Ontology-based Design of Knowledge Processing Machines

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

***Abstract*—The work is devoted to the development of intelligent systems knowledge processing machines design technology, which is based on an ontological model of the machine itself and the ontological model of the design process. A model of system of knowledge processing machines design support also considered.**

***Keywords*—knowledge processing machine, ontology, subject domain, ontology-based design, intelligent system, multi-agent systems, parallel systems, information search, logical inference.**

I. Introduction

A. Objective and Relevance of the Work

The objective of this paper is the creation of technology of design of efficient and flexible knowledge processing machines of different knowledge-based systems and reducing of time and overheads of such machines development.

Nowadays the application of intelligent systems becomes more and more urgent in different spheres of human activity, especially in those of them, where human being can be dangerous or can invoke errors, caused by so-called human factor. One of the most perspective branches in this area is the development of knowledge-based systems [19]. Ontologies are widely used as the formal basis of knowledge representation is such kind of systems [5].

One of the most important components of such a system is ***knowledge processing machine***, which provides an opportunity to solve different problems, related as with basic functionality of system (machine of information search and intelligent problem solver), so as with providing of correctness of system workability itself (information garbage collection, knowledge base quality enhancement etc.) and with providing of system evolution automation. It should be noted, that tasks, solved by any of knowledge processing machine components may not be explicitly formulated. Examples of such kind of tasks are information garbage detection and deleting, knowledge base optimization and so on.

Approaches to problem solvers development can be divided into two main classes:

- **problem solution using stored programs**. This case supposes that system already contains a program of given problem class solution, so solution reduces to search of that program and its interpreting with given input data. This class includes also systems, which use genetic algorithms [26], [37] and knowledge processing models, based on neural networks [22], [30].
- **problem solution in conditions, when solution program is unknown.**This case supposes that there may not be a program of given problem class solution, so it is necessary to apply additional methods of problem solution search, which are not oriented on the narrow class of tasks (for example, halving method, depth-first and breadth-first solution search method, method of random solution search, trial-and-error method, method of dividing tasks into subtasks, so on), as well as different models of logical inference (classic deductive [25], inductive [42], [43], abductive [25]; models, based on fuzzy logic [44], [21], [35], temporal logic [38], so on).

So, there are a lot of approaches to development of difference knowledge processing machines for different computer systems, including intelligent problems solvers, many of which are successfully implemented and being actively used.

Let's consider two main historically formed approaches to knowledge processing machines development.

First approach supposes that system contains fixed knowledge processing machine (for example, logical inference machine), and afterward there can be added a knowledge base, content of which is defined by subject domain, with which system will work. Such systems were named "empty" expert systems [46] or expert system shells [8]. This approach was generally used for development of relatively simple systems and in present time has no wide application.

Second approach, popular in present time, supposes using of software tools for access to information, stored in some knowledge base, which are compatible with different popular programming languages. This approach is widely used, for example, in systems, based on the W3C standards [18]. Structure of knowledge processing machine, based on that approach if defined by the developer in every case and is not regulated by any standards. Such an approach is more flexible, but lack of unification in knowledge processing machine structure and development process leads to inconsistence of machine components if they are designed by different developers. The result is a duplication of the similar solutions and rising of overheads during the development and support process.

It is evident that every intelligent system needs its own unique knowledge processing machine, which considers that system features and supposes an opportunity of its fast adjustment in case of necessary. But most of modern systems has fixed knowledge processing machine, which is able to solve problems from small limited class (for example, execute deductive logical inference, based on the simple if-then rules).

In the same time, the actual question is how to use different problem solution methods in one system in one time. The actuality is caused by high demand of systems, which can work autonomously in unpredictable conditions (for example, in open space or other places, which are unsuitable for humans). Systems, which use fixed set of algorithms cannot satisfy that requirement, so the next actual question is how to provide an opportunity of fast just-in-time enhancement of system abilities.

For example, it is evident, that for even simple autonomous system functioning there must be abilities of decision-making in conditions of uncertainty (for example, logical inference, as reliable, so as fuzzy and probabilistic); of analyzing of signals, received from different sensors, including tools for image processing; of analysis of the consequences of its own activity with the possibility of automatic adjustment of next acts of this activity; of predicting of the future environment states on the basis of the collected data, so on.

It is important, that we are not talking about the creation of a single universal knowledge processing machine, implementing all possible models and solve all possible problems. We are talking about the development of models and means that would support the creation of flexible and efficient knowledge processing machine of different systems within an acceptable timeframe.

B. Problems, Need to be Solved

In spite of the fact that there are a lot of models, methods and means of the knowledge processing, many of which have been successfully implemented in various systems, there still are:

- lack of common universal principles, underlying the implementation of various knowledge processing models leads to a lot of duplications of similar solutions in different systems and the inability to use the solutions, implemented in one system, in other systems. As a consequence, the development of each such machine has very high labor content, periods of their development are too large, it is not possible to use a variety of problem-solving models within one system;
- developed knowledge processing machines are not flexible, i.e. it is very difficult or even not possible to supplement developed machine with new components or make changes in already existing components. As the result, the support of such machines is very labour-intensive, what leads to their rapid obsolescence;
- the level of professional requirements for knowledge processing machines developers is very high;
- attempts to unite a large number of developers in the groups are not efficient enough due to the lack of hierarchy in the developed machines and, as a result, in groups of developers. Difficulties in the coordination of actions lead to additional overheads.

The consequence of these problems is the relatively high labor intensity of the knowledge-based systems development and maintenance, and as a result - their high cost.

In order to speak about the relevance of these problems, it is necessary to specify the criteria for analysis and comparison of knowledge processing machines, as well as to analyze the existing approaches to the development of such machines.

C. Analysis of Existing Approaches to Specified Problems Solution

We have specified the following basic criteria for analysis and comparison of knowledge processing machines:

- **flexibility and expandability**. This criterion shows how difficult it is and whether it is possible to make additions or changes in the already developed machine;
- **portability and platform independence**, i.e. the evaluation of the complexity of the developed knowledge processing machine model transfer to another platform of such models interpretation, for example, change the knowledge storage to another, etc.
- **performance**, that is the actual speed of operation of such machines and necessary resources, such as memory amount. Performance of knowledge processing machines can be enhanced through the use of models and methods of distributed and parallel knowledge processing;
- **class of problems, which the machine is able to solve**. It is clear that the comparison of knowledge processing machines, classes of solved problems for which differ significantly, by other specified criteria has no sense.

Further, because this paper is not devoted to the design of a single knowledge processing machine, and about the models and means, providing the development of such machines, it is necessary to specify the requirements for such kind of models and means:

- the ability to consider each designed model on different levels of the hierarchy;
- existence of the libraries of compatible components of knowledge processing machines of various levels;
- support of the activity coordination of developers' of different professionalism and responsibility levels (due to the hierarchy too);
- availability of automation of changes in already developed machines, existence of verification tools and their components, as well as means of automatic correction of identified deficiencies;
- providing of developed machines platform independence, which significantly increases the duration of their life cycle. This assumes not only already became traditional platform independence of programs, but also platform independence of components of whole knowledge processing machines, their specifications, etc.
- flexibility of the proposed models and means, which implies the ability to make changes in the already developed components and machines, and simplifies staff turnover due to the high level of automation of documentation means and high requirements level for the documentation of each component, as well as the availability of means of developers' information support and/or training;
- availability within the technology of means, fixing the current state of development object, as well as its development plans and the changes history;

Thus, for the development of universal models and means of knowledge processing machines development, which satisfy the above requirements, it is necessary to provide:

- the ability of integrating and coordinated functioning of all components of the knowledge processing machine within the same system;
- the ability to integrate different approaches within each of the components, for example, different approaches to problems solution. In this case, the class of problems, solved by the system, can be significantly expanded, for example, through the use of reliable and probabilistic inference, depending on the given task.
- the ability of knowledge processing machine supplementing with new components without significant overheads;
- the ability of parallel asynchronous solution of various problems within a single knowledge processing machine. Concrete model of parallelism used in the solution of a problem is largely determined by the problem itself. Thus, the technology must provide the ability to implement any parallelism model;

Currently in the development of knowledge-based systems the W3C standards are widely used. In particular, the OWL 2 [14] standard is used for semantic knowledge representation and RDF [15] standard is used for representing knowledge in form of semantic networks.

To access the data, presented in RDF models the SPARQL protocol and the same name language are used [17]. As the next step in relation to the SPARQL language the declarative query language Cypher can be regarded, which was designed by the creators of Neo4j storage [2].

Considered SPARQL and Cypher languages provide only access to stored knowledge, knowledge processing itself is carried out at the level of application, which is working with a repository of knowledge.

There are a lot of implementations of the so-called semantic reasoners, performing logical inference on the base of ontology represented in OWL format, as well as tools for creating and editing of ontologies. A full list of such tools, recognized by the W3C consortium, can be found at [13]. As seen from the table on it, the most of represented tools are capable to perform only the direct inference on the basis of the relations, described in the ontology.

Thus, it can be concluded that currently within the consortium were developed efficient means of knowledge representation, knowledge access and logical inference mechanisms, based on the ontologies, represented in this form. However, there is no common methodology and technology of knowledge processing machine design that causes a large amount of duplication and significantly increases the complexity of application development based on this representation. Reusing of solutions applicable to one system is not possible in most cases for the same reason.

Among integrated approaches to knowledge processing machines development the IACPaaS project can be considered [32], which is actively developed in present time. The aim of this project is to develop cloud platform to build on its base intelligent services for various purposes.

Integrated approaches to the knowledge processing machines design for intelligent systems of different classes, are actively developed in Novosibirsk city [39].

Problems of integration of different solutions, including related to the knowledge processing are also considered in [48].

Component design of knowledge-based intelligent systems is the subject of [23] article, in which the necessity of accumulation and reuse of the various components of intelligent systems is justified.

Analysis of the described approaches shows that the problems formulated in this paper are not solved completely in any of these approaches. This is largely due to the lack of a unified basis for the formal representation of all kinds of knowledge, including various types of programs, the lack of strict principles of the knowledge processing machine design and means for support of the developers of these machines and their components.

D. The Proposed Approach

Within this article we propose to use an ontological approach to solve given problems, in this case - the ontological approach to the design of knowledge processing machines. In general, the ontological approach involves the development of (1) the ontological model of the class of designed artifacts, (2) the ontology of design, describing the activities aimed at the artifacts development and methodology of development and (3) the ontological model of means to support the development of artifacts of given class throughout all stages of the artifact lifecycle.

Thus, to solve the problems in sphere of the knowledge processing machines development described above we need to develop:

- unified ontological model of the knowledge processing machine, having the properties of flexibility, modularity, platform independence, and allows to implement on the its basis any of existing models and methods of knowledge processing, including parallel and asynchronous;
- ontology of knowledge processing machines design, that is based on the specified model and includes a description of the methodology of design and formal typology of such machines' developers' actions;
- ontological model of the system of knowledge processing machines design support, built on the basis of specified model, and designed by the described methodology.

E. Tasks to be Resolved for Proposed Approach Implementation

There several tasks to be resolved for proposed approach implementation:

- to develop a model of the knowledge processing machine based on the system of ontologies which, firstly, will provide the flexibility of these machines, and secondly, would make the complex knowledge processing machines more productive due to parallel execution of certain processes, thirdly, will provide platform independence of developed machines;
- to develop an ontology, describing the activities of various subjects in the computer system memory, including the description of actions and tasks;

- to develop on the basis of this ontology means to coordinate the activity of the various components of knowledge processing machines;
- to develop a universal basic programming language, which does not depend on the interpretation platform and allows it to build on the its basis higher-level languages and their interpreters. To do this, the specified language must be built on the basis of the above-mentioned ontology that describes the activities of the different subjects. This will provide the independence of programs building principles from the programming language level, which is determined primarily by the operators' typology;
- to develop and implement an ontological model of this programming language interpreter;
- to develop an ontology of knowledge processing machines design, including the formal description of such machines developers' activities;
- to develop ontological model of the means of knowledge processing machines design support, built on the basis of specified model, and designed by the described methodology.

As a formal basis for the development of all specified models we will use ***SC-code*** - a format of unified knowledge representation in the form of homogeneous semantic networks with set-theoretic interpretation, used in ***OSTIS Technology*** [29]. Nodes of such a semantic network were named ***sc-nodes***, connections - ***sc-connectors*** (***sc-arcs***, ***sc-edges***). Within this technology is defined the concept of an ***ontology*** as ***subject domain*** specification [34], their typology is released. In this regard, further we will not talk about the formation of ontologies that describe a set of concepts, we will talk about the formation of the subject domains and its specifications, implying that this process includes the formation of all the necessary ontologies.

Computer systems, based on OSTIS Technology were named ***ostis-systems***, due to that we will talk about the ***knowledge processing machines of ostis-systems***.

As a basic programming language the ***SCP Language***, described in the same paper, will be used. Programs of that language are also stored in the form of SC-code constructions. Thus, the task of the basic programming language development is reduced to clarification of the SCP Language evolving it to meet additional requirements discussed above. The use of this language will provide platform independence for developed knowledge processing programs. In addition, the SCP Language also has a number of advantages, which will be discussed in details below.

Model of knowledge processing machine will be build on the basis of multi-agent approach [47], [31], also it is assumed that the interaction of the agents will be performed exclusively by means of semantic memory, which stores the SC-code constructions (***sc-memory***). Such an approach would provide the flexibility and modularity of developed machines, as well as provide the ability of parallel execution of different knowledge processing processes. In addition, in the case of implementation of agent programs on platform-dependent level (not in the SCP Language), such an approach would eliminate the need of the implementation of direct agent communication mechanisms, which have to be implemented using various programming languages.

The effectiveness of multi-agent systems is justified by the use of such systems in various fields [10]. Currently, there are a lot of agent-based modeling environments. A detailed review of these environments is given in the works [12], [11].

In order to build a strict ontological model of multi-agent system, which can be used as a basis for knowledge processing machines design, it is necessary to specify a model of each component included in its composition, namely:

- **model of agent itself**, which is a member of the multi-agent system;
- **model of the environment** in which agents are, at the events in which they react, and with which they can act [20].;
- **model of the agents' communication**, and in particular, the model of agents' activity coordination and resolve conflicts;

A formal model of agents' communication was proposed in [45], but for the application of this model within the proposed approach it is necessary to specify each of its components:

- **principles of the messages exchange between agents**, i.e. the way in which these messages are sent from agent to agent. The proposed approach to messaging uses sc-memory for messaging. This approach can be considered as a development of the idea of «blackboard», proposed in [9].;
- **typology, semantics and pragmatics of such messages**, i.e. the sense of the transferred information and the purpose of such an interaction. Attempts to clarify the mentioned concepts carried out in the ACL and KQML standards [1], [4].;
- **principles of agents' activity coordination**. Several approaches to coordination were considered in [6], [16], [3].

In addition, this work uses several ideas related to improving of the data and knowledge processing efficiency, considered in the works [24], [41].

Let's enumerate those subject domains, development of which is necessary to resolve tasks given above:

- ***Subject domain of actions and tasks***
- ***Subject domain of actions executed in semantic memory of ostis-system (actions in sc-memory)***
- ***Subject domain of agents, which work in semantic memory of ostis-system (sc-agents)***
- ***Subject domain of actions and actions specifications of basic knowledge processing machine (Abstract scp-machine)***
- ***Subject domain of abstract scp-machine agents and corresponding microprograms***
- ***Subject domain of knowledge processing machines developers' actions***
- ***Subject domain of incorrectness in scp-programs***

Each of the described subject domains will be considered in details below. In addition, all subject domains and corresponding ontologies, presented in this paper, were described with SC-code and included in the relevant sections of the IMS Metasystem knowledge base [7].

II. Unified Ontology-based Models of Knowledge Processing

A. Unification of Formal Means of Description of the Various Subjects' Activity in Semantic Memory

In order to be able to formally describe the transformations, performed by knowledge processing machine, we have designed ***Subject domain of actions and tasks***, and the corresponding integrated ***Ontology of actions and tasks***, as well as all ontologies of particular type. Within that subject domain such general concepts are researched as an action, a subject, an object of action, a task and its solution, etc. Further these concepts are used to develop a formal means of agents' coordination in the shared memory, as well as the programming language, oriented on the semantic networks processing and underlying the proposed approach.

Given subject domain and its relation to other works on similar topics were presented in [50], so we will take a closer look only on a few basic concepts which are studied in that subject domain and provide the basis for some of the solutions presented below.

We consider the *class of actions* concept and particular types of that concept:

class of actions
= set of actions, similar in one way or another
<= family of subsets:*
action
<= partitioning:*
{
• *class of autonomous actions*
• *class of non-autonomous actions*
}

The autonomy of an action is determined regardless of exactly how the execution of concrete actions that belong to this class is performed.

Every action, which is member of the ***class of autonomous action*** is executed regardless of whether the specified action is a part of the decomposition of a more general action. When such an action is executed the fact that this action precedes any other actions or follow them (that is specified by the relation *actions sequence**) must not be taken into account.

If any of these conditions is not succeeded, the autonomy of the action is broken too.

Thus, autonomous action is semantically consistent act of transformation, executed by some subject, including, for example, in semantic memory.

To each action can be assigned some ***subject*** that executes the action. In relation to ostis-system we can consider concepts of the internal and external subject, and the system itself is considered as a subject of some activity too.

For the detailing of some action execution process we have introduced such relations as *action decomposition**, *sub-action**, *action sequence** and others.

It is important to note that the use of the proposed formal means of the different subjects' activity description at different levels will not only provide the universality and the "clarity" of this description due to the use of the most general concepts, but also the provide possibility of implementing any parallelism model at any level, from parallel execution of operators within the same program, up to agents' communication entire groups in shared semantic memory. The possibility of implementation of a particular model of parallelism in this case is determined only by the characteristics of the problem being solved.

Along with the classes of actions themselves within the given subject domain are also studied various classes of action specifications (*semantic neighborhoods* [34]), such as a *task*, *question*, *plan*, *program* and *solution* (*protocol*).

A ***task*** is considered as formal description of the conditions of a task or problem, that is, in fact, a formal specification of an action, aimed at the solution of this task or problem, which is sufficient to execute this action by any subject. Depending on the specific class of tasks, the internal state of the intelligent system or the required state of the environment can be described.

Each ***plan*** is a *semantic neighborhood*, *key sc-element'* of which is an action for which supposed details of its executing process are described. In the description of the ***plan*** a procedural and declarative approach can be used. In the case of a procedural approach appropriate action is specified by its decomposition into more specific sub-actions and the necessary specification of the sub-actions is given. In the case of the declarative approach plan specifies a set of sub-goals (e.g. using logical statements), the achievement of which is necessary to execute the appropriate action. In practice, both considered approach can be combined.

Each ***program*** is a generalized *plan* of the execution of actions that are members of appropriate class, that is, the *semantic neighborhood*, which *key sc-element'* is a *class of actions*, for elements of which details of their execution process are given.

B. Typology of Actions in Semantic Memory

Special attention should be paid to actions, executed in the semantic memory of a computer system (***sc-memory***). Actions of this class are studied within the ***Subject domain of actions executed in the abstract unified semantic memory*** and the corresponding ontology.

Every ***action in sc-memory*** is a sign of a transformation, performed by some subject (or subjects collective), and aimed on the sc-memory transformation. Specification of action after its execution may be included in the protocol of some task solution.

Transformation of the knowledge base state includes information search, involving (1) the localization of the response to the request in the knowledge base, an allocation of the response structure and (2) translation of the response into some external language.

Set of ***actions in sc-memory*** consists of signs of the actions of various kinds, the semantics of each of which depends on the specific context, i.e., orientation of the action on some specific objects and action membership to some particular class of actions.

It should be clearly separated:

- Each concrete ***action in sc-memory***, which is a sign of process that transforms sc-memory from one state to another;
- Each type of ***action in sc-memory***, which is a class of the same type (in some sense or another) of actions;
- sc-node, denoting a specific ***action in sc-memory***;
- sc-node, denoting a *structure* that is the description, specification or setting of the appropriate action;

Fragment of *set-theoretical ontology*, which specifies the concept of ***action in sc-memory*** using the SCn language [7]:

action in sc-memory
<= *inclusion*:*
process in sc-memory
=> *inclusion*:*
- *action in sc-memory, initiated by question*
- *action of ostis-system knowledge base editing*
- *action of ostis-system mode setting*
- *action of editing of the file stored in sc-memory*
- *action of interpretation of the program stored in sc-memory*

C. Unification of Model of Subject, Performing Transformations Within Shared Semantic Memory

The only kind of *subjects* that perform transformations in the *sc-memory*, we assume ***sc-agents***. For a formal definition of *sc-agent* concept we use the previously introduced concept of a *class of autonomous action*. So, an ***sc-agent*** is some entity that can perform *actions in the sc-memory*, which are members of some *particular class of autonomous action*. Within OSTIS Technology there was developed ***Subject domain of abstract sc-agents*** and the corresponding set of ontologies, specifying the notion of ***sc-agent*** and related concepts, including formal means of providing synchronization of actions executed by sc-agents in sc-memory.

Autonomy of actions, executed by each sc-agent presupposes that each sc-agent reacts on the corresponding class of situations and/or events in the sc-memory, and performs a certain transformation of *sc-text* (SC-code text) in the semantic neighborhood of the processed situation and/or event. Thus each sc-agent usually has no information about what other sc-agents are currently presented in the system and interacts with other sc-agents only by forming messages in shared sc-memory. That message can be, for example, a question, addressed to other sc-agents in the system (not known exactly to which of them), or an answer to the other sc-agents question (again, not known of which agents exactly).

It is important to note that the end user of the ostis-system in terms of knowledge processing is also considered as the sc-agent, forming messages in sc-memory by executing elementary actions, provided by user interface. In the same manner ostis-system interaction with other systems and the environment in general is performed. All information incomes the ostis-system and outcomes the ostis-system exclusively through the relevant interface sc-agents. In the proposed approach, direct access to the sc-memory of the end user or other external actors is not allowed.

Here are some advantages of the proposed approach to the organization of knowledge processing in sc-memory:

- because of the processing is performed by the agents that can exchange messages only through shared memory, adding a new agent or removing (deactivation) of one or more existing agents usually does not lead to changes in other agents because the agents does not exchange messages directly;
- agents often work in parallel and independently from each other, executing different actions in sc-memory; thus even a significant increase of the agents number in one system does not lead to reduction of its performance;
- agents specifications and, as will be shown below, their programs can be written in the same language with processed knowledge; this fact significantly reduces the list of special means, intended for the design of such agents and their collectives, and simplifies the whole system design process due to use of more versatile components.

Since it is supposed, that copies of the same *sc-agent* or functionally equivalent *sc-agents* may work in different ostis-system, being physically different sc-agents, it is rational to consider properties and typology of not sc-agents but the classes of functionally equivalent sc-agents, which we call ***abstract sc-agents***. Concepts of *sc-agent* and *abstract sc-agent* are the *maximal classes of studying objects* within the subject domain considered in this section.

Thus, the ***abstract sc-agent*** is a certain class of functionally equivalent *sc-agents*, various items of which can be implemented in different ways. Each ***abstract sc-agent*** has corresponding specification.

Typology of abstract sc-agents and means of their specification are considered in details in the work [49].

Let's consider the general typology of *abstract sc-agents*, which is a fragment of the set-theoretic ontology of the ***Subject domain of abstract sc-agents,*** presented in the SCn-code:

abstract sc-agent
<= *partitioning*:*
{
- *non-atomic abstract sc-agent*
- *atomic abstract sc-agent*

}
<= *partitioning*:*
{
- *internal abstract sc-agent*
- *effectoral abstract sc-agent*
- *receptoral abstract sc-agent*

}
<= *partitioning*:*
{
- *non scp-implementable abstract sc-agent*
  <= *partitioning**
  {
  - *effectoral abstract sc-agent*
  - *receptoral abstract sc-agent*
  - *abstract sc-agent of scp-program interpreting*

  }

• *scp-implementable abstract sc-agent*
}
<= *partitioning*:*
{
• *abstract sc-agent of scp-program interpreting*
• *program abstract sc-agent*
<= *partitioning*:*
{
• *effectoral abstract sc-agent*
• *receptoral abstract sc-agent*
• *scp-implementable program abstract sc-agent*
}
• *abstract sc-metaagent*
}
<= *partitioning*:*
{
• *platform-dependent abstract sc-agent*
=> *inclusion*:*
non scp-implementable abstract sc-agent
• *platform-independent abstract sc-agent*
<= *partitioning*:*
{
• *abstract sc-metaagent*
• *scp-implementable program abstract sc-agent*
}
}

D. The Formal Means of Describing the Actions and Action Specifications of Basic Machine of Unified Semantic Networks Processing

SCP Language is offered as the base language to write programs, which describe *sc-agent* activity within the *sc-memory*.

SCP Language is the graph procedural programming language, developed for efficient processing of homogeneous semantic networks with set-theoretic interpretation, coded using *SC-code*. *SCP Language* is the language of the parallel asynchronous programming.

The language of the submission of data for SCP Written (scp-programs) is the SC-Code. This fact allows us to ensure the independence of the programs implemented in the language of SCP on their interpretation of the platform, in connection with which the majority of the program sc-agents to be realized exactly in the specified language. However, it is obvious that in such a case, the system should also be implemented in a platform-dependent sc-level agents, carrying the interpretation SCP language programs. This restriction is taken into account in the typology of abstract sc-agents provided above.

The language of the representation of *SCP Language* programs (***scp-programs***) is the *SC-code*. This fact allows to ensure the independence of the programs implemented in the *SCP Language* from their interpretation platform, so that variant of the sc-agents program implementation is to be major within proposed approach. However, in that case, there should also be implemented platform-dependent sc-agents, carrying out the interpretation of SCP Language programs. This fact was taken into account in the typology of *abstract sc-agents* provided above.

SCP Language itself is based on the *SC-code*, so the scp-program may also be the processing object for other scp-programs, including first-mentioned program itself. Thus, SCP Language provides the possibility of reconfigurable software creating. However, to enable the reconfiguration of the program directly in the interpretation process it is necessary to ensure on the SCP Language interpreter level (*Abstract scp-machine*) the uniqueness of each executable copy of the original program. This executable copy, generated on the base of *scp-program,* was named ***scp-process***. Adding sign of some *action in sc-memory* in the *scp-process* set guarantees that in the decomposition of that action there will be only signs of elementary actions (***scp-operators***), that can be interpreted by the implementation of *Abstract scp-machine*.

SCP Language is considered as an assembler for graphodynamic computer, which is oriented on the semantic networks processing and storing.

Actually development of SCP Language was not the objective of this work, because this language was already developed and described in details, for example, in [27].

Within this work we discuss the problem of the *SCP Language* adaptation to modern level of knowledge processing machines models and design tools and development of the ***Subject domain of actions and the action specifications of abstract scp-machine***, in which we study all the concepts related to the given language. The mentioned subject area is particular to the *Subject area of actions in sc-memory*, because each *scp-operator* is a sign of elementary action in *sc-memory*.

Within the considered subject domain are allocated additional classes of *structures* (sc-constructions) [34] to work with which some scp-operators are oriented. These classes are *single element sc-construction, three element sc-construction, five element sc-construction, and sc-construction of non-standart type.*

The maximal class of the researched objects within specified subject domain is an ***scp-program***. Each ***scp-program*** is a *generalized structure*, describing one of the possible decomposition variants of the given class of *action in sc-memory*. In fact, each ***scp-program*** is a description of the sequence of elementary operations to be executed in the semantic memory, to execute a more complex action of a given class.

A particular case of *scp-program* is an ***agent scp-program***. *scp-programs* of this class are the programs of knowledge processing agents, and has fixed set of arguments.

An ***scp-process*** is an action *in sc-memory* that uniquely describes a particular act of scp- program execution with the given input values. If the scp-program describes an algorithm for a problem solving in a general way, the ***scp-process*** is a sign of a specific action that implements the algorithm for the specified input parameters.

In fact, ***scp-process*** is a unique copy, created on the base of the *scp-program* in which each generated *sc-constant* corresponds to the *sc-variable* in that *scp-program* [28].

Membership of some action in the *scp-process* set guarantees that in the decomposition of the action only elementary actions signs (*scp-operators*) will be attended, so that action can be interpreted by the implementation of *Abstract scp-machine*.

Each ***scp-operator*** (***scp-process*** operator) is a sign of an elementary *action sc-memory*. Arguments of scp-operator will

be called operands. The order of operands is indicated by the relevant role relations (*1'*, *2'*, *3'*, and so on). The type and the meaning of each operand is specified by various subclasses of the *scp-operand'* role relation. In general, the operand of the scp-operator could be any sc-element, including a sign of scp-program, including the program containing that operator itself.

Each ***scp-operator*** must have one or more operands, and there must be specified ***scp-operator*** (or more) to be executed next. The exception to this rule is an *scp-operator of the program execution termination*, which does not contain any operands and after which no scp-operators can be executed within this program.

Let's consider the of fragment set-theoretical ontology that describes the general typology of the scp-operators presented in the SCn-code:

scp-operator
<= *inclusion*:*
    action on sc-memory
=> *family of subsets*:*
    scp-operator atomic type
<= *partitioning*:*
    {
    • *scp-operator of generation*
    • *scp-operator of associative search*
    • *scp-operator of erasing*
    • *scp-operator of condition check*
    • *scp-operator of operand values operating*
    • *scp-operator of scp-processes management*
    • *scp-operator of sc-memory events management*
    • *scp-operator of files content processing*
    • *scp-operator of locks management*
    }

Membership of arguments to the scp-operator is specified using subclasses of role relation ***scp-operand'***. Fragment of set-theoretical ontology that describes the typology of the scp-operands roles presented in SCn-code:

scp-operand'
<= *inclusion*:*
    action argument'
∈ *non-basic concept*
∈ *role relation*
<= *partitioning*:*
    {
    • *scp-constant'*
    • *scp-variable'*
    }
<= *partitioning*:*
    {
    • *scp-operand with fixed value'*
    • *scp-operand with unassigned value'*
    }
<= *partitioning*:*
    {
    • *constant sc-element'*
    • *variable sc-element'*
    }
=> *inclusion*:*
    • *forming set'*
    • *erasing sc-element'*
    • *sc-element type' /* further is partitioned by the sc-elements typology*/*

Most of the advantages of the basic model of *SC-code* texts processing occur due to its following main features:

- texts of SCP Language programs are written using the same unified semantic networks as processed information;
- approach to the description and interpretation of the *scp-programs* is based on common principles of the various subjects' activities description, in particular, it is supposed to create a unique *scp-process* at each *scp-program* execution.

These advantages are:

- in one time in the shared memory can be executed several independent processes, wherein the processes corresponding to the same *scp-program* can be executed even on different servers in case of distributed realization of sc-model interpreter (*interpretation platform of computer system sc-models*).
- *SCP Language* allows to perform concurrent asynchronous calls of subroutines (create subprocesses within *scp-processes*), as well as to perform parallel scp-operator execution within one *scp-process*;
- As *scp-programs* are written using *SC-code*, transfer of *sc-agent,* implemented on the base of *SCP Language*, from one system to another reduces to simple transfer of the knowledge base fragment, without any additional operations that depend on the interpretation platform;
- the fact that the *sc-agents'* specifications and their programs can be written in the same language as the processed knowledge, significantly reduces the list of special funds intended for the knowledge processing machines design, and simplifies their development through the use of universal components;
- the fact that for the interpretation of scp-program the corresponding unique scp-process is created, allows to optimize the execution plan as much as possible (1) before its implementation and (2) even directly during the execution without the potential danger to break the general-purpose algorithm of the entire program. Moreover, such an approach to the programs design and interpretation allows to speak about the possibility of *self-reconfigurable programs* creating;

We cannot say that the idea of creating of a unique process, based on a program, for each act of its execution is a fundamentally new and is only implemented in the *SCP Language*. A similar approach is used in most modern systems, based on the von Neumann architecture and oriented on the work with traditional linear memory. In the discussed in this paper models and traditional systems both, this approach allows to speak about the possibility of the process reconfiguration during its execution, and, in the limit, about self-reconfigurable processes.

However, such an approach used in to semantic memory has a significant advantage, which consists in the such memory *associativity*. Indeed, in traditional memory access to the data is carried out exclusively by the address. In the case of reconfigurable software development address of each fragment in the neighborhood of which the change is performed, or at least the structure of the reconfigured process must be known to the process, which performs such a reconfiguration. This means, that a sense of a single piece of information in the traditional memory almost never can be resolved correctly without pre-known context. This fact leads to a great complexity of reconfigurable programs development, reducing of number of their application fields, and the high level of dependence of reconfiguration performing programs on the changes made by developers of software, in which this reconfiguration is performed. Using of the associative memory and the unified formal semantic basis for the description of any kind programs allows to eliminate these restrictions. Access to the elements within the associative memory is performed not on the basis of addresses, but by the connections between elements, and the number of key nodes, to which the binding is carried, is relatively small. Specification of each element within such memory is constructed by forming appropriate links, which can then be analyzed within the third-party process, i.e., the semantic of each element can be resolved by any process by analyzing the relations of considered elements with others. The number of such connections is not logically limited, so each element can be specified with the required level of detailing. In addition, the construction of *SCP Language* programs and higher-level languages programs bases on common formal means of description of the activity of all kinds of subjects, due to that the reconfiguration algorithms become more versatile because general structure of the program and semantics of transitions it in this case remains the same regardless of the language level.

E. Unification of Formal Means of Synchronization of Parallel Processes Execution in the Shared Memory

Because the only type of entities that perform the transformations in sc-memory are sc-agents, the general principles of the synchronization of their activity are also considered within the *Subject domain of abstract sc-agents*.

One of the concepts, researched in this subject domain is the concept of ***process in sc-memory***, which generalizes the concept of *action in sc-memory*. Concept of process in sc-memory memory is used to describe the principles of the synchronization using terminology in accordance with the literature on the subject, in which it is typically told about the *processes* in the memory of the computer system.

Before talking about the general principles of the various subjects' activity synchronization in the sc-memory, it is necessary to consider the typology of the processes in the sc-memory, which is a fragment of corresponding set-theoretical ontology. In that case we consider not a typology, based on the semantics of the performing transformations (such typology presented above for the class of *action in sc-memory*), but the typology associated with distinguished classes of processes, which use the same synchronization mechanisms.

Typology of *processes in sc-memory* presented in SCn-code:

process in sc-memory
<= *partitioning*:*
{
- *process in sc-memory, corresponding to platform-dependent sc-agent*
  <= *partitioning*:*
  {
  - *process in sc-memory, which corresponds to platform-dependent sc-agent and is not action of abstract scp-machine*
  - *action of abstract scp-machine*
    => *inclusion*:*
    action of scp-program interpretation

  }
- *scp-process*
  <= *partitioning*:*
  {
  - *scp-process, which is not scp-metaprocess*
  - *scp-metaprocess*

  }

}

In that work ***scp-metaprocess*** is the *process in sc-memory* that describes the activity of *sc-metaagent*, which is implemented with *SCP Language*.

To synchronize the execution of *processes in sc-memory*, the proposed approach uses a ***lock*** mechanism. The ***lock**** relation links the sign of *action in sc-memory* with signs of situational *structures* that contain sc-elements, locked for the duration of that action or any part of this period. Each such structure is a member of some *lock type*.

Three classes of *sc-agents* can be considered from the synchronization mechanisms point of view:

- *sc-agent of scp-program interpreting*
- *program sc-agent*
- *sc-metaagent*

Mechanism of locks, described in this section is used to synchronize the activity of the *program sc-agents*. This class includes all the agents, responsible for the tasks, assigned to a specific *ostis-system*, i.e. actually *sc-agents* of this class provide ostis-system functionality.

The task of *sc-agents of scp-program interpreting* is the maintenance of all described coordination rules (at the level of *SCP Language* programs interpretation platform). Principles of synchronization of agents of this class are more trivial than in the case of *program sc-agents*.

The task of *sc-metaagents* is conflict resolution and optimization of *program sc-agents* activity. Agents of this class operate on a higher level, and to synchronize their activities there are several special mechanisms, that will be discussed below.

To use the terminology, which is common in the literature, we will say that the process performs a transformation within sc-memory (e.g. deleting or generation of sc-element, setting

or removing *lock*), meaning that the corresponding transformation is performed by some sc-agent and is a part of *action in sc-memory* (i.e. *process in sc-memory* that describes the transformations, performed in the sc-memory by the active subject) to which mentioned sc-agent is linked by the *executor** relation.

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

- *full lock*
- *change lock*
- *deleting lock*

Each *structure,* which is ***full lock,*** contains sc-elements, viewing and editing (deleting or adding of incident sc-connectors, deleting sc-elements themselves, content change in the case of file) of which is prohibited for all *sc-agents*, except sc-agent, performing appropriate *action in sc-memory*, which is linked with the structure with *lock** relation. In fact, *sc-elements* falling within the ***full lock***, corresponding to a certain *process in sc-memory*, are temporarily absent for other processes in the current state of memory.

In order to exclude the possibility of implementing of *sc-agents*, which can make changes in the structure, describing the locks of other *sc-agents*, all of the elements of these structures, including structure sign itself (belonging to a set of *full lock*, and any other *lock type*) and links of *lock** relation linking structure and the correspond *action in sc-memory*, are added into the *full lock*, corresponding to this *action in sc-memory*. Thus, each *full lock* has corresponded membership loop, connecting it with the its sign itself.

Each structure which is member of ***change lock*** contains sc-elements, change (deleting or adding of incident sc-connectors, deleting of sc-elements themselves, content change in the case of file) of which is prohibited for all *sc-agents*, except *sc-agent*, executing the appropriate *action in sc-memory*. However, viewing (reading) of these elements is not prohibited for any *sc-agent*.

Basic principles of working with locks that do not depend on the *lock type*:

- at one time one process in sc-memory may correspond to only one lock of each type;
- at one time one process in sc-memory may correspond to only one lock installed on some concrete sc-element. Thus, the locked structures, corresponding to the same *process in sc-memory* do not intersect;
- at the end of the execution of any process in sc-memory all locks, set by it, are automatically removed;
- to improve the efficiency of the whole system, each process at any moment shall lock the minimal required set of sc-elements, removing the lock from each sc-element as it becomes possible (safe);

Each type of lock corresponds to a number of features and algorithms that cannot be discussed in details in article format and presented within the IMS Metasystem knowledge base [7].

If any process tries to set a lock of any type on any sc-element, already locked by another process, on the one hand, the lock cannot be set until another process releases the mentioned sc-element; on the other hand, in order to allow deadlock finding and resolving it is necessary to indicate the fact that a process wants to get access to the sc-element, blocked by any other process. A similar situation and the approach to its solution for the processes in the traditional memory described in [36].

Under the proposed approach, in order to be able to specify which processes are trying to lock an already locked sc-element, it is proposed, along with the *lock** relation use the *planned lock** relation, fully analogous to the *lock** relation. The process, which corresponds the *planned lock**, suspends execution until the already set lock will not be removed, after that the *planned lock** becomes a real lock, and the process continues execution in accordance with the general rules. In the case when several processes plan to install the lock on the same sc-element, also used the *lock priority** relation, linking the pairs of the *planned lock** relation.

In the case of deleting attempt of some sc-element, process can continue execution only when on that sc-element is not set (or planned) any lock by some other process.

To implement this possibility every process can be corresponded by set of *deleting sc-elements**. Sc-elements, appeared in that set, are available to processes, which have already set (or plan to set) the lock on these sc-elements previously (before attempting to delete them), and for all other processes these sc-elements are considered to be deleted already.

In some cases, in order to ensure synchronization, it is necessary to unite several elementary operations in sc-memory into a single atomic action (*transaction*) for which it is guaranteed that no third-party process will be able to read or modify the sc-elements involved in this action, until the action is complete.

In the proposed approach, *seven* transactions are considered. The implementation of them at the level of sc-models interpretation platform is necessary to ensure compliance with all the principles of working with the locks. A list of these transactions and their specifications can be found in IMS knowledge base [7].

Sometimes there can occur conflicts of two *processes in sc-memory*. This is possible, for example, when each of these processes is expected while the second process will unset lock form the desired sc-element, without unsetting the lock set by them on the same sc-element, access to which is required by the second process. Eliminating such a deadlock is impossible without the intervention of specialized sc-metaagent, which has the right to ignore the locks, set by other processes.

In general, the problem of specific deadlock can be resolved by performing the following steps by specialized sc-metaagent:

- rollback of several transactions performed by one of the processes involved in the deadlock to provide the second process access to the necessary sc-elements and possibility to continue execution;
- waiting for the second process until the terminating or until it unset all locks from sc-elements, access to which is necessary for the first process;
- re-execution within the first process of all cancelled operations and the continuation of its execution, but

with the changes in memory, made by the second process.

As mentioned above, the described synchronization rules are valid for the *software sc-agents*. For *sc-metaagents* all sc-elements, including those which describe locks, planned locks, etc., are fully equivalent to each other from the access point of view, i.e., any sc-metaagent has access to any sc-elements, even put into the *full lock* of some other process. This is necessary in order to sc-metaagents be able to identify and eliminate various problems such as the above-described problem of deadlock.

Thus, the problem of synchronization of sc-metaagents activity requires the introduction of additional rules.

Said problem is divided into two more particular:

- ensuring the synchronization of activity among the *sc-metaagents* themselves;
- ensuring the synchronization of activity among *sc-metaagents* and *program sc-agents*;

The first problem is solved by prohibiting parallel execution of *sc-metaagents*. Thus, at any given time within one *ostis-system* can exist only one process, corresponding to *sc-metaagent* and being *present entity* (executing at present moment).

The second problem is proposed to solve by introducing additional privileges for the *sc-metaagents* during the access to any sc-element.

The described mechanism of locks is applicable also in the case where some transformation in sc-memory is performed by the *ostis-system* user with means of the user interface *sc-agents*. In terms of knowledge processing, user is also considered as sc-agent, and therefore has the ability to lock the sc-elements with locks of different type.

In conclusion, it should also be noted that there may be a situation in which the execution of a process in the sc-memory is interrupted due to occurrence of any error. In this case, there is a possibility that the lock, set by that process will not be unset as long as it will be made by the sc-metaagent, which has discovered such a situation. However, at the level of sc-model this problem can be solved only partially, in cases where an error occured during the interpretation of scp-program, it shall be reported to sc-metaagent by the scp-interpreter (corresponding construction shall be formed in sc-memory). Cases where there is an error at the level of the scp-interpreter or sc-storage should be considered at the level of the *sc-models interpretation platform*.

For sc-agents, which programs are implemented using *SCP Language*, all the general principles of the organization of sc-agents and users interaction in shared memory of ostis-systems are applicable, but in that case there is a number of additional refinements, described in details in the IMS Metasystem knowledge base [7].

F. Unification of Ontological Ostis-systems Knowledge Processing Models

1) Ontological Model of Knowledge Processing Machine: ontological knowledge processing machine model, proposed in this paper, is built on the basis of the all principles described above.

Ontological knowledge processing machine model built by SC-code means (*sc-model of knowledge processing machine*), is considered as an ontological model of multi-agent system over shared memory. That memory acts as a communication environment for the agents. Agents included in such a multi-agent system, were named an *sc-agents*.

Using the earlier introduced terminology, we assume that the *sc-model of knowledge processing machine* is a *non-atomic abstract sc-agent*, which is the result of union of all the *abstract sc-agents,* that are part of a particular ostis-systems, into one. In other words, the *sc-model of knowledge processing machine* is a collective of sc-agents, consisted in a given ostis-system and perceived as a whole. For this reason, there is currently no need to introduce the *Subject domain of sc-models of knowledge processing machines*, as all such models are researched within the *Subject domain of abstract sc-agents*.

Thus, there are several basic levels of detailing of any knowledge processing machine:

- the level of knowledge processing machine itself;
- the level of non-atomic sc-agents that are part of the machine, including particular knowledge processing machines;
- the level of atomic sc-agents;
- the level of scp-programs, or programs implemented on the platform level;

Such a hierarchy of levels allows to provide, firstly, the possibility of component stage-by-stage design of the knowledge processing machine, and secondly - the possibility of designing, debugging and verification of machine components at different levels, regardless of other levels, which greatly simplifies the task of development of a knowledge processing machine, reducing overheads.

In addition, on each level there is a probability that some or all of the necessary components have already been implemented by anyone previously and can be reused in the developed machine. In details the methodology of machines component design will be discussed below.

2) The semantic typology of computer systems knowledge processing machines: classification of computer systems knowledge processing machines can be performed according to several criteria. Let's consider a several variants of such a classification, presented in SCn-code, and are part of the set-theoretic ontology of subject domain, presented earlier.

According to the type of the computer system:

Typology of *processes in sc-memory* presented in SCn-code:

knowledge processing machine
=> inclusion:*
- ...
  ∋ Knowledge processing machine of IMS
- *knowledge processing machine of auxiliary computer system*
  => inclusion:*

- *knowledge processing machine of computer system interface*
  => inclusion:*
    - *knowledge processing machine of computer system user interface*
    - *knowledge processing machine of interface between computer system and other computer systems*
    - *knowledge processing machine of interface between computer system and the environment*
- *knowledge processing machine of subsystem of some kind of components development support*
  => inclusion:*
    - *knowledge processing machine of subsystem of knowledge bases development support*
      => inclusion:*
      machine of knowledge base quality improvement
      => inclusion:*
        - *machine of knowledge base verification*
          => inclusion:*
            - *machine of search and elimination of incorrectness*
            - *machine of search and elimination of incompleteness*
        - *machine of knowledge base optimization*
        - *machine of information garbage detection and elimination*
    - *knowledge processing machine of subsystem of knowledge processing machines development support*
      => inclusion:*
        - *knowledge processing machine of subsystem of knowledge processing programs development support*
        - *knowledge processing machine of subsystem of knowledge processing agents development support*
    - *knowledge processing machine of subsystem of computer systems and their components development control*
- *knowledge processing machine of separated ostis-system*

According to the type of interpreted knowledge processing models:

knowledge processing machine
=> inclusion:*
- *machine of information search*
  => inclusion:*
    - *machine of search of information, which satisfies given specification*
    - *machine of search of information, which doesn't satisfy given specification*
    - *machine of detection that information, which satisfies given specification, is absent*
- *machine of problem solution using stored programs*
  => inclusion:*
    - *machine, providing neural models interpreting*
    - *machine, providing genetic algorithms interpreting*
    - *machine, providing imperative programs interpreting*
      => inclusion:*
        - *machine, providing procedural programs interpreting*
        - *machine, providing object-oriented programs interpreting*
    - *machine, providing declarative programs interpreting*
      => inclusion:*
        - *machine, providing logical programs interpreting*
        - *machine, providing functional programs interpreting*
- *machine of problem solution in conditions, when there isn't solution program*
  => inclusion:*
    - *machine of depth-first solution search method*
    - *machine of breadth-first solution search method*
    - *machine, implementing trial-and-error method*
    - *machine, implementing problem halving method*
    - *machine of problem solution using analogy*
    - *machine of reduction of problem condition to first-order predicate logic*
    - *machine of logical inference*
      => inclusion:*
        - *machine of deductive inference*
          => inclusion:*
            - *machine of direct deductive inference*
            - *machine of inverse deductive inference*
        - *machine of inductive inference*
        - *machine of abductive inference*
        - *machine of fuzzy inference*
        - *machine of temporal logical inference*

According to the processing object, the goal of the problem solution:

knowledge processing machine
=> inclusion:*
- *machine of actually formulated problems solution*
  => inclusion:*
    - *machine of given quantities values retrieval*
    - *machine of given logical sentence validating within given formal theory*
    - *machine of given problem solution method forming*
      => inclusion:*
        - *machine of given sentence proof forming within given formal theory*

- *machine of given task answer verification*
- *machine of given task solution method verification*

  => *inclusion*:*
  machine of verification of given sentence validity within given formal theory

- *machine, providing decision-making support*

  => *inclusion*:*
  machine, providing selection from given set of alternatives

- *machine of classification*

  => *inclusion*:*
  - *machine of classification of given entity within the given set of classes*
  - *machine of classification of given entities using given set of attributes*

- *machine of natural language texts synthesis*
- *machine of natural language texts analysis*

  => *inclusion*:*
  - *machine of natural language texts recognition*
  - *machine of natural language texts verification*

- *machine of signal synthesis*

  => *inclusion*:*
  machine of speech synthesis

- *machine of signal analysis*

  => *inclusion*:*
  machine of speech analysis

  => *inclusion*:*
  machine of speech recognition

- *machine of multimedia data processing*

  => *inclusion*:*
  machine of image analysis

  => *inclusion*:*
  machine of images recognition

G. Ontological Model of Interpretation Machine of Knowledge Processing Programs

On the basis of the considered ontological model of knowledge processing machine models of knowledge processing machines of a particular kind are built. One of the most important among them is the ontological model of the interpretation machine of the basic programming language for sc-text processing (i.e. *SCP Language*).

This model is considered as a set of sc-agents on which *Abstract scp-machine* is decomposed, i.e., *atomic sc-agents,* implemented on the platform-dependent level. All these agents are specified within the ***Subject domain of abstract scp-machine agents and corresponding microprograms***.

Decomposition of ***Abstract scp-machine***, presented in the SCn-code:

Abstract scp-machine
<= *abstract sc-agent decomposition*:*
{
- *Abstract sc-agent of scp-programs embedding in sc-memory*

  <= *abstract sc-agent decomposition*:*
  {
  - *Abstract sc-agent of scp-programs actual embedding in sc-memory*
  - *Abstract sc-agent of scp-programs preprocessing*

  }
- *Abstract sc-agent of scp-processes creation*
- *Abstract sc-agent of scp-operators interpretation*
- *Abstract sc-agent of scp-programs interpretation syncronization*
- *Abstract sc-agent of scp-processes destruction*
- *Abstract sc-agent of syncronization of events in sc-memory and sc-memory realization*

  <= *abstract sc-agent decomposition*:*
  {
  - *Abstract sc-agent of translation of sc-event formed specification into internal representation*
  - *Abstract sc-agent of processing of event, which initiates agent scp-program*

  }

}

To implement the proposed model of *Abstract scp-machine* web-oriented realization of sc-models interpretation platform was used, described in [40]. Detailed specification of the listed agents is presented in the IMS Metasystem.

III. The Ontological Model of Activity, Aimed at Knowledge Processing Machines Design

As mentioned above, all of the platform-independent components ostis-systems knowledge processing machines may be represented using the SC-code. In this case we are talking about an sc-agent specification, and the full texts of scp-programs, describing the algorithms of these agents.

Thus, the design of ostis-system knowledge processing machine is reduced to the design of such a system knowledge base fragment of special kind. In this regard, for the design of knowledge processing machines can be used all the existing means of knowledge bases development automation of the OSTIS Technology, considered, in particular, in [33], as well as the whole ontology of knowledge bases design.

Next, in is necessary to consider some aspects of the development, specific to the knowledge processing machines. These aspects include methodology of such machines design, discussed in detail in [51], suggesting the six main stages of the machine development, from the formation of the requirements, to debugging and implementation of designed components, as well as ***Subject domain of knowledge processing machines developers' actions*** and corresponding ***Ontology of knowledge processing machines design*** (or rather, their sc-models).

In this paper we consider the typology of classes of actions, aimed at the design and implementation of the ostis-system knowledge processing machine in according to mentioned methodology and are part of the ontology ***Ontology of knowledge processing machines design***.

It is important to note that according to the previously presented knowledge processing machine model of any ostis-system is an *abstract sc-agent*, and therefore the development of the machine is reduced to the development of such an agent.

action. develop ostis-system knowledge processing machine
= action. develop abstract sc-agent
<= partitioning:*
{
- *action. develop atomic abstract sc-agent*
  => inclusion:*
  action. develop platform-independent atomic abstract sc-agent
  => inclusion:*
  - *action. decompose platform-independent atomic abstract sc-agent into scp-programs*
  - *action. develop scp-program*
    => abstract sub-action:*
    - *action. specify scp-program*
    - *action. find in library an scp-program, which satisfies given specification*
    - *action. implement specified scp-program*
    - *action. verify scp-program*
    - *action. debug scp-program*
- *action. develop non-atomic abstract sc-agent*
  => inclusion:*
  - *agent. decompose non-atomic abstract sc-agent into particular sc-agents*
  - *action. develop abstract sc-agent*

}
=> abstract sub-action:*
- *action. specify abstract sc-agent*
- *action. find in library an sc-agent, which satisfies given specification*
- *action. verify sc-agent*
  <= partitioning:*
  {
  - *action. verify atomic sc-agent*
  - *action. verify non-atomic sc-agent*

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

  }

IV. The Ontological Model of System of Knowledge Processing Machines Design Support

Among the tasks of system of knowledge processing machines design support are information and technical support of such machines development, including ensuring the correct and effective implementation of all stages provided by correspondent methodology.

In the design of all components of ostis-systems similar principles are used, some of them are shown in details in [52]. One of the basic principles is the principle of use of already implemented components of various kinds, already available in the Library of OSTIS components, part of the IMS Metasystem. All systems, built on the OSTIS Technology, except of the Metasystem, we will be called *child ostis-systems*.

Thus, each child ostis-system and each system, supporting the design of some components class are closely linked with the *IMS Metasystem*, in particular, with the *Library of OSTIS components*, which is the part of the metasystem knowledge base. It is supposed, that the library is not physically transferred into the child ostis-system, but the signs of required libraries, and specification of these structures are part of the knowledge base of the design support system for given component class, as will be shown below.

The considered system of knowledge processing machines design support itself is ostis-system too, and has the appropriate structure. Thus, this system model includes an *sc-model of the knowledge base*, the *sc-model of knowledge processing machine* and *sc-model of user interface*. Due to this approach, in this system can be used, if necessary, all the agents, used in other systems, such as information search agents.

The considered system can actually be used in three ways:

- as a subsystem within the metasystem of support of intelligent systems design (IMS). This case supposes the debugging of required components within the metasystem and then transferring them to the child system;
- as an independent ostis-system, designed exclusively for the development and debugging of knowledge processing machines components. In this case, the designed components are debugged within that system, and then have to be transferred to the child ostis-system;
- as a subsystem within the child ostis-system. In this case, components debugging is carried out directly in the same system, in which they are supposed to use and an additional transfer is not required.

Regardless of the selected case of system using, developed components can be included in the Library of OSTIS components.

An important stage in the life cycle of any software system is its debugging. There are two fundamentally different level of knowledge processing machine debugging:

- debugging on the sc-agents level;
- debugging on the scp-programs level;

In the case of debugging at sc-agents level, act of each agent execution is considered as indivisible and cannot be interrupted. In this way debugging may be performed for atomic and non-atomic sc-agents both. Initiation of a particular sc-agent, including a member of the non-atomic, is performed by creating appropriate construction in sc-memory, thereby debugging can be carried out at different levels of agents detailing, up to atomic.

Debugging on the sc-agents level supposes the possibility of force setting unsetting of lock, enabling or disabling of any agents, etc., so agents of such debugging support system must be sc-metaagents. Due to the fact that the proposed in this paper model of interaction between agents uses the universal approach of agent interaction through shared memory, the considered system of agents' design support can be used as a basis for building of agents modeling systems with other

principles of communication, for example, the direct message exchange between agents.

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

Let's consider the formal model of system of knowledge processing machines design support. In the following part of this chapter, speaking about this subsystem or any of its components, we mean that we are talking about the sc-model of described entity, that is, about its ontological model, built with the SC-code means.

Subsystem of OSTIS knowledge processing machines development support, and accordingly, its sc-model is decomposed into two particular:

Subsystem of OSTIS knowledge processing machines development support
<= *basic decomposition*:*
{
- *Subsystem of knowledge processing agents development support*
- *Subsystem of OSTIS basic programming language programs development support*
}

These subsystems are decomposed in accordance with the general principles of ostis-systems architecture. Next, we will consider the main components of these subsystems.

Knowledge base of ***subsystem of knowledge processing machines development support*** includes in addition to *Kernel and kernel extensions of knowledge bases sc-models*, provided on the OSTIS Technology level, and description of the basic unified text processing model (*Abstract scp-machine*) a description of the key concepts, related to the scp-programs verification and debug. Thus, the subsystem includes all the necessary documentation for the developer of the various components of knowledge processing machines.

Now let's consider the basic concepts that are specific to the knowledge base of subsystem of *SCP Language* program design support.

In order to ensure the ability to debug scp-programs, within the ***Subject domain of actions and actions specifications of basic knowledge processing machine (Abstract scp-machine)*** there is a number of additional concepts. In particular, to enable the use of breakpoints within scp-programs appropriate relative and absolute concept are introduced.

Links of *quasybinary relation breakpoints** connect scp-program with a set of *sc-variables*, corresponding to scp-operators within this program. With each generating of scp-process, corresponding to this scp-program, all the *scp-operators*, corresponding to such variables will be added in a set of *breakpoint*, i.e., the execution of the scp-process will be interrupted when the each of these scp-operators will be achieved.

Using of this relation defines the breakpoints for all scp-processes generated on the base of a given scp-program. To specify a breakpoint within the single scp-process, the ***breakpoint*** set shall be used.

To enable scp-programs verification, ***Subject domain of incorrectness in scp-programs*** and the corresponding ontologies were developed.

A typology of such incorrectnesses:

incorrectness in scp-program
<= *inclusion*:*
incorrect structure
=> *inclusion*:*
error in scp-program
=> *inclusion*:*
- *unachievable scp-operator*
- *potentially infinite loop*

error in scp-program
<= *partitioning*:*
{
- *syntax error in scp-program*
- *semantic error in scp-program*
}
<= *partitioning*:*
{
- *error in scp-program on the program level*
- *error in scp-program on the parameters set level*
- *error in scp-program on the operators set level*
- *error in scp-program on the operator level*
- *error in scp-program on the operand level*
}

All of shown subject domains and their ontologies are part of the knowledge base of the system of knowledge processing machines design support. In addition, one of the most important part of the knowledge base of the system of knowledge processing machines design support is a library of reusable components of such machines.

Reusable component of OSTIS is some ostis-system component that can be used within other ostis-systems [52]. In this work the general typology of these components (the library structure) and the main principles to work with them are considered.

If ***reusable component of abstract knowledge processing sc-machines*** is *platform-dependent reusable component of OSTIS*, its integration is made in accordance with the instructions, provided by the developer and depending on the platform, as well as in case of any component of this kind. Otherwise, the integration process can be concretized depending on the subclass of components of given type.

Let's consider the structure of the ***Library of reusable components of abstract sc-machines***:

Library of reusable components of abstract sc-machines
= *reusable component of abstract knowledge processing sc-machines*
<= *partitioning*:*
{
- *Library of reusable abstract sc-machines*
- *Library of reusable atomic abstract sc-agents*

• *Library of reusable sc-text processing programs*
}

The ***reusable abstract sc-agent*** (both atomic and non-atomic, i.e. whole sc-machine) is the component, corresponding to some *abstract sc-agent* that may be used in other systems, possibly as a part of more complex *abstract non-atomic sc-agents*. Each ***reusable abstract sc-agent*** should contain all the information, necessary for the operation of the corresponding sc-agent in the child system.

After the ***reusable atomic abstract sc-agent*** has been copied to the child system, it is necessary to generate an *sc-node*, indicating a particular *sc-agent,* working in that system and which is member of the selected implementation of *abstract sc-agent*, and add it in a set of *active sc-agents* if necessary.

The structure of the ***Library of reusable atomic abstract sc-agents***:

Library of reusable atomic abstract sc-agents
= reusable atomic abstract sc-agent
<= partitioning:*
{
• *Library of information search sc-agents*
• *Library of sc-agents of knowledge integration into knowledge base*
• *Library of sc-agents of ontologies alignment*
• *Library of sc-agents of actually formulated tasks solution planning*
• *Library of logical inference sc-agents*
• *Library of sc-agents of information garbage collection*
• *Library of coordinate sc-agents*
• *Library of sc-agents of high level programming languages and correspondent interpreters*
• *Library of knowledge base verification sc-agents*
• *Library of knowledge base editing sc-agents*
• *Library of sc-agents of knowledge base developers activity automation*
}

The ***reusable sc-texts processing program*** is a component corresponding to a program, written on an arbitrary programming language, which is focused on the processing of structures that are stored in the memory of *ostis-system*. The priority in this case is the use of *scp-programs* because of their platform independence, except the development of some components of the interface when the full platform independence is not possible (for example, in the case of *effector sc-agents* and *receptor sc-agents*).

In turn, a ***reusable scp-program*** is a component, corresponding to some enough universal *scp-program*, which can be used as part of several *sc-agents*.

After the ***reusable scp-program*** was copied to the child system, it is necessary to add it in a set of *correct scp-programs* (correctness is verified if it enters the library of components within the *IMS*).

A. The Ontological Model of Tools of Knowledge Processing Machines Design Support

Let's consider the structure of the relevant subsystems' knowledge processing machines.

The structure of the knowledge processing machine of *Subsystem of knowledge processing agents development support*:

Knowledge processing machine of subsystem of knowledge processing agents development support
<= abstract sc-agent decomposition:*
{
• *Abstract sc-agent of sc-agents verification*
• *Abstract sc-metaagent of sc-agent collectives debugging*
}

Abstract sc-agent of sc-agents verification
<= abstract sc-agent decomposition:*
{
• *Abstract sc-agent of sc-agent specification verification*
• *Abstract sc-agent of verification of non-atomic sc-agent specification consistency to specifications of particular sc-agents in its composition*
}

Abstract sc-metaagent of sc-agent collectives debugging
<= abstract sc-agent decomposition:*
{
• *Abstract sc-metaagent of search all running processes, which correspond to given sc-agent*
• *Abstract sc-agent of given sc-agent initiating using given arguments*
• *Abstract sc-metaagent of given sc-agent activation*
• *Abstract sc-metaagent of given sc-agent deactivation*
• *Abstract sc-metaagent of setting of given type lock for given process on given sc-element*
• *Abstract sc-metaagent of removing of all given process locks*
• *Abstract sc-metaagent of removing of all locks from given sc-element*
}

Structure of knowledge processing machine of subsystem of OSTIS basic programming language programs development support:

Knowledge processing machine of subsystem of OSTIS basic programming language programs development support
<= abstract sc-agent decomposition:*
{
• *Abstract sc-agent of scp-programs verification*
• *Abstract sc-agent of scp-programs debugging*
}

Abstract sc-agent of scp-programs debugging
<= abstract sc-agent decomposition:*
{
• *Abstract sc-agent of given scp-program starting using given parameters set*

- *Abstract sc-agent of given scp-program starting using given parameters set in single-stepping mode*
- *Abstract sc-agent of search of all breakpoints within scp-program*
- *Abstract sc-agent of search of all breakpoints within scp-process*
- *Abstract sc-agent of adding the breakpoint into scp-program*
- *Abstract sc-agent of breakpoint removing from scp-program*
- *Abstract sc-agent of adding the breakpoint into scp-process*
- *Abstract sc-agent of breakpoint removing from scp-process*
- *Abstract sc-agent of scp-process execution proceeding on one step*
- *Abstract sc-agent of scp-process execution proceeding till the breakpoint or ending*
- *Abstract sc-agent of scp-process information displaying*
- *Abstract sc-agent of scp-operator information displaying*

}

B. The Ontological Model of User Interface of System of Knowledge Processing Machines Design Support

Since the objects of design for the described system are knowledge processing machines models, and particularly, knowledge processing agents and programs represented in *SC-code*, such a system can use the base means of external representation of SC-code texts, for example, SCn or SCg languages [7].

To visually simplify the process of verification and debugging of knowledge processing machine components, an approach is used which supposes, that only the minimum required set of sc-elements is displayed at any given time to the user of the system. For example, when one debug scp-process, it is enough to display only scp-operators and connections between them. If necessary, the user can manually request and view the specification of the necessary scp-operator at the program break moment. This approach underlies the algorithms of all the agents of the described system.

Thus, currently, the user interface of system of knowledge processing machines design support is represented by a set of commands that allow a user to initiate activity of the desired agent, which is part of the system. This set fully corresponds the set of agents discussed above and its detailed consideration in this paper is inappropriate.

V. Approbation of Models and Tools of Knowledge Processing Machines Design

Initial testing of the developed models and tools was carried out on the basis of the IMS Metasystem itself [7] [28]. In the process, the initial filling of the library of reusable sc-agents and knowledge processing programs was carried out.

In addition, the proposed solutions are used in the work of the students of the department of Intelligent Information Technologies of Belarusian State University of Informatics and Radioelectronics to develop knowledge processing machines of intelligent reference systems (IRS) in various subject domains.

Of particular interest are the prototypes of knowledge processing machines of the Euclidean geometry IRS and graph theory IRS. This is due, firstly, regarding the development level and the complexity of these prototypes, and secondly, the fact that these systems use fundamentally different approaches to problems solving.

As part of the Euclidean geometry IRS were implemented depth-first problem solution search strategy and logical inference engine, allowing to solve the problem in a few steps using logical statements (theorems, axioms), stored in the knowledge base.

In turn, within the graph theory ISS was implemented concept of programs package, which is based on a mechanism of reducing the problem to sub-problems, each of which is finally solved by executing a program stored in system memory for some input. This mechanism also allows to solve problems in several steps, i.e. solve such problems, for which there is no ready-made program, with which it would be possible to solve this problem.

In addition, the knowledge processing machine of each of the prototypes under consideration consists a set of search agents, many of which were taken from the *Library of reusable sc-agents*, and some specially implemented for the relevant system, as they are subject-dependent.

A. Knowledge Processing Machine of IMS

Currently, knowledge processing machine of IMS itself, excluding subsystem, includes a set of information search agents, implementing mechanisms of basic navigation within the knowledge base. All these sc-agents included in the correspondent libraries.

The structure of the knowledge processing machine in SCn-code:

Knowledge processing machine of IMS
<= abstract sc-agent decomposition:*
{
- *Abstract sc-agent of search of all outgoing constant positive permanent arcs of membership*
- *Abstract sc-agent of search of all outgoing constant positive permanent arcs of membership with role relations*
- *Abstract sc-agent of search of all ingoing constant positive permanent arcs of membership*
- *Abstract sc-agent of search of all ingoing constant positive permanent arcs of membership with role relations*
- *Abstract sc-agent of search of all identifiers of given element*
- *Abstract sc-agent of search of full semantic neighborhood for given object*
- *Abstract sc-agent of search of decomposition links for given sc-element*
- *Abstract sc-agent of search of all entities which are general for given entity*
- *Abstract sc-agent of search of all entities which are particular for given entity*

- *Abstract sc-agent of search of definition or explanation for given object*

}

B. Knowledge Processing Machine of Euclidian Geometry IRS

In addition to the mentioned standard set of basic agents of information search, for Euclidian geometry IRS additional search agents have been implemented and subsequently included in the *Library of information search sc-agents* of IMS, and one of the problem-solving models has been implemented too.

The list of information search agents, implemented within the Euclidean geometry IRS:

- *Abstract sc-agent of search of annotation for given section*
- *Abstract sc-agent of search of axioms for given ontology*
- *Abstract sc-agent of search of theorems for given ontology*
- *Abstract sc-agent of search of direct connections between two objects*
- *Abstract sc-agent of search of concepts, on the base of which given concept is defined*
- *Abstract sc-agent of search of definitional domain for given relation*
- *Abstract sc-agent of search of definition or explanation for given object*
- *Abstract sc-agent of search of examples for given concept*
- *Abstract sc-agent of search of formal notation for given statement sign*
- *Abstract sc-agent of search of illustrations for given object*
- *Abstract sc-agent of search of key sc-elements for given subject domain*
- *Abstract sc-agent of search of concepts, defined on the base of given*
- *Abstract sc-agent of search of construction by given pattern*
- *Abstract sc-agent of search of proof sc-text for given statement*
- *Abstract sc-agent of search of relations, defined on the given object*
- *Abstract sc-agent of search of problem condition and solution sc-text*
- *Abstract sc-agent of search of statements about given object*

As part of Euclidian geometry IRS also implemented a prototype of intelligent problem solver which, as well as some of its components, can be used in other systems. The structure of the solver:

Non-atomic abstract sc-agent of problem solution
<= abstract sc-agent decomposition:*

{

- *Abstract sc-agent of search of given quantity value*
- *Abstract sc-agent of statement validity check*
- *Abstract sc-agent of problem solving strategy application*
- *Abstract sc-agent of logical inference*
- *Non-atomic abstract sc-agent of mathematical expressions interpreting*
  <= abstract sc-agent decomposition:*
  {
  - *Abstract sc-agent of mathematical expressions calculating coordination*
  - *Abstract sc-agent of exponention, rooting and finding the logarithm*
  - *Abstract sc-agent of numbers and quantitites addition and substitution*
  - *Abstract sc-agent of numbers and quantitites multiplication and division*
  - *Abstract sc-agent of numbers and quantitites comparison*
  - *Abstract sc-agent of trigonometrical expressions evaluating*

  }

}

C. Knowledge Processing Machine of Graph Theory IRS

At the moment, for the graph theory IRS the following agents have been implemented (excluding agents, taken from the library):

Agents which answer general questions about the graph:

- *Abstract sc-agent of graph specification*
- *Abstract sc-agent of search of the graph characteristics*
- *Abstract sc-agent of search of the graph numeric characteristics*
- *Abstract sc-agent of search of sets, characterizing the graph*

Agents, forming sets characterizing the graph:

- *Abstract sc-agent of search of minimal spanning tree of the graph*
- *Abstract sc-agent of search of articulation points set of the graph*
- *Abstract sc-agent of search of bridges set of the graph*
- *Abstract sc-agent of search of deadends set of the graph*
- *Abstract sc-agent of search of anti-deadends set of the graph*

Agents to identify the type of the graph:

- *Abstract sc-agent of graph directivity check*
- *Abstract sc-agent of graph planarity check*
- *Abstract sc-agent of graph reflexivity check*
- *Abstract sc-agent of graph connectivity check*
- *Abstract sc-agent of graph symmetry check*
- *Abstract sc-agent of graph transitivity check*
- *Abstract sc-agent of graph cyclicity check*

Agents of graph numeric characteristics evaluation:

- *Abstract sc-agent of search of connected components of the graph*

Most of the agents above, corresponds to scp-program, which implements the basic algorithm of the agent, and has a specification that allows to resolve the possibility and feasibility of that program use in the process of solving a problem.

To enable the use of multiple programs or logic statements in the process of solving a problem, within the graph theory IRS, discussed above *Non-atomic abstract sc-agent of problem solving* was included in and modified. After the modifying, the agent became to be able to analyze not only the logical statements but the program specifications too, and, if necessary, initiate the implementation of these programs with the required input data. Unlike the original, the modified agent implements a strategy of problem solution search from the target (inverse inference), and tries to construct a sequence of programs and logical statements, use of which on existing input will lead to the desired result.

Actions, executed by modified agent has two arguments. The first argument is a sign of the *entity*, the characteristic of which it is necessary to find or calculate (for example, the sign of a concrete graph), the second - a sign of the class, corresponding to the characteristic described, and the second argument can be an absolute concept or relative. For example, if it is necessary need to check whether a given graph is acyclic, the second argument is a sign of the concept *acyclic graph*; if it is necessary to determine the diameter of a given graph, then the second argument will be the sign of the *diameter** relation.

VI. Conclusion.

In conclusion, we list the main advantages of the obtained results. The development of a universal model of knowledge processing machine, based on the system of subject domains and ontologies presented above, allows to unify different approaches to the knowledge processing, which in turn makes it possible to:

- on a basis of the proposed model, provide the implementation of any knowledge processing models and problems solution methods, including parallel and asynchronous;
- integrate, if necessary, different approaches to problems solution in a single system, and to ensure their simultaneous execution;
- consider any knowledge processing machine as a hierarchical system, which significantly increases the efficiency of the processes of its design, implementation and debugging;
- ensure platform independence of implemented knowledge processing machines and their components;
- through the use of multi-agent approach and unification of principles of agents distinguishing provide the flexibility of implemented knowledge processing machines;
- due to the universal and unified representation of the processed knowledge generalize existing approaches to solution of certain classes of problems, letting to turn the problems formulating way from procedural into declarative, thus providing greater flexibility of implemented solutions;

Development of standardized methods of knowledge processing machines design as ontology of design can significantly reduce the number of situations in which a similar solution being implemented by different developers are incompatible, which leads to the need for duplication of similar solutions in different systems. Furthermore, this approach makes it possible to use the already implemented components in the design of new machines, thus significantly reducing the overheads of their implementation.

Develop and implemented model of system of knowledge processing machines design support will provide automation of the activity of these machines developers and reduce the overhead of their verification and debugging. Moreover, this model is designed with all of the above mentioned approaches, which allows to provide the flexibility of that system itself.

This work was supported by BRFFR-RFFR (Ф15РМ-074, Ф16Р-102).

Список литературы

[1] (2016, Apr.) FIPA ACL Message Structure Specification [Online]. – Available: http://www.fipa.org/specs/fipa00061/SC00061G.html.

[2] (2016, Apr.) Cypher Query Language [Online]. – Available: http://neo4j.com/docs/stable/cypher-query-lang.html.

[3] C. B. Excelente-Toledo, N. R. Jennings. The Dynamic Selection of Coordination Mechanisms. Autonomous Agents and Multi-Agent Systems Vol. 9, Issue 1, February 2004, p. 55-85

[4] T. Finin, R. Fritzson, D. McKay, R. McEntire. KQML as an agent communication language // Proceedings of the third international conference on Information and knowledge management - CIKM '94. — 1994. — P. 456.

[5] Gruber T.R. A Translational Approach to Portable Ontologies // Knowledge Acquisition. – 1993. – V. 5. – No. 2. – P. 199 – 220.

[6] Hartung R.L., A. Hakansson A. Using Meta-agents to Reason with Multiple Ontologies KES-AMSTA 2008. Pp. 261-270.

[7] (2016, Apr.) The IMS.OSTIS website [Online]. – Available: http://www.ims.ostis.net.

[8] Jackson, P. Introduction to Expert Systems / P. Jackson // Boston: Addison-Wesley, 1998.

[9] Jagannathan V., Dodhiawala K., Baum L. Blackboard Architectures and Applications. — N.Y.: Academic Press, 1989.

[10] (2016, Apr.) Autonomous Agents and Multi-Agent Systems[Online]. – Available: http://www.springer.com/computer/ai/journal/10458.

[11] Macal C.M., North M.J. Tutorial on Agent-based Modeling and Simulation // Proceedings of the 2005 Winter Simulation Conference. WSC'05. 2005.

[12] Marietto M., David N., Sichman J.S., Coelho H. Requirements Analysis of Agent-Based Simulaton Platforms: State of the Art and New Prospects // Multi-Agent-Based Simulation II, Vol. 2581 of LNAI series, Springer-Verlag. 2002.

[13] (2016, Apr.) OWL Implementations[Online]. – Available: https://www.w3.org/2001/sw/wiki/OWL/Implementations/

[14] (2016, Apr.) OWL 2 Web Ontology Language [Online]. – Available: http://www.w3.org/TR/owl2-overview.

[15] (2016, Apr.) RDF 1.1 Concepts and Abstract Syntax [Online]. – Available: http:// http://www.w3.org/TR/rdf11-concepts/.

[16] M. Sims, D. Corkill, V. Lesser. Automated organization design for multi-agent systems. Autonomous Agents and Multi-Agent Systems Vol. 16, Issue 2, June 2008, p. 151-185

[17] (2016, Apr.) SPARQL 1.1 Overview [Online]. – Available: https://www.w3.org/TR/sparql11-overview/.

[18] (2016, Apr.) World Wide Web Consortium [Online]. – Available: http:// http://www.w3.org.

[19] Waterman D. A. Guide to expert systems / D. A. Waterman // Boston: Addison-Wesley, 1985.

[20] D. Weyns, A. Omicini, J. Odell. Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems (Special Issue on Environments for Multi-agent Systems) Vol. 14, Issue 1, February 2007, p. 5-30

[21] Батыршин И.З. Основные операции нечеткой логики и их обобщения / И.З. Батыршин; – Казань: Отечество, 2001.

[22] Беркинблит М. Б. Нейронные сети. — М.: МИРОС и ВЗМШ РАО, 1993. — 96 с.

[23] Борисов, А.Н., Построение интеллектуальных систем, основанных на знаниях, с повторным использованием компонентов / А.Н. Борисов // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2014): материалы IV Междунар.научн.-техн.конф. – Мн.: БГУИР, 2014

[24] Борщев В.Б. Вегетативная машина // Программирование. - 1989. - N 5. - с. 16-28.

[25] Вагин В.Н. Достоверный и правдоподобный вывод в интеллектуальных системах / Вагин В.Н.[и др.]; – М. : ФИЗМАТЛИТ, 2008.

[26] Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие. — 2-е изд. — М: Физматлит, 2006. — С. 320.

[27] Голенков В.В., Гулякина Н.А., Елисеева О.Е. Описание языка SCP (Материалы по математическому обеспечению ЭВМ). - Минск: Ин-т техн. кибернетики АН Беларуси, 1995. - 152 с.

[28] Голенков В.В., Гулякина Н.А. Проект открытой семантической технологии компонентного проектирования интеллектуальных систем. Часть 1: Принципы создания. / В. В. Голенков, Н.А. Гулякина // Онтология проектирования. – 2014. – №1. с.42-64

[29] Голенков В.В., Гулякина Н.А. Проект открытой семантической технологии компонентного проектирования интеллектуальных систем. Часть 2: Унифицированные модели проектирования. / В. В. Голенков, Н.А. Гулякина // Онтология проектирования. – 2014. – №4. с.34-53

[30] Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. — Новосибирск: Наука, 1996. — 276 с.

[31] Городецкий, В.И. Многоагентные системы (обзор)/В.И. Городецкий, М.С. Грушинский, А.В. Хабалов// Новости искусственного интеллекта. - 1998. - № 2. - С.64-116

[32] Грибова, В.В. Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. Часть 1. Разработка базы знаний и решателя задач / Грибова В.В.[и др.] // Программная инженерия. – №12, 2015, с. 3 - 11.

[33] Давыденко, И.Т. Семантическая модель коллективного проектирования баз знаний / И.Т. Давыденко // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2016): материалы VI Междунар.научн.-техн.конф. – Мн.: БГУИР, 2016.

[34] Давыденко, И.Т. Средства структуризации семантических моделей баз знаний / И.Т. Давыденко, Н.В. Гракова, Е.С. Сергиенко, А.В. Федотова // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2016): материалы VI Междунар.научн.-техн.конф. – Мн.: БГУИР, 2016.

[35] Деменков Н.П. Нечеткое управление в технических системах / Н.П. Деменков; – М : Изд. им. Баумана, 2005.

[36] Дийкстра Э. Взаимодействие последовательных процессов / Э. Дийкстра // Языки программирования. - М.: Мир, 1972. - с. 9-86.

[37] Емельянов В. В., Курейчик В. В., Курейчик В. М. Теория и практика эволюционного моделирования. — М: Физматлит, 2003. — С. 432.

[38] Еремеев А.П. Построение решающих функций на базе тернарной логики в системах принятия решений в условиях неопределенности // А.П. Еремеев Известия академии наук. Теория и системы управления, 1997. №5.

[39] Загорулько Г.Б., Загорулько Ю.А. Подход к интеграции разнородных методов поддержки принятия решений для сложных задач / Г.Б. Загорулько, Ю.А. Загорулько // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2013): материалы Междунар. научн-техн. конф. (Минск, 16–18 февраля 2013 г.); – Минск: БГУИР, 2013.

[40] Корончик, Д.Н. Реализация хранилища унифицированных семантических сетей / Д.Н. Корончик // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2013): материалы III Междунар.научн.-техн.конф. Мн.: БГУИР, 2013 – С.125-129

[41] Котов В.Е., Нариньяни А.С. Асинхронные вычислительные процессы над общей памятью // Кибернетика. - 1966. - N 3. - с. 64-71.

[42] Кулик, Б. А. Логика естественных рассуждений / Б. А. Кулик; - СПб.: Изд-во «Невский диалект», 2001.

[43] Пойа Д. Математика и правдоподобные рассуждения / Пойа Д.; – М.: Изд-во «НАУКА», 1975.

[44] Поспелов Д.А. Моделирование рассуждений. Опыт анализа мыслительных актов / Д.А.Поспелов; – М. :Изд-во «Радио и связь», 1989.

[45] Рыбина Г.В. Основы построения интеллектуальных систем. — М.: Финансы и статистика; ИНФРА-М, 2010. — 432 с.

[46] Справочник. Искусственный интеллект. Книга 1: системы общения и экспертные системы // Под ред. Э. В. Попова. — М.: «Радио и связь», 1990

[47] Тарасов В.Б. От многоагентных систем к интеллектуальным организациям / В.Б. Тарасов // Эдиториал УРСС, 2002. 352 с.

[48] Филиппов А.А. Единая онтологическая платформа интеллектуального анализа данных / А.А. Филиппов, В.С. Мошкин, Д.О. Шалаев, Н.Г. Ярушкина // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2016): материалы VI Междунар.научн.-техн.конф. – Мн.: БГУИР, 2016.

[49] Шункевич Д.В. Принципы построения машин обработки знаний интеллектуальных систем на основе семантических сетей. / Д.В. Шункевич // Электроника-инфо. — 2014. - № 3.

[50] Шункевич, Д.В. Формальное семантическое описание целенаправленной деятельности различного вида субъектов / Д.В. Шункевич, А.В. Губаревич, М.Н. Святкина, О.Л. Моросин // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2016): материалы VI Междунар.научн.-техн.конф. – Мн.: БГУИР, 2016.

[51] Шункевич, Д.В. Унифицированная семантическая модель процесса проектирования машин обработки знаний/ Д.В. Шункевич // Информационные технологии и системы (ITS-2016): материалы Междунар.научн.конф. – Мн.: БГУИР, 2016.

[52] Шункевич, Д.В. Средства поддержки компонентного проектирования систем, управляемых знаниями / Д.В. Шункевич, И.Т. Давыденко, Д.Н. Корончик, И.И. Жуков, А.В. Паркалов // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2015): материалы VI Междунар.научн.-техн.конф. – Мн.: БГУИР, 2015.

ОНТОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ МАШИН ОБРАБОТКИ ЗНАНИЙ

Шункевич Д.В.

Работа посвящена разработке технологии проектирования машин обработки знаний интеллектуальных систем, в основе которой лежат онтологическая модель самой машины и онтологическая модель процесса проектирования.

В настоящее время все более актуальным становится использование интеллектуальных систем в самых различных областях человеческой деятельности, в особенности в тех ситуациях, где нахождение человека может быть опасным или приводить к возникновению ошибок, обусловленных так называемым человеческим фактором. В частности, одним из наиболее перспективных направлений в данной области является разработка систем, основанных на знаниях. В свою очередь в качестве основы для формального представления знаний в такого рода системах широко используются онтологии.

Одним из ключевых компонентов каждой такой системы является машина обработки знаний, обеспечивающая возможность решать различные задачи, связанные как с непосредственно основным функционалом системы (машина информационного поиска и интеллектуальный решатель задач), так и с обеспечением корректности работы самой такой системы (машина сборки информационного мусора, повышения качества базы знаний), а также с обеспечением автоматизации развития самой этой системы. Следует отметить, что задачи, решаемые некоторыми компонентами машины обработки знаний, не всегда явно сформулированы. К таким задачам можно отнести, например, выявление и сборку информационного мусора, оптимизацию базы знаний и т.д.

Важнейшим компонентом машины обработки знаний является интеллектуальный решатель задач.

Можно разделить существующие подходы к построению решателей задач на два класса:

- решение задач с использованием хранимых программ. В данном случае предполагается, что в системе заранее присутствует программа решения задачи заданного класса и решение сводится к поиску такой программы и интерпретации ее на заданных входных данных. К данному классу относятся, в том числе, системы, использующие генетические алгоритмы и нейросетевые модели обработки знаний.
- решение задач в условиях, когда программа решения не известна. В этом случае предполагается, что в системе необязательно присутствует готовая программа решения для класса задач, которому принадлежит некоторая сформулированная задача, подлежащая решению. В связи с этим необходимо применять дополнительные методы поиска путей решения задачи, не рассчитанные на какой-либо узкий класс задач (например, разбиение задачи на подзадачи, методы поиска решений в глубину и ширину, метод случайного поиска решения и метод проб и ошибок, метод деления пополам и др.), а так же различные модели логического вывода (классические дедуктивные, индуктивные, абдуктивные; модели, основанные на нечетких логиках, темпоральной логике и т.д.).

Таким образом, существует большое число подходов к построению различных компонентов машин обработки знаний компьютерных систем, в том числе - интеллектуальных решателей задач, многие из которых успешно реализованы и активно используются.

Рассмотрим два основных исторически сложившихся подхода к построению машин обработки знаний.

Первый подход предполагает наличие в системе фиксированной машины обработки знаний (например, машины логического вывода), к которой впоследствии добавляется база знаний, наполнение которой определяется предметной областью, в которой должна работать система. Такие системы получили название «пустых» экспертных систем или «оболочек» (expert system shells). Данный подход, как правило, использовался для разработки относительно несложных систем и в настоящее время не имеет широкого применения.

Второй подход, широко используемых в настоящее время, предполагает наличие программных средств доступа к информации, хранящейся в некоторой базе, совместимых с различными популярными языками программирования. Данный подход широко используется, например, в системах, построенных на основе стандартов W3C. Структура всей машины обработки, построенной на базе таких средств, определяется разработчиком в каждом конкретном случае и не фиксируется какими-либо стандартами. Такой подход обладает большей гибкостью, но отсутствие унификации в структуре и процессе проектирования машины приводит к отсутствию совместимости компонентов машин, созданных разными разработчиками, большому количеству дублирований одних и тех же решений, повышению накладных расходов в процессе разработки и поддержки машины.

Очевидным становится тот факт, что каждой разрабатываемой системе необходима своя уникальная машина обработки знаний, учитывающая особенности конкретной системы и предполагающая возможность ее быстрой корректировки в случае необходимости, в то время как большинство современных систем имеют фиксированную машину обработки знаний, способную решать задачи из небольшого ограниченного класса (например, осуществлять дедуктивный логический вывод на основе нескольких правил).

В то же время, актуальным становится вопрос о возможности одновременного использования в рамках одной системы нескольких механизмов решения задач, что обусловлено высокой востребованностью систем, способных автономно работать в условиях, имеющих высокий уровень непредсказуемости (например, в космосе или других условиях, не пригодных для работы в них человека). Системы, реализующие жестко фиксированный набор алгоритмов, не могут удовлетворить данному требованию, в связи с чем актуальным становится вопрос о возможности быстрого наращивания функционала системы прямо в процессе ее работы.

Несмотря на то, что в настоящее время существует большое число моделей, методов и средств обработки знаний, многие из которых успешно используются в различных системах, до сих пор остаются актуальными следующие проблемы:

- отсутствие единых универсальных принципов, лежащих в основе реализации различных моделей обработки знаний приводит к большому количеству дублирований аналогичных решений в разных системах и невозможности использовать решения, реализованные в одной системе, в других системах. Как следствие, высока трудоемкость разработки каждой такой машины, велики сроки их разработки, затруднена возможность одновременного использования различных моделей решения задач в рамках одной системы;
- разрабатываемые машины обработки знаний не обладают гибкостью, т.е. отсутствует или сильно затруднена возможность дополнения уже созданной машины новыми компонентами и внесения изменений в уже существующие компоненты. Таким образом, высока трудоемкость поддержки разработанных машин, что приводит к их быстрому моральному старению;
- высок уровень профессиональных требований к разработчикам машин обработки знаний;
- попытки объединения большого числа разработчиков в коллективы недостаточно эффективны по причине отсутствия иерархичности в разрабатываемых машинах и, как следствие, в коллективах разработчиков. Трудности в согласовании действий приводят к дополнительным накладным расходам.

Следствием указанных проблем является сравнительно высокая трудоемкость разработки и сопровождения систем, основанных на знаниях, а как следствие — их высокая стоимость.

В рамках данной работы решение указанных проблем предлагается осуществлять с использованием онтологического подхода, в данном случае – онтологического подхода к проектированию машин обработки знаний.

Таким образом, для решения описанных выше проблем в области построения машин обработки знаний предлагается разработать:

- унифицированную онтологическую модель машины обработки знаний, обладающей свойствами гибкости, модульности, платформенной независимости и позволяющую реализовать на ее основе любые существующие модели и методы обработки знаний, в том числе параллельной и асинхронной;
- онтологию проектирования машин обработки знаний, построенных на основе указанной выше модели, включающую описание методики проектирования и формальную типологию действий разработчика таких машин;
- онтологическую модель системы поддержки проектирования машин обработки знаний, построенных на основе указанной модели и проектируемых по описанной методике.

Первоначальная апробация разработанных моделей и средств осуществлялась на базе самой метасистемы поддержки проектирования интеллектуальных систем IMS. В процессе работы осуществлено первоначальное наполнение библиотеки многократно используемых sc-агентов и программ обработки знаний.

Разработка универсальной модели машины обработки знаний на основе представленной выше системы предметных областей и онтологий позволяет унифицировать различные подходы к обработке знаний, что в свою очередь дает возможность:

- обеспечить на основе предлагаемой модели реализацию любых моделей обработки знаний и решения задач, в том числе параллельных и асинхронных;
- при необходимости интегрировать различные подходы к решению задач в рамках одной системы и обеспечить их одновременную работу;
- рассматривать любую машину обработки знаний как иерархическую систему, что существенно повышает эффективность процессов ее проектирования, реализации и отладки;
- обеспечить платформенную независимость реализованных машин обработки знаний и их компонентов;
- за счет использования многоагентного подхода и унификации принципов выделения агентов обеспечить гибкость реализованных машин обработки знаний;
- за счет универсального и унифицированного представления обрабатываемых знаний обобщать существующие подходы к решению некоторых классов задач, позволяя перейти от процедурной формулировки задач к декларативной, обеспечивая таким образом большую гибкость реализованных решений;

Разработка унифицированной методики проектирования машин обработки знаний в виде онтологии проектирования позволяет существенно снизить количество ситуаций, в которых аналогичные решения при их реализации различными разработчиками оказываются несовместимыми, что влечет за собой необходимость дублирования аналогичных решений в разных системах. Кроме того, такой подход дает возможность широко использовать уже реализованные компоненты при проектировании новых машин, существенно снижая при этом накладные расходы на их реализацию.

Кроме того, разработана модель системы поддержки проектирования машин обработки знаний. Разработанная и реализованная модель системы поддержки проектирования машин обработки знаний позволит автоматизировать деятельность разработчиков таких машин и снизить накладные расходы на их верификацию и отладку. Кроме того, указанная модель разработана с использованием всех перечисленных выше подходов, что позволяет обеспечить гибкость самой такой системы.