Ontological approach to the development of hybrid problem solvers for intelligent computer systems

Daniil Shunkevich
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: shunkevich@bsuir.by

***Abstract*—The paper considers an ontological approach to the development of problem solvers for intelligent computer systems based on the OSTIS Technology. The formal interpretation of such concepts as action, problem, class of actions, class of problems, method, skill is clarified, which together made it possible to define on their basis the concepts of a problem-solving model and a problem solver. The results obtained will improve efficiency of the component approach to the development of problem solvers and automation tools for the development of problem solvers.**

***Keywords*—OSTIS, problem solver, multiagent system, problem-solving model, ontological approach**

I. Introduction

The problem solver (along with the knowledge base) is a key component of an intelligent system, on which its ability to solve various problems significantly depends. The peculiarity of problem solvers of intelligent systems in comparison with other modern software systems is the need to solve problems in conditions when the information required is not explicitly localized in the knowledge base of the intelligent system and must be found based on any criteria [1].

If at the dawn of the development of artificial intelligence technologies scientists have been trying for a long time to find some universal mechanism that would allow solving any problem, at present, for each specific intelligent system a special problem solver is being developed, the composition of which is determined by a set of classes of problems that the appropriate intelligent system should solve. As a rule, each class of problems corresponds to some *problem-solving model*. Currently, in the field of artificial intelligence a large number of such models have been developed, some of which are considered as traditional (for example, classical algorithms, procedural and object-oriented programs) and some – as intelligent (neural network models, logical models, genetic algorithms).

The expansion of the scope of intelligent systems requires such systems to be able to solve complex problems, the solution of each of which involves the joint usage of many different knowledge representation models and various problem-solving models. In addition, the solution of complex problems implies the usage of common informational resources (in the limiting case – of the entire knowledge base of an intelligent system) by various components of the solver focused on solving various subproblems. Since the solver of complex problems integrates various problem-solving models, we will call it a *hybrid problem solver* [1].

Modern approaches to the construction of hybrid problem solvers, as a rule, involve a combination of heterogeneous problem-solving models without any single basis, for example, using specialized software interfaces between different components of the system, which leads to considerable overhead costs when developing such a system and especially when its modifying, including when adding a new problem-solving model to the system [1].

An approach to the development of hybrid solvers that allows them to be modifiable is proposed within the framework of the OSTIS Technology [2] and is considered in detail in some papers, in particular, in [1].

Within the framework of this approach, the problem solver is interpreted as a hierarchical system of agents (sc-agents) that work on shared semantic memory (sc-memory) and interact by the specification of the actions they perform within this memory. It is assumed that each problem-solving model corresponds to some sc-agent (most often – a non-atomic one that could be decomposed into simpler sc-agents). Thus, it becomes possible to combine different problem-solving models when solving the same complex problem as well as to add new problem-solving models to the solver or exclude them without having to make modifications in its other components.

However, the further development of this approach and, in particular, its usage when developing various applied intelligent systems has shown that the capabilities of the problem solver are also in large part determined by the quality of the knowledge base of the appropriate intelligent system. It may safely be said that the approach to the development of solvers discussed above is connected with the description of the *operational semantics* of the solver, that is, interpreters of the appropriate problem-solving models, while it is obvious that for solving problems it is also necessary to describe the *declarative semantics* of the problem-solving model, that is, the texts of programs itself (not the programs of sc-agents but higher-level programs interpreted by the corresponding set of sc-agents), logical statements, certain configurations of artificial neural networks, etc.

Within the framework of the OSTIS Technology, powerful tools have been developed that allow describing any type of knowledge in a unified form, structuring the knowledge base according to various criteria as well as verifying its quality and editing the knowledge base directly in its use [3]. The basis of the knowledge base created using the OSTIS Technology is a hierarchical system of subject domains and the corresponding ontologies. An ontology is interpreted as a specification of the system of concepts of the corresponding subject domain, while various types of ontologies are distinguished, each of which reflects a certain set of the concept features of the subject domain, for example, *terminological ontology*, *logical ontology*, *set-theoretic ontology*, etc. Speaking about ontologies in the context of this paper, we will have in mind an integrated ontology, which is a combination of ontologies of all types that correspond to a specific subject domain.

II. Proposed approach

Within the framework of this paper, it is proposed to take as a basis the approaches to the development of hybrid problem solvers and hybrid knowledge bases proposed within the context of the OSTIS Technology, to formally clarify and coordinate the interpretation of such concepts as *problem*, *problem-solving model*, *problem solver*, *skill* and others within the appropriate set of ontologies and on the basis of the results obtained to clarify the actual model of the hybrid problem solver, which would allow taking into account the abovementioned aspects.

The systems developed on the basis of the OSTIS Technology are called *ostis-systems*. The *OSTIS Technology* is based on a universal method of semantic representation (encoding) of information in memory of intelligent computer systems called *SC-code*. Texts of the *SC-code* (sc-texts) are unified semantic networks with a basic set-theoretic interpretation. The elements of such semantic networks are called *sc-elements* (*sc-nodes* and *sc-connectors*, which, in turn, can be *sc-arcs* or *sc-edges*, depending on the directivity). The *SC-code Alphabet* consists of five main elements, on the basis of which SC-code constructs of any complexity are built as well as more particular types of sc-elements (for example, new concepts) are introduced. Memory that stores the SC-code constructs is called semantic memory or *sc-memory*.

Within the framework of the technology, several universal variants of visualization of the *SC-code* constructs are also proposed, such as *SCg-code* (graphic version), *SCn-code* (non-linear hypertextual version), *SCs-code* (linear string version).

As it was mentioned earlier, the basis of the knowledge base within the framework of the OSTIS Technology is a hierarchical system of subject domains and ontologies. From there, to solve the problems set within the framework of this paper, it is proposed to develop a complex *Subject domain of actions and problems and the corresponding ontology of problem-solving methods and models*.

Within the framework of this paper, fragments of structured texts in the SCn-code [4] will often be used, which are simultaneously fragments of source texts of the knowledge base, which are understandable both to a human and to a machine. This allows making the text more structured and formalized while maintaining its readability. The symbol ":=" in such texts indicates alternative (synonymous) names of the described entity, which reveal in more detail some of its features.

The development of the specified family of sc-models of subject domains and ontologies will allow:

- explicitly linking the class of problems and the way (method) of its solution;
- this, in turn, will allow accumulating more complex components of solvers and massively simplify their integration, since the appropriate component combined with the group of sc-agents will also include the necessary fragments of the knowledge base, which are a priori squared with the specified group of sc-agents;
- this, in turn, will allow making the automation tools for the development of solvers more intelligent, in particular, it will allow automating the process of selecting solver components based on the specification of classes of problems that the designed intelligent system should be able to solve;
- in the future, this will allow the intelligent system to independently access the library of problem solver components and select components based on new classes of problems that the system has encountered, that is, it will allow the intelligent system to independently learn new *skills*;
- on the other hand, this approach will allow the intelligent system to independently select a combination of problem-solving models for solving problems of a certain class (more exactly, since the solver is based on a multiagent approach, a group of sc-agents that interpret different problem-solving models will be able to determine better, which of the sc-agents and

in what order should work when solving a specific complex problem).

The subject areas and ontologies discribed in this work were developed on the basis of the theory of subject-object influences proposed in the work of V. Martynov and his colleagues [5], [6], [7], [8], [9].

Consider next in more detail fragments of sc-models of specified subject domains and ontologies.

III. Concept of an action and the classification of actions

Before getting to the problem-solving models and the problem solver, it is necessary to formally clarify the concept of a problem and the concept of an action aimed at solving a particular problem or its subproblems.

Within the framework of the *OSTIS Technology*, we will interprete the problem as a formal specification of some action, so it is reasonable at first to clarify the concept of an *action*. Let us consider the specification of the concept *action* in the SCn-code.

action
:= [a purposeful process performed by one or more subjects (cybernetical systems) with the possible usage of certain tools]
:= [a process of influencing some (possibly shared) entity (the subject of influence) on one or several entities (objects of influence – source objects (arguments) or target (created or modified) objects)]
:= [an actio]
:= [an act]
:= [an operation]
:= [a conscious influence]
:= [an active influence]
⊂ *influence*
  := [a process, in which at least one influencing entity (the subject of influence$'$) and at least one entity that is being influenced (the object of influence$'$) can be clearly distinguished)]
  ⊂ *process*
:= [a purposeful ("conscious") process performed (managed, implemented) by some subject]
:= [a process of solving some problem]
:= [a purposeful process managed by some subject]
⇒ *decomposition**:
  Decomposition of a class of actions in relation to memory of a cybernetical system
  = {• *informational action*
      ⊃ *action in sc-memory*
    • *behavioural action*
      ⊃ *action in the environment of the ostis-system*
    • *effector action*
      ⊃ *effector action of the ostis-system*
    • *receptor action*
      ⊃ *receptor action of the ostis-system*
  }
⊃ *atomic action*
  := [an action, the performance of which does not require its decomposition into a set of sub-actions (particular actions, actions of a lower level)]
  ⇒ *explanation**:
    [An atomic action is performed by a single individual subject and is either an atomic action performed in memory of this subject (an atomic action of its "processor") or an atomic action of one of its effectors.]
⊃ *complex action*
⇒ *subdividing**:
  {• *an action performed by a cybernetical system in its own memory*
   • *an action performed by a cybernetical system in its environment*
   • *an action performed by a cybernetical system on its physical shell*
  }

The result of performing an ***informational action*** is generically a certain new state of information system memory (not necessarily of *sc-memory*) achieved only by transforming the information stored in system memory, that is, either by generating new knowledge based on existing ones or by deleting knowledge that has become unnecessary for whatever reason. It should be noted that if the question is about changing the state of *sc-memory*, then any transformation of information can be reduced to some atomic actions of generating, deleting or changing the incidence of *sc-elements* relative to each other.

In the case of a ***behavioral action***, the result of its performance will be a new state of the environment. It is very important to note that in this case the environment also means the components of the system that are external from the point of view of memory, that is, they are not information structures stored in it. Such components include, for example, various manipulators and other means of influencing the system on the external world, that is, behavioral problems can include changing the state of a mechanical limb of a robot or directly displaying some information on the screen for the user experience.

From the point of view of the problem solving formulated in this paper, the informational actions performed in memory of the ostis-system, that is, *actions in sc-memory*, promote outstanding interest. The classification of *actions in sc-memory* is presented in the knowledge base of the *IMS. ostis Metasystem* that describes the documentation of the current state of the *OSTIS Technology* [4].

On the set of actions a number of relations are set, such as *action subject$'$* (*performer$'$*), *customer**, *action object$'$*, *action context**, *sub-action**, *sequence of actions**, *result** and others [1], [4].

IV. Concept of a problem and the classification of problems

In turn, a *problem* will be interpreted as a specification of some action, within which, depending on the situation, the context of the action performance, the way of its performance, the performer, the customer, the planned result, etc. can be specified in advance using the relations listed above.

Let us consider the specification of the concept *problem* in the SCn-code.

problem
:= [a description of some desirable state or event either in the knowledge base or in the environment]
:= [a problem definition]
:= [a task for performing some action]
:= [a problem description]
:= [a problem situation]
:= [a specification of some action that has sufficient completeness to perform this action]

Each ***problem*** is a specification of an action that either has already been performed, or is currently being performed, or is planned (should) be performed, or can be performed (but not necessarily). Depending on the specific class of problems, both the internal state of the intelligent system itself and the required state of the environment can be described.

Classification of problems can be carried out on a didactic basis within each subject domain, for example, triangle problems, problems on sets of equations, etc.

Each *problem* can include:

- the fact that an *action* belongs to some particular class of *actions* (for example, *action. form a complete semantic neighborhood of the specified entity*), including the state of the *action* from the point of view of the life cycle (initiated, performed, etc.);
- a description of the *purpose** (*result**) of the *action*, if it is exactly known;
- specifying the action *customer**;
- specifying the action *performer** (including a collective one);
- specifying the *action argument(-s)′*;
- specifying a tool or mediator of the *action*;
- a description of the *action decomposition**;
- specifying a *sequence of actions** within the *action decomposition**, i.e., construction of a procedural plan for solving the problem. In other words, the construction of a solution plan is a decomposition of the corresponding *action* into a system of interconnected sub-actions;
- specifying the domain of the *action*;
- specifying the condition for initiating the *action*;
- the moment of the starting and ending the *action*, including the planned and actual ones, the expected and/or actual duration of the performance.

Some problems can be clarified further by the context – additional information about the entities considered in the *problem* definition, i.e., a description of what is given, what is known about these entities.

In addition, a *problem* can include any additional information about the action, for example:

- a list of resources and means that are supposed to be used in solving the problem, for example, a list of available performers, timescales, available funding, etc.;
- the restriction of the domain, in which the *action* is performed, for example, one *sc-construct* must be replaced by another according to some rule but only within some *knowledge base section*;
- the restriction of knowledge that can be used for solving a particular problem, for example, it is necessary to solve an algebra problem using only those statements that are included in the course of the school curriculum up to and including the seventh grade and not using statements studied in high school;
- etc.

As in the case of actions solved by the system, it is possible to classify *informational problems* and *behavioral problems*.

On the other hand, from the point of view of the problem definition, *declarative problem definitions* and *procedural problem definitions* can be distinguished. It should be noted that these classes of problems are not opposed to each other and there may be problem definitions that use both approaches.

problem
⊃ *procedural problem definition*
⊃ *declarative problem definition*
⊃ *question*
⊃ *command*
⊂ *knowledge*
⊃ *initiated problem*
  := [a problem definition to be performed]
⊃ *declarative problem definition*
⊃ *procedural problem definition*
⊃ *declarative-procedural problem definition*
  := [a problem, in the definition of which there are both declarative (target) and procedural aspects]
⊃ *problem solved in memory of a cybernetical system*
  ⊃ *problem solved in memory of an individual cybernetical system*
  ⊃ *problem solved in shared memory of a multiagent system*

:= [an informational problem]
:= [a problem aimed either at generation or search for information that meets the specified requirements or at some transformation of the specified information]
⊃ *mathematical problem*

The *problem* definition may not contain an indication of the context (solution domain) of the *problem* (in this case, the *problem* solution domain is either the entire *knowledge base* or its compliant part) and may also not contain either a description of the underlying situation or a description of the target situation. For example, a description of the target situation for an explicitly specified contradiction found in a *knowledge base* is not required.

Declarative problem definition is a description of the underlying (initial) situation, which is a condition for performing the corresponding action, and the target (final) situation, which is the result of performing this action, that is, a description of the situation (state) that should be achieved as a result of performing the planned action. In other words, such a problem definition includes an explicit or implicit description of:

- what is given – the source data, conditions for performing a specified action;
- what is required – the definition of the purpose and the result of performing the specified action.

In the case of the ***procedural problem definition***, the characteristic of the action specified by this problem is explicitly indicated, namely, for example:

- a subject or subjects that perform this action;
- objects, on which the action is performed – arguments of the action;
- tools that are used to perform the action;
- the moment and, possibly, additional conditions for starting and ending the action;
- a class or classes that each *action* belongs to (including sub-actions) are explicitly specified.

At the same time, it is not explicitly specified what should be the result of performing the corresponding action.

Let us note that, if necessary, the *procedural problem definition* can be reduced to the *declarative problem definition* by translating based on some rule, for example, of the definition of the class of actions through a more general class.

Particular types of problems are a *question* and a *command*.

question
:= [a request]
⊂ *problem solved in memory of a cybernetical system*
:= [a non-procedural problem definition for searching (in the current state of the knowledge base) or for generating knowledge that meets the specified requirements]
⊃ *question – what is it*
⊃ *question – why*
⊃ *question – wherefore*
⊃ *question – how*
:= [a request for a method (way) for solving a given (specified) type of problems or class of problems or a plan for solving a particular specified problem]
:= [a problem aimed at satisfying the information needs of a certain subject-customer]

command
:= [an initiated problem]
:= [a specification of the initiated action]

It should be noted that along with the given extremely general classification of problems, which inherently reflects the classes of problems from the point of view of their definition, there should be a classification of problems from the point of view of their semantics, that is, in terms of the essence of the specified action. This classification can be based on the classification presented in [10].

Within the framework of this paper, as already mentioned, the problems solved in sc-memory promote outstanding interest.

V. Concepts of a class of actions and a class of problems

From the point of view of the organization of the problem-solving process, the concepts of an *action* and a *problem* are not more important than the concepts of a *class of actions* and a *class of problems*, since it is for them that the appropriate performance algorithms and solution methods are being developed.

Let us define a ***class of actions*** as a maximal set of coincident (similar in a certain way) actions, for which there is (but is not necessarily currently known) at least one **method** (or mean) that provides the performance of any action from the specified set of actions.

class of actions
⇐ *family of subclasses**:
action
:= [a set of similar actions]
⊃ *class of atomic actions*
⊃ *class of easily performable complex actions*

Each distinguished *class of actions* corresponds to at least one common *method* for performing these *actions*. It means that the question is about semantic "clustering" of a set of *actions*, i.e., about the allocation of *classes of actions* on the basis of the semantic similarity (coincidence) of *actions* that are part of the selected *class of actions*.

In this case, first of all, the coincidence (similarity) of *underlying situations* and *target situations* of the *actions* being considered, i.e., the coincidence of *problems* solved as a result of performing the corresponding *actions*, is taken into account. Since one and the same *problem* can be solved as a result of performing several different *actions* that belong to different *classes of actions*, we should talk not only about *classes of actions* (sets of similar actions) but also about ***classes of problems*** (sets of similar problems) solved by these *actions*. For example, the following *relations* are set on the set of *classes of actions*:

- a *relation*, each bunding of which connects two different (disjoint) *classes of actions* that solve one and the same *class of problems*;
- a *relation*, each bunding of which connects two different *classes of actions* that solve different *classes of problems*, one of which is a *superset* of the other.

In addition to the class of actions, the concept of a ***class of atomic actions*** is also distinguished, that is, the set of atomic actions, the indication of belonging to which is a necessary and sufficient condition for performing this action. The set of all possible atomic actions performed by each subject should be divided into classes of atomic actions.

Belonging of some *class of actions* to the set of the ***classes of atomic actions*** fixes the fact that, when all the necessary arguments are specified, belonging of *action* to this class is sufficient for some subject to start performing this action.

At the same time, even if the *class of actions* belongs to the set of the ***class of atomic actions***, it is not forbidden to introduce more particular *classes of actions*, for which, for example, one of the arguments is fixed in advance.

If a specified ***class of atomic actions*** is more particular in relation to *actions in sc-memory*, this indicates that there is at least one *sc-agent* in the current version of the system that is focused on performing actions of this class.

In addition, it is also reasonable to introduce the concept of a *class of easily performable complex actions*, that is, a set of *complex actions*, for which at least one ***method*** is known and available, the interpretation of which allows performing a complete (final, ending with atomic actions) decomposition into sub-actions of each complex action from the above set.

Belonging of some *class of actions* to the set of the ***class of easily performable complex actions*** fixes the fact that, even when specifying all the necessary arguments of belonging the *action* to this class, it is unsufficient for some *subject* to start performing this action, and additional clarifications are required.

In turn, by the ***class of problems*** we will mean the set of problems, for which it is possible to construct a generalized definition of problems that corresponds to the whole set of problems. Each *generalized definition of the problems of the corresponding class* is in fact nothing more than a strict logical definition of the specified class of problems.

class of problems
⇐ *family of subsets**:
  problem

A specific class of actions can be defined in at least two ways.

class of actions
⇒ *subdividing**:
  {• ***class of actions that is precisely defined by the class of problems being solved***
    := [a *class of actions* that provide a solution of the corresponding *class of problems* and at the same time use a wide variety of *methods* for solving problems of this class]
  • ***class of actions that is precisely defined by the used method of solving problems***
  }

Further, let us consider in more detail the formal interpretation of the concept of a *method*.

VI. Concept of a method

By the method we will mean a description of how any or almost any (with explicit exceptions) action that belongs to the corresponding class of actions can be performed.

method
⇐ *second domain**:
  *method**
:= [a method for solving the corresponding class of problems that provides a solution of any or most of the problems of the specified class]
:= [a program for solving problems of the corresponding class, which can be both procedural and declarative (non-procedural) ones]
⊂ *knowledge*
∈ *type of knowledge*
:= [a way]
:= [a knowledge of how it is necessary to solve problems of the corresponding class of problems (a set of equivalent (similar) problems)]
:= [a method (way) for solving a certain (corresponding) class of problems]
:= [an information (knowledge) sufficient to solve any *problem* that belongs to the corresponding *class of problems* using the corresponding *problem-solving model*]

The specification of each *class of problems* includes a description of how to "bind" a *method* to the source data of a specific *problem* that is being solved using this *method*. The description of such a method of "binding" includes:

- a set of variables that are included both in the *method* and in the *generalized definition of problems of the corresponding class* and whose values are the corresponding elements of the source data of each specific problem being solved;
- a part of the *generalized definition of problems* of the class, to which the *method* being considered corresponds, which are a description of the conditions of usage of this *method*.

The very "binding" of the *method* to a specific *problem* being solved using this *method* is carried out by searching in the *knowledge base* for such a fragment that satisfies the conditions for using the specified *method*. One of the results of such a search is to establish a correspondence between the abovementioned variables of the used *method* and the values of these variables within the framework of a specific *problem* being solved.

Another option for establishing the correspondence being considered is an explicit appeal (call) of the corresponding *method* (program) with the explicit transmission of the corresponding parameters. But this is not always possible, because, when performing the process of solving a specific *problem* based on the declarative specification of performing this action, it is not possible to set:

- when it is necessary to initiate the call (use) of the required *method*;
- which specific *method* should be used;
- what parameters that correspond to the specific initiated *problem* should be transmitted for "binding" the used *method* to this *problem*.

The process of "binding" the *method* of solving *problems* to a specific *problem* solved using this *method* can also be represented as a process that consists of the following stages:

- construction of a copy of the used *method*;
- binding the main (key) variables of the used *method* with the main parameters of the specific *problem* being solved.

As a result, on the basis of the considered *method* used as a sample (template), a specification of the process for solving a specific problem – a procedural specification (*plan*) or a declarative one – is built.

Let us note that *methods* can be used even when constructing *plans* for solving specific *problems*, in the case when there is a need for multiple repetition of certain chains of *actions* with an a priori unknown number of such repetitions. It is question about various types of **cycles**, which are the simplest type of procedural *methods* for solving problems that are repeatedly used when implementing *plans* for solving some *problems*.

It is also obvious that several *methods* can correspond to one *class of actions*.

Thus, we assume that the term "method" is with the term "program" synonymous in the generalized sense of this term.

method
:= [a program]
:= [a program for performing actions of a certain class]
⊃ *procedural program*
    := [a generalized plan]
    := [a generalized plan for performing a certain class of actions]
    := [a generalized plan for solving a certain class of problems]
    := [a generalized specification of the decomposition of any action that belongs to a given class of actions]
    ⊂ *algorithm*

Let us consider in more detail the concept of a procedural program (procedural method). Each ***procedural program*** is a generalized plan for performing *actions* that belong to a certain class, that is, it is a *semantic neighborhood; the key sc-element ′* is a *class of actions*, for the elements of which the process of their performance is additionally detailed.

The input parameters of the *procedural program* in the traditional sense correspond to the arguments that correspond to each *action* from the *class of actions* described by this *procedural program*. When generating a specific *plan* of performing a specific *action* from this class based on this program, these arguments take specific values.

Each *procedural program* is a system of described actions with an additional indication for the action:

- or a *sequence of actions** (transmission of initiation) when the condition for performing (initiating) actions is the performance of one of the specified or all of the specified actions;
- or an event in the knowledge base or the environment that is a condition for its initiation;
- or a situation in the knowledge base or the environment that is a condition for its initiation.

The concept of a method allows determining the relation *problem equivalence** on a set of problems. Problems are equivalent if and only if they can be solved by interpreting one and the same *method* (way) stored in memory of a cybernetical system.

Some *problems* can be solved by different *methods*, one of which, for example, is a generalization of the other. Thus, some relations can also be set on a set of methods.

Let us note that the concept of a *method* allows localizing the domain of solving problems of the corresponding class, that is, limiting the set of knowledge that is sufficient to solve problems of this class in a certain way. This, in turn, allows increasing the efficiency of the system as a whole, eliminating the number of unnecessary actions.

relation defined on a set of methods
∋ *submethod**
:= [a subprogram*]
:= [to be a method that is supposed to be used (accessed) when implementing a given method*]
⇔ *it is important to distinguish**:
*particular method**
:= [to be a method that provides a solution to a class of problems, which is a subclass of problems being solved using a given method*]

In the literature dedicated to the construction of problem solvers, the concept of a ***problem-solving strategy*** is found. Let us define it as a meta-method for solving problems that provides either the search for one relevant known method or the synthesis of a purposeful sequence of actions using various known methods in the general case.

problem-solving strategy
⊂ *method*

It can be said about a universal meta-method (universal strategy) for solving problems that explains all kinds of particular strategies.

In particular, we can talk about several global *strategies for solving informational problems* in knowledge bases. Let us assume that a sign of an initiated action with the definition of the appropriate informational purpose, i.e., a purpose aimed only at changing the state of the knowledge base, has appeared in the knowledge base. And the current state of the knowledge base does not contain a context (source data) sufficient to achieve the above purpose, i.e., such a context, for which there is a method (program) in the available package (set) of methods (programs), the usage of which allows achieving the above purpose. To achieve such a purpose, the context (source data) of which is insufficient, there are three approaches (three strategies):

- decomposition (reduction of the initial purpose to a hierarchical system and/or subpurposes (and/or subproblems) based on the analysis of the current state of the knowledge base and the analysis of what is missing in the knowledge base for using a particular method).
  At the same time, the most attention is paid to methods that require less effort to create conditions for using them. Ultimately, we must reach (at the lowest level of the hierarchy) subpurposes, the context of which is sufficient for the usage of one of the available methods (programs) for solving problems;
- generation of new knowledge in the semantic neighborhood of the definition of the initial purpose using any available methods in the hope of obtaining such a state of the knowledge base that will contain the necessary context (sufficient source data) to achieve the initial purpose using any available method of solving problems;
- combination of the first and second approaches.

Similar strategies exist for finding ways to solve problems being solved in the environment.

VII. Specification of methods and the concept of a skill

Each specific *method* is considered by us not only as an important type of specification of the corresponding class of problems but also as an object that itself needs a specification that provides direct usage of this method. In other words, the method is not only a specification (the specification of the corresponding class of problems) but also an object of the specification. The most important type of such specification is the indication of the *operational semantics of the method*.

operational semantics of the method*
⊂ *specification**
:= [a family of methods that provide interpretation of a given method*]
:= [a formal description of the interpreter of a given method*]
⇒ *second domain**:
operational semantics of the method
⊃ ***complex representation of the operational semantics of the method***
:= [a representation of the *operational semantics of the method* brought (detailed) to the level of all *specifications of atomic actions* performed during the interpretation of the corresponding *method*]

declarative semantics of the method*
⊂ *specification**
:= [a description of the system of concepts that are used within the framework of this method*]

The relation *declarative semantics of the method** connects the *method* and the formal description of the system of concepts (a fragment of the *logical ontology* of the corresponding *subject domain*) that are used (mentioned) within this method. This is necessary for ensuring that one and the same concept is interpreted unambiguously within the framework of the method and the rest of the knowledge base, which is especially important when borrowing a method from a library of reusable components of problem solvers. It is important to note that the fact that any concepts are used within the framework of the method does not mean that the formal record of their definitions is part of this method. For example, a method that allows solving problems for calculating the area of a triangle will include various formulas for calculating the area of a triangle but will not include the definitions of the concepts ‘area", ‘triangle", etc., since if there are a priori correct formulas, these definitions will not be used directly in the process of solving the problem. At the same time, the formal definitions of these concepts will be part of the declarative semantics of this method.

Combining the *method* and its operational semantics, that is, information about how this *method* should be interpreted, we will call a ***skill***.

skill
:= [an ability]
:= [a combination of a *method* with its comprehensive specification – a *complex representation of the operational semantics of the method*]
:= [a method + a method of its interpretation]
:= [an ability to solve the corresponding class of equivalent problems]
:= [a method plus its operational semantics, which describes how this method is interpreted (performed, implemented) and is at the same time the operational semantics of the corresponding problem-solving model]
⇒ *subdividing**:
{• *active skill*
:= [a self-initiating skill]
• *passive skill*
}

Thus, the concept of a *skill* is the most important concept from the point of view of constructing problem solvers, since it combines not only the declarative part of the description of the method of solving a class of problems but also the operational one.

Skills can be *passive skills*, that is, such *skills*, the usage of which must be explicitly initiated by some agent, or *active skills*, which are initiated independently when a corresponding situation occurs in the knowledge base. To do this, in addition to the *method* and its operational semantics, the *sc-agent*, which responds to the appearance of a corresponding situation in the knowledge base and initiates the interpretation of the *method* of this *skill*, is also included in the *active skill*.

This separation allows implementing and combining different approaches for solving problems, in particular, *passive skills* can be considered as a way to implement the concept of a smart software package.

VIII. Concepts of a class of methods and a language for representing methods

Like actions and problems, methods can be classified into different classes. We will define a set of methods, for which it is possible to unify the representation (specification) of these methods, as a ***class of methods***.

class of methods
⇐ *family of subclasses**:
method
:= [a set of methods, for which the representation language of these methods is set]
∋ *procedural method for solving problems*
⊃ *algorithmic method for solving problems*
∋ *logical method for solving problems*
⊃ *productional method for solving problems*
⊃ *functional method for solving problems*
∋ *artificial neutral network*
:= [a class of methods for solving problems based on artificial neural networks]
∋ *genetic "algorithm"*
:= [a set of methods based on a common ontology]
:= [a set of methods represented in the same language]
:= [a set of methods for solving problems, which corresponds to a special language (for example, an sc-language) that provides a representation of methods from this set]
:= [a set of methods that corresponds to a separate problem-solving model]

Each specific *class of methods* mutually identically corresponds to a *language for representing methods* that belong to this (specified) *class of methods*. Thus, the specification of each *class of methods* is reduced to the specification of the corresponding *language for representing methods*, i.e., to the description of its syntactic, denotational and operational semantics.

Examples of *languages for representing methods* are all *programming languages*, which mainly belong to the subclass of *languages for representing methods* – to *languages for representing methods for information processing*. But now the need to create effective formal languages for representing methods for performing actions in the environment of cybernetical systems is becoming increasingly relevant. Complex automation, in particular, in the industrial sphere, is impossible without this.

There can be a whole set of such specialized languages, each of which will correspond to its own model of problem solving (i.e., to its own interpreter).

language for representing methods
:= [a method language]

:= [a language for representing methods that correspond to a specific class of methods]
⊂ *language*
:= [a programming language]
⊃ *language for representing methods for information processing*
    := [a language of representing methods for solving problems in memory of cybernetical systems]
⊃ *language of representing methods for solving problems in the environment of cybernetical systems*
    := [a programming language for external actions of cybernetical systems]

IX. Concept of a problem-solving model

By analogy with the concept of a problem-solving strategy, we introduce the concept of a ***problem-solving model***, which we will interpret as a meta-method for interpreting the corresponding class of methods.

problem-solving model
⊂ *method*
:= [a meta-method]
:= [an abstract machine for interpreting the corresponding class of methods]
:= [a hierarchical system of "microprograms" that provide interpretation of the corresponding class of methods]
⊃ *algorithmic problem-solving model*
⊃ *procedural parallel synchronous problem-solving model*
⊃ *procedural parallel asynchronous problem-solving model*
⊃ *productional problem-solving model*
⊃ *functional problem-solving model*
⊃ *logical problem-solving model*
    ⊃ *coherent logical problem-solving model*
    ⊃ *fuzzy logical problem-solving model*
⊃ *"neural network" problem-solving model*
⊃ *"genetic" problem-solving model*

Each *problem-solving model* is defined by:

- the corresponding class of methods for solving problems, i.e., the language of representing methods of this class;
- the subject domain of this class of methods;
- the ontology of this class of methods (i.e., the denotational semantics of the language of representing these methods);
- the operational semantics of the specified class of methods.

It is important to note that for the interpretation of all problem-solving models, an agent-oriented approach considered in [1] can be used.

specification*
⊃ ***problem-solving model****
    = *narrowing the relation by the first domain(specification*; class of methods)**
    := [a specification of the *class of methods**]
    := [a specification of the *language for representing methods**]

The problem-solving model associates the syntax, denotational and operational semantics of the language for representing methods of the corresponding class with a certain class of methods.

denotational semantics of the language for representing methods of the corresponding class
:= [an ontology of the corresponding class of methods]
:= [the denotational semantics of the corresponding class of methods]
:= [the denotational semantics of a language (an sc-language) that provides a representation of methods of the corresponding class]
:= [the denotational semantics of the corresponding problem-solving model]
⇒ *note**:
    [If the question is about a language that provides an internal representation of the methods of the corresponding class in the ostis-system, the syntax of this language coincides with the syntax of the sc-code]
⊂ *ontology*

operational semantics of the language for representing methods of the corresponding class
:= [a meta-method of interpretation of the corresponding class of methods]
:= [a family of agents that provide interpretation (usage) of any method that belongs to the corresponding class of methods]
:= [the operational semantics of the corresponding problem-solving model]

Since each *method* corresponds to a *generalized definition of problems* solved using this *method*, then each *class of methods* must correspond not only to a certain *language of representing methods* that belong to the specified *class of methods* but also to a certain *language of representation of generalized definitions of problems for various classes of problems* that are solved using *methods* that belong to the specified *class of methods*.

X. Concepts of a problem solver and a knowledge processing machine

Taking into account the system of concepts discussed above, we will define the problem solver of an intelligent computer system as a set of *skills* that allow the system to solve problems of a particular class.

problem solver
:= [a problem solver of an intelligent computer system]
:= [a set of all the skills (abilities) acquired by the computer system by now]
⊃ *combined problem solver*
⊃ *hybrid problem solver*

By the ***combined problem solver*** we will mean a solver that provides all the functionality of an intelligent system, that is, the solution of all problems that are related to the direct purpose of the system and ensure the efficiency of its work. Thus, for example, a solver that implements some variant of logical inference cannot be considered as combined, since to use a system that contains such a solver it is necessary to have at least basic information search tools that allow localizing the received answer as well as means that ensure the translation of a question from the user to the system and an answer from the system to the user.

In general, the *combined problem solver*, in contrast to the *problem solver* in a general sense, solves problems related to:

- ensuring the main functionality of the system (solving explicitly defined problems on demand);
- ensuring the correctness and optimization of the system (permanently throughout the system life cycle);
- ensuring the automation of the development of an intelligent system.

By the ***hybrid problem solver*** we will mean a *problem solver*, within which several different *problem-solving models* are used. It is obvious that the *combined problem solver* is predominantly a *hybrid problem solver*, since for the functioning of even a fairly simple intelligent system it is necessary to solve problems of fundamentally different classes discussed above.

In turn, by a *knowledge processing machine* we will mean the set of interpreters of all skills that build some *problem solver*. Taking into account the approach to information processing used within the framework of the OSTIS Technology and discussed in [1], a *knowledge processing machine* is a *sc-agent* (most often – a *non-atomic sc-agent*), which includes simpler sc-agents that provide interpretation of the corresponding set of *methods*.

Thus, we can talk, for example, about a *deductive logical inference machine* or an *information search machine*.

XI. Example of the usage of the developed ontologies

Let us consider the usage of the abovementioned fragments of ontology on the example of the description in the knowledge base of ways to solve a simple problem – the problem of finding roots of a quadratic equation.

As it is known from the school course in mathematics, the problems of this class can be solved in at least two ways – through the discriminant and the Vieta formulas for the quadratic equation.

On the other hand, from the point of view of implementation in the ostis-system, both of these options can also be implemented in two ways:

- in a particular way when an *abstract sc-agent* designed to solve problems of a specific class in a specific way is being developed;
- in a more general way when the corresponding formulas are written in the form of logical rules, which are further interpreted by a group of domain-independent sc-agents. This option is worse in terms of performance but much better in terms of flexibility and extensibility of the system.

Figure 1 shows an example of implementing the solution of problems of the considered class in both ways (through the discriminant and Vieta formulas) in a more particular way in the form of active skills. In this variant, it is assumed that the sc-agent programs are implemented in the SCP language, which is the basic language for processing SC-code texts and whose programs are also written in the SC-code. Based on this, the operational semantics of the methods is an *Abstract scp-machine*, that is, an interpreter of SCP programs.

In turn, figure 2 shows an example of the implementation of the same skills but in a more general way and in the form of passive skills. In this case, the operational semantics of the methods is a *Non-atomic abstract sc-agent of logical inference*, which allows using logical statements and, if necessary, calculating mathematical expressions that are obtained as a result of using a logical statement.

The presented figures also show how the declarative semantics of the corresponding methods are set.

XII. Conclusion

The paper considers an ontological approach to the development of problem solvers for intelligent computer systems based on the OSTIS Technology. The formal interpretation of such concepts as action, problem, class of actions, class of problems, method, skill is clarified, which together made it possible to define on their basis the concepts of a problem-solving model and a problem solver.

Examples of describing skills that allow solving one and the same class of problems in different ways are given.

In the future, the results obtained will increase the efficiency of the component approach to the development of problem solvers and automation tools for the development of problem solvers as well as provide an opportunity not only for the developer but also for the intelligent system to automatically select ways to solve a particular problem.

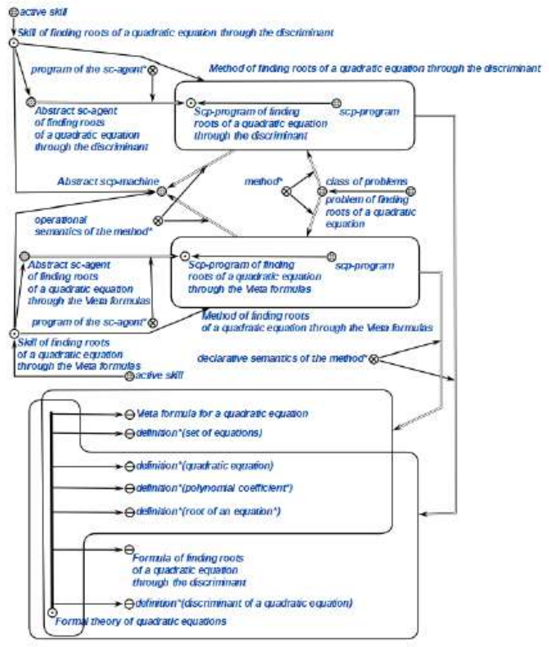
active skill
Skill of finding roots of a quadratic equation through the discriminant
program of the sc-agent*
Method of finding roots of a quadratic equation through the discriminant
Abstract sc-agent of finding roots of a quadratic equation through the discriminant
Scp-program of finding roots of a quadratic equation through the discriminant
scp-program
Abstract scp-machine
method*
class of problems
problem of finding roots of a quadratic equation
operational semantics of the method*
Abstract sc-agent of finding roots of a quadratic equation through the Vieta formulas
Scp-program of finding roots of a quadratic equation through the Vieta formulas
scp-program
program of the sc-agent*
Method of finding roots of a quadratic equation through the Vieta formulas
Skill of finding roots of a quadratic equation through the Vieta formulas
declarative semantics of the method*
active skill
Vieta formula for a quadratic equation
definition*(set of equations)
definition*(quadratic equation)
definition*(polynomial coefficient*)
definition*(root of an equation*)
Formula of finding roots of a quadratic equation through the discriminant
definition*(discriminant of a quadratic equation)
Formal theory of quadratic equations

Figure 1. Example of usage of active skills

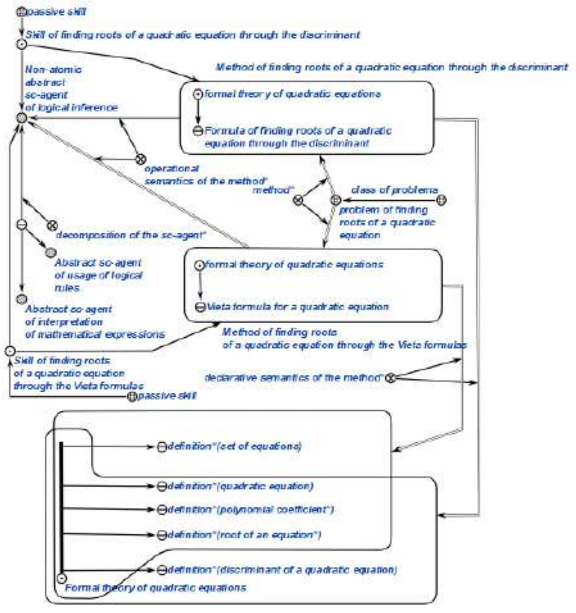
passive skill
Skill of finding roots of a quadratic equation through the discriminant
Non-atomic abstract sc-agent of logical inference
Method of finding roots of a quadratic equation through the discriminant
formal theory of quadratic equations
Formula of finding roots of a quadratic equation through the discriminant
operational semantics of the method*
method*
class of problems
problem of finding roots of a quadratic equation
decomposition of the sc-agent*
Abstract sc-agent of usage of logical rules
formal theory of quadratic equations
Vieta formula for a quadratic equation
Abstract sc-agent of interpretation of mathematical expressions
Method of finding roots of a quadratic equation through the Vieta formulas
Skill of finding roots of a quadratic equation through the Vieta formulas
declarative semantics of the method*
passive skill
definition*(set of equations)
definition*(quadratic equation)
definition*(polynomial coefficient*)
definition*(root of an equation*)
definition*(discriminant of a quadratic equation)
Formal theory of quadratic equations

Figure 2. Example of usage of passive skills

Acknowledgment

The author would like to thank the research groups of the Departments of Intelligent Information Technologies of the Belarusian State University of Informatics and Radioelectronics and the Brest State Technical University for their help in the work and valuable comments.

The work was carried out with the partial financial support of the BRFFR (BRFFR-RFFR No. F21RM-139).

References

[1] D. Shunkevich, “Agentno-orientirovannye reshateli zadach intellektual’nyh sistem [Agent-oriented models, method and tools of compatible problem solvers development for intelligent systems],” in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed. BSUIR, Minsk, 2018, pp. 119–132.

[2] V. Golenkov, N. Guliakina, I. Davydenko, and A. Eremeev, “Methods and tools for ensuring compatibility of computer systems,” in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed. BSUIR, Minsk, 2019, pp. 25–52.

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

[4] (2021, Jun) IMS.ostis Metasystem. [Online]. Available: https://ims.ostis.net

[5] V. Martynov, *Semiologicheskie osnovy informatiki [Semiological Foundations of Informatics]*. Minsk: Nauka i tekhnika [Science and technics], 1974.

[6] ——, *Universal’nyi semanticheskii kod (Grammatika. Slovar’. Teksty) [Universal Semantic Code (Grammar. Dictionary. Texts)*. Minsk: Nauka i tekhnika [Science and technics], 1977.

[7] ——, *Universal’nyi semanticheskii kod: USK-3*.

[8] A. Hardzei, *Theory for Automatic Generation of Knowledge Architecture: TAPAZ-2. Transl. from Rus. I. M. Boyko. Rev. English edn.* Minsk: The Republican Institute of Higher School Publ., 2017.

[9] ——, “Plagiarism problem solving based on combinatory semantics,” in *Open Semantic Technologies for Intelligent System*, V. Golenkov, V. Krasnoproshin, V. Golovko, and E. Azarov, Eds. Cham: Springer International Publishing, 2020, pp. 176–197.

[10] A. Fayans and V. Kneller, “About the ontology of task types and methods of their solution,” *Ontology of designing*, vol. 10, no. 3, pp. 273–295, Oct. 2020. [Online]. Available: https://doi.org/10.18287/2223-9537-2020-10-3-273-295

Онтологический подход к разработке гибридных решателей задач интеллектуальных компьютерных систем

Шункевич Д.В.

В работе рассмотрен онтологический подход к разработке решателей задач интеллектуальных компьютерных систем на основе Технологии OSTIS. Уточнена формальная трактовка таких понятий как действие, задача, класс действий, класс задач, метод, навык, что в совокупности позволило определить на их основе понятие модели решения задач и решателя задач. Полученные результаты позволят повысить эффективность компонентного подхода к разработке решателей задач и средств автоматизации разработки решателей задач.

Received 01.06.2021