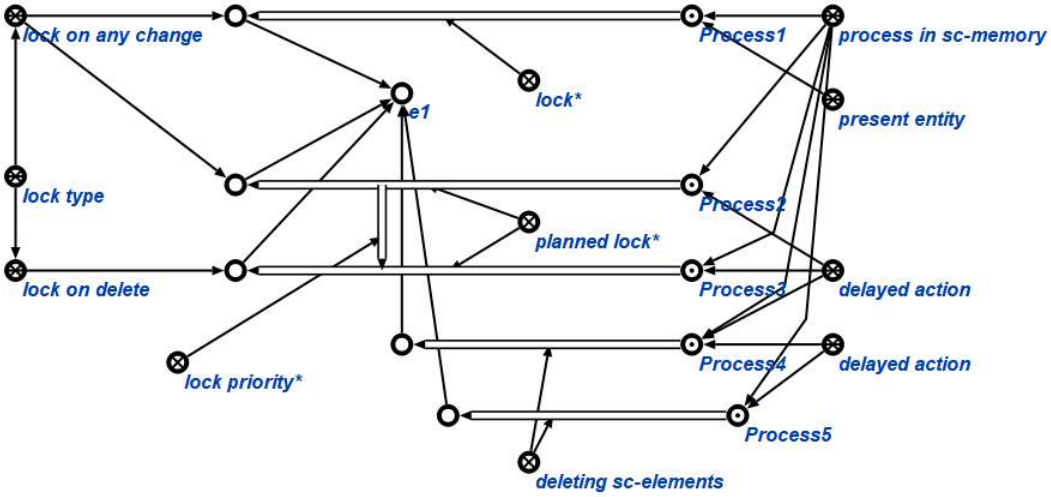

Figure 7. Example of the specification of locks in semantic memory

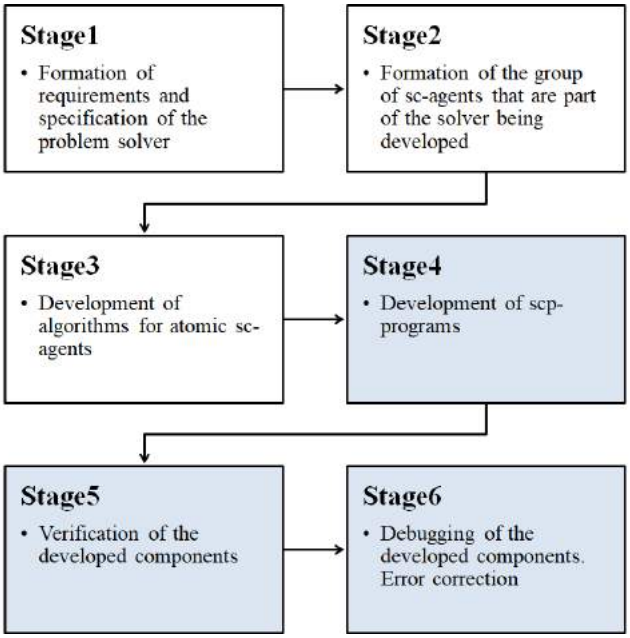

Figure 8. Stages of the method for problem solvers constructing and modifying

General structure of *Library of problem solvers reusable components* in the language SCn:

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

}

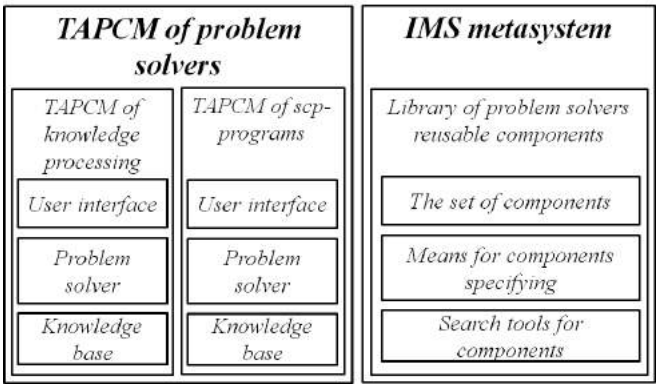

Figure 9. The architecture of tools for problem solvers constructing and modifying

In turn, the *Library of reusable abstract sc-agents* has the following structure:

Library of reusable abstract sc-agents
= *reusable abstract sc-agent*
<= *subdividing**:
{
- *Library of information search sc-agents*
- *Library of sc-agents of integrable knowledge immersion into the knowledge base*
- *Library of sc-agents for aligning the ontology of integrable knowledge with the basic ontology of the knowledge base current state*
- *Library of sc-agents for planning solutions to explicitly formulated tasks*
- *Library of logical inference sc-agents*
- *Library of sc-models of high-level programming languages and their interpreters*
- *Library of sc-agents of knowledge base verification*

- *Library of sc-agents of knowledge base editing*
- *Library of sc-agents for automation of activity of knowledge-base developers*

}

The solvers debugging tools developed provide the possibility of debugging on two levels of the solver:

- debugging at the sc-agents level;
- debugging at the scp-program level;

In the case of debugging at the sc-agents level, the act of each agent executing is considered to be indivisible and can not be interrupted. At the same time, debugging of atomic sc-agents and non-atomic agents can be performed. Initiation of an agent, including one that is part of a non-atomic one, is carried out by creating appropriate structures in the sc-memory, so debugging can be performed at different levels of agent detail, up to atomic levels.

Debugging at the level of sc-agents involves the ability to set or release the locks, enable or disable any agents, etc. Taking into account that the model of agent interaction proposed in this paper uses a universal variant of interaction of agents through common memory, the considered system of agent design support can serve as a basis for modeling systems of agents using other principles of communication [40], [41], for example, direct exchange of messages between agents.

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

The main feature of the considered tools for solver constructing and modifying is their implementation on the basis of OSTIS Technology, that is, including the use of previously considered solver model for the tools constructing. This feature allows to ensure the modifiability of the tools themselves, i.e. ease of their functionality expansion, including by using components from the permanently extending library of solvers reusable component.

VII. Conclusion

The proposed models, method and tools have been successfully applied to the development of problem solvers for intelligent reference systems in various academic disciplines, such as geometry, graph theory, history, chemistry, as well as in the development of a prototype automation system for batch production enterprises [42]. On certain examples, the proposed approach showed such advantages as the universality of the developed agents and the ease of modification of solvers constructed on the basis of the proposed model.

Acknowledgment

This work was supported by BRFFR (BRFFR-RFFR №Ф16Р-102, BRFFR-SFFRU №Ф16К-068).

Список литературы

[1] A. Kolesnikov, I. Kirikov, and S. Listopad, Gibridnye intellektual'nye sistemy s samoorganizatsiei: koordinatsiya, soglasovannost', spor [Hybrid intelligent systems with self-organization: coordination, consistency, dispute]. M.: IPI RAN, 2014, (in Russian).

[2] O. Castillo, P. Melin, and J. Kacprzyk, Recent Advances on Hybrid Intelligent Systems. Springer, 2014.

[3] S. Nikolenko and T. A.L., Samoobuchayushchiesya sistemy [Self-learning systems]. M.: MTsNMO, 2009, (in Russian).

[4] C. Kaner, J. Falk, and H. Nguyen, Testing Computer Software, 2nd ed. Wiley, 1999.

[5] E. Efimov, Reshateli intellektual'nykh zadach [Intelligent problem solvers]. M.: Nauka, 1982, (in Russian).

[6] A. Podkolzin, Komp'yuternoe modelirovanie logicheskikh protsessov. Arkhitektura reshatelya i yazyki reshatelya zadach [Computer modeling of logical processes. Solver architecture and problem solver languages]. M.: Fizmatlit, 2008, (in Russian).

[7] (2017, Jun.) Wolfram alpha. [Online]. Available: http://www.wolframalpha.com/

[8] A. Vladimirov, O. Varlamov, A. Nosov, and T. Potapova, "Programmnyi kompleks "UDAV": prakticheskaya realizatsiya aktivnogo obuchaemogo logicheskogo vvoda s lineinoi vychislitel'noi slozhnost'yu na osnove mivarnoi seti pravil [the program complex "UDAV": practical implementation of the active learning logical input with linear computational complexity on the basis of a miwar rule network]," Trudy nauchno-issledovatel'skogo instituta radio [Proceedings of the radio research institute], no. 1, pp. 108–116, 2010, (in Russian).

[9] V. Golenkov and N. Gulyakina, "Proekt otkrytoi semanticheskoi tekhnologii komponentnogo proektirovaniya intellektual'nykh sistem. Chast' 2: Unifitsirovannye modeli proektirovaniya [project of open semantic technology of component design of intelligent systems. part 2: Unified design models]," Ontologiya proektirovaniya [Ontology of design], no. 4, pp. 34–53, 2014, (in Russian).

[10] I. Davydenko, N. Grakova, E. Sergienko, and A. Fedotova, "Sredstva strukturizatsii semanticheskikh modelei baz znanii [Means of semantic models of knowledge bases structuring]," in Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh sistem [Open semantic technologies for intelligent systems]. Minsk.: BSUIR, 2016, pp. 93–106, (in Russian).

[11] I. T. Davydenko, "Ontology-based knowledge base design," in Open semantic technologies for intelligent systems (OSTIS-2017): materials of VII International.sc.-tech.conf. Minsk: BSUIR, 2017, pp. 57–72.

[12] D. Koronchik, "Realizatsiya khranilishcha unifitsirovannykh semanticheskikh setei [implementation of the unified semantic networks storage]," in Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh sistem [Open semantic technologies for intelligent systems]. Mn.: BSUIR, 2013, pp. 125–129, (in Russian).

[13] (2017, Jun.) IMS metasystem. [Online]. Available: http://ims.ostis.net/

[14] M. Wooldridge, An Introduction to MultiAgent Systems - Second Edition. Wiley, 2009.

[15] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall, 2010.

[16] V. Gorodetskii, M. Grushinskii, and A. Khabalov, "Mnogoagentnye sistemy (obzor) [Multi-agent systems (survey)]," Novosti iskusstvennogo intellekta [Artificial intelligence news], no. 2, pp. 64–116, 1998, (in Russian).

[17] V. Gorodetskii, V. Samoilov, and D. Trotskii, "Bazovaya ontologiya kollektivnogo povedeniya avtonomnykh agentov i ee rasshireniya [Basic ontology of autonomous agents collective behavior and its extension]," Izvestiya RAN. Teoriya i sistemy upravleniya [Proceedings of the RAS. Theory and control systems], no. 5, pp. 102–121, 2015, (in Russian).

[18] V. Tarasov, Ot mnogoagentnykh sistem k intellektual'nym organizatsiyam [From multi-agent systems to intelligent organizations]. M.: Editorial URSS, 2002, (in Russian).

[19] V. F. Răzvan, Autonomous artificial intelligent agents. Romania: Center for Cognitive and Neural Studies (Coneural), 2003.
[20] (2017, Jun.) Autonomous agents and multi-agent systems. [Online]. Available: http://www.springer.com/computer/ai/journal/10458
[21] D. Weyns, A. Omicini, and J. Odell, "Environment as a first class abstraction in multiagent systems," ser. 1, vol. 14, February 2007, pp. 5–30.
[22] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "Kqml as an agent communication language," in Proceedings of the third international conference on Information and knowledge management - CIKM '94, 1994.
[23] (2017, Jun.) FIPA ACL Message structure specification. [Online]. Available: http://www.fipa.org/specs/fipa00061/SC00061G.html
[24] R. L. Hartung and A. Hakansson, "Using meta-agents to reason with multiple ontologies," in Agent and Multi-Agent Systems: Technologies and Applications, 2008, pp. 261–270.
[25] W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman, "Normative conflict resolution in multi-agent systems," ser. 2, vol. 19, June 2009, pp. 124–152.
[26] T. Rumbell, J. Barnden, S. Denham, and T. Wennekers, "Emotions in autonomous agents: comparative analysis of mechanisms and functions," ser. 1, vol. 25, June 2012, pp. 1–45.
[27] A. Omicini and F. Zambonelli, "Coordination for internet application development," Autonomous Agents and Multi-Agent Systems, vol. 2, no. 2, pp. 251–269, June 1999, issue 2.
[28] M. V. Nagendra Prasad and V. R. Lesser, "Learning situation-specific coordination in cooperative multi-agent systems," Autonomous Agents and Multi-Agent Systems, vol. 2, no. 2, pp. 173–207, 1999, issue 2.
[29] (2017, Jun.) GOAL agent programming language. [Online]. Available: https://goalapl.atlassian.net/wiki/spaces/GOAL/overview
[30] (2017, Jun.) GAMA Platform. [Online]. Available: http://gama-platform.org/
[31] (2017, Jun.) EVE - a web-based agent platform. [Online]. Available: http://eve.almende.com/index.html
[32] R. Evertsz, M. Fletcher, R. Jones, J. Jarvis, J. Brusey, and S. Dance, Implementing Industrial Multi-agent Systems Using JACK. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 18–48.
[33] (2017, Jun.) JAVA Agent DEvelopment framework. [Online]. Available: http://jade.tilab.com/
[34] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with jacamo," Sci. Comput. Program., vol. 78, no. 6, pp. 747–761, Jun. 2013.
[35] M. W. Rafael H. Bordini, Jomi Fred Hübner, Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley, 2007.
[36] V. Jagannathan, K. Dodhiawala, and L. Baum, Blackboard Architectures and Applications. N.Y.: Academic Press, 1989.
[37] D. Shunkevich, A. Gubarevich, M. Svyatkina, and O. Morosin, "Formal'noe semanticheskoe opisanie tselenapravlennoi deyatel'nosti razlichnogo vida sub"ektov [Formal semantic description of the purposeful activity of various types of subjects]," in Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh sistem [Open semantic technologies for intelligent systems]. Mn.: BSUIR, 2016, pp. 125–136, (in Russian).
[38] D. Shunkevich, "Vzaimodeistvie asinkhronnykh parallel'nykh protsessov obrabotki znanii v obshchei semanticheskoi pamyati [Interaction of asynchronous parallel processes of knowledge processing in common semantic memory]," in Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh sistem [Open semantic technologies for intelligent systems]. Mn.: BSUIR, 2016, pp. 137–144, (in Russian).
[39] D. V. Shunkevich, "Ontology-based design of knowledge processing machines," in Open semantic technologies for intelligent systems (OSTIS-2017): materials of VII International.sc.-tech.conf. Minsk: BSUIR, 2017, pp. 73–94.
[40] C. M. Macal and M. J. North, "Tutorial on agent-based modeling and simulation," in Proceedings of the 2005 Winter Simulation Conference. WSC'05, 2005, pp. 2–15.
[41] M. Marietto, N. David, J. S. Sichman, and H. Coelho, "Requirements analysis of agent-based simulation platforms: State of the art and new prospects," in Third International Workshop of Multi-Agent-Based Simulation. Bologna, 2002, pp. 125–141.
[42] V. Taberko, D. Ivanyuk, V. Golenkov, K. Rusetskii, D. Shunkevich, I. Davydenko, V. Zakharov, V. Ivashenko, and D. Koronchik, "Proektirovanie predpriyatii retsepturnogo proizvodstva na osnove ontologii [Designing enterprises of batch production on the basis of ontologies]," Ontologiya proektirovaniya [Ontology of design], no. 2, pp. 123–144, 2017, (in Russian).

АГЕНТНО-ОРИЕНТИРОВАННЫЕ МОДЕЛИ, МЕТОД И СРЕДСТВА РАЗРАБОТКИ СОВМЕСТИМЫХ РЕШАТЕЛЕЙ ЗАДАЧ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Шункевич Д.В. (БГУИР)

Статья посвящена разработке агентно-ориентированных моделей, метода и средств разработки совместимых решателей задач интеллектуальных систем, способных решать комплексные задачи. Рассматриваются требования, предъявляемые к таким решателям, модель решателя задач, удовлетворяющего предъявленным требованиям, а также метод и средства разработки и модификации таких решателей.

Основной проблемой, решаемой в работе, является проблема низкой согласованности принципов, лежащих в основе реализации различных моделей решения задач. Как следствие, затруднена возможность одновременного использования различных моделей решения задач в рамках одной системы при решении одной и той же задачи, практически невозможно повторно использовать технические решения, реализованные в некоторой системе, фактически отсутствуют комплексные методы и средства построения решателей задач.

Как основу для построения решателей задач предлагается использовать многоагентный подход. Процесс решения любой задачи предлагается декомпозировать на логически атомарные действия, что позволит обеспечить совместимость и модифицируемость решателей. Сам решатель предлагается рассматривать как иерархическую систему, состоящую из нескольких взаимосвязанных уровней, что позволяет обеспечить возможность проектирования, отладки и верификации компонентов на разных уровнях.