Principles of Organizing the Semantic Ontology of Programming

V.N. Kuchuganov, D.R. Kasimov
Department of Automated Data Processing and Control Systems
Kalashnikov Izhevsk State Technical University
Izhevsk, Russia
kuchuganov@istu.ru, kasden@mail.ru

M.V. Kuchuganov
Parallel Computation Lab
Udmurt State University
Izhevsk, Russia
qmikle1@yandex.ru

***Abstract*—A promising way to improve the programming process is increasing the declarativeness of programming, approximation to natural language on the base of accumulating and actively using knowledge. The essence of the proposed approach is representing the semantics of a program in the form of a set of concepts about actions, participants, resources and relations between them, accumulation and classification of machine-readable knowledge of computer programs in order to increase the degree of automation of programming. The novelty of the presented work is retrieving relevant precedents by the given description of resources and actions under them. The paper describes the ontological model of a program, considers the technology of programming using ontologies, gives an example of definition of the semantics of a problem, and discusses the positive aspects of the proposed approach.**

Keywords—programming automation; process ontology; semantic model of a program; attributed graph; action; participant; resource; selection

I. INTRODUCTION

A promising way to improve the programming process is increasing the declarativeness of programming, approximation to natural language on the base of accumulating and actively using knowledge. Technologies of knowledge-based programming are still in their infancy, but even with significant limitations, their application in practice can provide an obvious benefit, in particular, in the field of teaching programming.

Ontologies have become actively used in the tasks of software management [1] and teaching programming languages and technologies [2, 3, 4]. At the same time, ontologies used in these areas do not contain knowledge of particular programs; they represent only general concepts from the sphere of software development (data formats, control structures, OOP conceptions, standard libraries, platforms, etc.).

In [5] it is proposed to map a program code onto an ontology in order to simplify the comprehension of programs. The Java code is described by the graph, in which vertices are identifiers (names of variables, functions, classes, etc.), and edges are relationships between them in accordance with the modular structure ("memberOfPackage", "memberOfClass") and the type system ("subTypeOf", "hasType", "assignedTo", "boundWith"). The ontology-dictionary WordNet is used, which contains words connected by the relationships of hyponymy/hypernymy and meronymy. The ontology is also represented as a graph. The mapping of the code to the ontology reduces to finding a subgraph homomorphism between the graphs. During the matching, the compatibility of string names in vertices as well as the compatibility of relations encountered in paths between vertices are analyzed. The problem of incompleteness of the knowledge base is solved with the help of the user, who iteratively refines the results of the comparison and adds missing knowledge.

The work [6] offers the technology of declarative analysis of programs using ontology. In the ontology, experts describe the hierarchy of concepts used in the analysis of programs. Knowledge is extracted from a program by a special translator and is entered into the OWL knowledge base in the form of a set of triplets (subject, property, object). A user or a software agent can then ask logical questions about the program's properties of interest (for example, which cycles are canonical, what the variable is referring to, etc.). Answers are provided by the logical inference system (Prolog or some DL-reasoner).

The domain-specific metalanguage Magic Potion [7], which is based on the popular general-purpose programming language Clojure, is designed for modeling business domains using the ontological approach. Staying within the standard Clojure syntax, a programmer can describe a business task in the form of statements in the description logic, defining concepts, properties, roles and restrictions. Such seamless integration with the ontological paradigm became possible due to the presence of advanced metaprogramming tools in the language Clojure. Due to the fact that the foundation of the language is the description logic, it becomes possible to analyze the generated code by conducting automatic reasoning on the described models. For application software developers, the language Magic Potion looks more convenient than the available languages and tools for creating ontologies (OWL, Protégé, etc.).

The programming environment OLP [8] integrates logical programming with ontological reasoning. An OLP program looks like a Prolog program in which, in addition to standard predicates defined in the program itself, ontological predicates defined in a separate OWL-DL ontology can also be used. Ontological predicates have a special syntax for writing, which makes it easy to recognize them in the text of the program. The execution of the OLP program takes place in much the same way as the standard Prolog program with the only difference

978-1-5386-3592-6/17/$31.00 ©2017 IEEE

being that for computing an ontological predicate a DL-reasoner is called, which analyzes facts and relationships in the ontology.

The essence of our approach is representing the semantics of a program in the form of a set of concepts about actions, participants, resources and relations between them, accumulation and classification of machine-readable knowledge of computer programs in order to increase the degree of automation of programming. The novelty of the presented work is retrieving relevant precedents by the given description of resources and actions under them.

II. Semantic Models of a Program

Let's represent a program as a virtual model of a certain process that transforms the initial situation into a target one by performing allowed actions with provided resources (participants in the actions).

We distinguish 3 varieties of elements from which models of situations [9] are composed, and represent them in the form of symbols (Fig. 1). The semantic model describing a situation (SeMS) is constructed with the help of the plex-grammar [10], [11], in which symbols of grammatical constructions have not two (left, right), but N "points of contiguity".

Actions are connected with elements through the points of contiguity responding to questions: *(who), {from what}, {by what}, (where), (how), (what), (how many), (to whom), (for what), (when)*. Brackets mean that several symbols of the same sort can be associated with this symbol. For example, an action may require several material names (from what) and tools (by what).

Subjects are associated with other elements in accordance

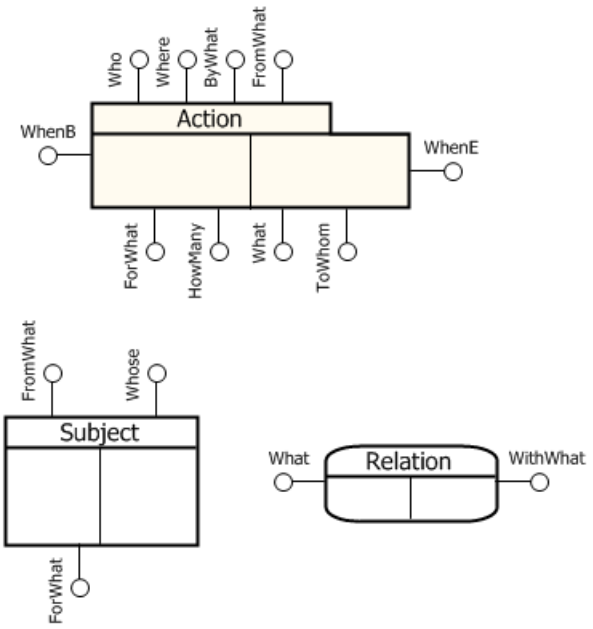

Fig. 1. Symbols of elements of semantic models of situations.

with the questions: *(from what), (whose), {for what}*. A subject has several purposes (for what) – various actions in which it can be used.

A relation connects two or more entities by a formula.

III. Ontology of Actions, Resources and Relations between them

For the description of the syntax and semantics of precedents, we use the description logic *ALC(D)*, which is created on the base of the logic *ALC* (Attributive Concept Language with Complements) [12], [13]. Let's divide the set *AF* of atomic abstract attributes of the ontology into two subsets:

$$AF = PAF \cup EAF,$$

where:

PAF – private, own qualities;

EAF – external, situational attributes.

Own qualities are inherent to each object from the nature. By areas of science these are physical, chemical, etc. properties. For example: *hasWeight*, *hasDensity*, *hasSpeed*, *hasDuration*, *hasCost*.

Situational attributes give additional characteristics to an object, describing its relationships with the external environment changing from situation to situation. For example, *hasDiploma*, *hasHouse*, *hasWife*, *hasNeighbors*.

As a basic concept, we introduce the abstract non-empty finite (generally infinite) set *Thing* of entities that covers all atomic concepts *CN*:

$$CN \subseteq Thing.$$

Meta-concept Subject from all sorts of individuals distinguishes the set of individuals that are represented by a geometric model (drawing, image, kinematic scheme, map, 3D geometric model, screen form, etc.), have private (own) qualities and external situational attributes and have the ability (for artificial objects can be considered as the purpose) to participate in some actions [14]:

$$Subject \equiv Thing \sqcap \exists hasModel.GModel \sqcap \exists hasQuality.PAF \sqcap \exists hasAttribute.EAF \sqcap \exists hasPurpose.Action$$

Subjects differ among themselves by geometric models, their own qualities and external attributes that depend on the purpose (artificial objects), or vice versa, they find some application due to their inherent qualities (natural objects).

The concept-subject describes a category of objects that exist in a static state, i.e. don't change (within certain limits) during the time interval considered in a problem, unlike processes described below, which exist in dynamics.

Meta-concept Process. As already mentioned, processes exist in dynamics. During a process something is always changed. Specifically, qualities are changed including spatial characteristics of objects. For example: aging, heating, movement.

Unlike subjects, processes have an algorithmic model (formula, algorithm or program). The purpose of a process is changing something:

$$Process \equiv Thing \sqcap \exists hasModel.AModel \sqcap \exists hasQuality.PAF$$

Meta-concept Relation. In the logic *ALC* binary and n-ary relations are defined as roles of two or more concepts in some connectives. For clarity of the ontology and for the convenience of working with it, it is useful to predetermine typical semantic categories of relations between instances in a studied domain.

We define the meta-concept relation as follows:

$$Relation \equiv Thing \sqcap \exists hasModel.PModel \sqcap \exists hasType.TypeR \sqcap \exists hasQuality.PAF \sqcap \exists (\geq 2\ hasAttribute.EAF)$$

In the interpretation it means that there is a relation *TypeR* between subjects specified in the situational attributes *hasAttribute*, which has the specific value *hasQuality*. The relation has a predicate model – a statement written in the natural language or in terms of the first-order predicate logic.

Entities-relations $Relation \sqsubseteq Thing$, like other categories of entities, have a model and properties.

Meta-concept Action binds a process and a system of procedural-role relations with participants:

$$Action \equiv Process \sqcap \exists hasPrecondition.Relation \sqcap \exists hasPostcondition.Relation \sqcap \exists\ hasActor.\ Subject \sqcap \exists hasRecipient.Subject\ \sqcap \exists hasObject.Subject \sqcap \exists hasTool.Subject \sqcap ...$$

Here an algorithmic model of a process serves as a model of an action. Participants of an action are: actor (performer), beneficiary – a customer in whose interests the action is performed; recipient – a receiver of the action (for example, "Vasya gives an apple Kate", the recipient is Kate); subject of impact (in the above example – an apple); scene of the action; instrument; co-agent; effect; etc.

The ontological knowledge base accumulates precedents of actions and their participants. For example, results of the action "Dig" depend on the available resources: *who* (Ivanov or Petrova), *what* (ditch or garden bed), *by what* (trencher or shovel), etc.; the effectiveness of the action "Cross" depends on the way (by boat, ferry, powerboat) and the capabilities of the selected resource. Accordingly, program implementations of actions of the same name can be different, but can also be the same and only values of their parameters change.

Another problem that the ontology helps to solve is the choice of the way of allocating/grouping a resource. Methods for selecting resources and their program implementations make a choice of a subset from a set of subjects (randomly, successively, binary, by ranking, by specified rules and restrictions), division of a resource into parts (equally, arbitrarily, on demand, to limit).

Fig. 2 shows the semantic model of the action "Cross" in the situation described in the famous problem of missionaries and cannibals. In the ontological knowledge base, semantic models of situations and actions are stored in the form of attributed graphs.

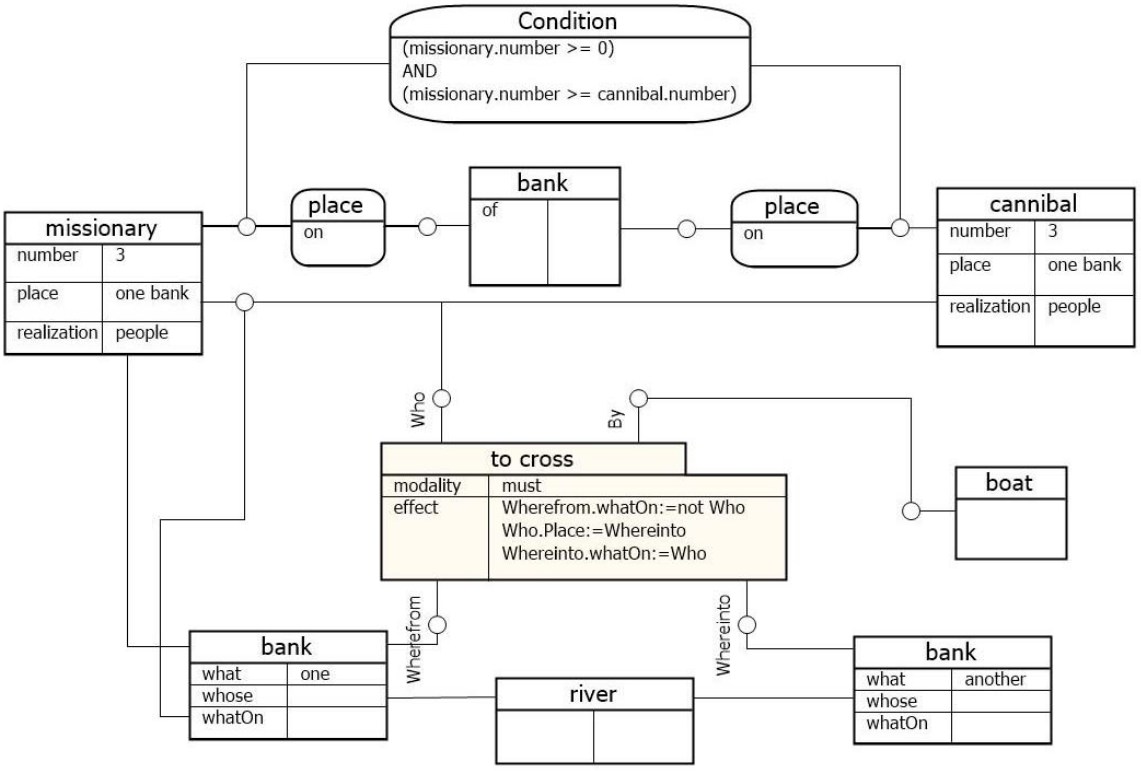

Fig. 2. Semantic model of the precedent "Cross" in the missionaries and cannibals problem.

IV. Technology of Programming Using Ontologies

Fig. 3 shows a standard scheme for analyzing the admissibility of actions p_i for transforming the initial situation into a target one (Ψ).

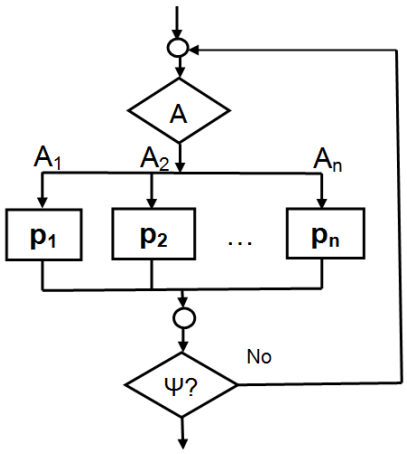

Fig. 3. Standard scheme for the analysis of the admissibility of actions.

Having the ontology of precedents describing actions, resources, and relations between them, we can insert a resource selection operator for each action p_i before executing it, based on the current situation. Then the typical program is:

Loop until the target state is reached

Select an action

Select resources

Perform the action

Check the state

Loop end

In general, a program is a combination of such blocks.

Thus, with the help of the ontology, we can select from the knowledge base suitable program implementations: functions, procedures, subroutines, which are stored in the knowledge base as algorithmic models of concepts. We also can find analogues and on their basis make new implementations.

Example. Consider the well-known problem "Monkey and Banana" [15]. Near the door of a room is a monkey. In the middle of this room a banana is suspended from the ceiling, beyond the monkey's reach. Near the window of the room on the floor is a box, which the monkey can use. The monkey can take the following actions: walk on the floor, climb on the box, move the box and grab the banana. How does the monkey get the banana?

The following possible actions are described in the problem statement:

1) grab the banana – if the monkey is on the box in the middle of the room and does not have the banana;

2) climb on the box, if the monkey is on the floor near the box;

3) move the box from one permitted place to another, if the monkey is on the floor near the box;

4) move on the floor from one permitted place to another.

Obviously, the key (target) action is the action "Grab the banana"; other actions transform the situation so that grabbing can be executed.

Fig. 4 presents a formalized description of the problem. It was performed in the intelligent problem solving environment SDL Solver [16].

№	1 Name	2 Length	3 Width	4 Height	5 Model	6 PlaceXY	7 PlaceZ
1	Room	560	300	220	2D-Array		
2	Monkey			140		(NearNorthernEdge, InCenter)	OnFloor
3	Banana					InCenter	AtCeiling
4	Box	80	60	80		(NearSouthernEdge, InCenter)	OnFloor

(a)

№	1 Name	2 Who	3 What	4 WhereTo	5 Precondition
1	Grab	Monkey	Banana		Place(Monkey) = Box; PlaceXY(Monkey) = PlaceXY(Banana); ¬ Have(Monkey, Banana);
2	MoveTo	Monkey			Beside(Place(Monkey), WhereTo); Free(WhereTo);
3	ClimbOn	Monkey		Box	PlaceZ(Monkey) = OnFloor; Beside(Place(Monkey), Place(Box));
4	Push	Monkey	Box		PlaceZ(Monkey) = OnFloor; Beside(Place(Monkey), Place(Box)); Beside(Place(Box), WhereTo); Free(WhereTo);

(b)

№	1 Name	2 Have
1	Monkey	Banana

(c)

Fig. 4. Description of subjects (a), actions (b) and the target situation (c) of the problem.

Here parameters of the subjects are more specifically given. This is required by program implementations that transform situations in solving practical problems and, at the same time, narrow the search for precedents.

Searching for precedents in the knowledge base of the ontological type is carried out by two ways:

1) Action (Subjects, Relations);

2) Situation (Subjects, Actions, Relations).

In the first case, actions with similar participants and restrictions are sought. The results of the search are sorted in descending order of the number of matched participants and the number of matched restrictions. Coincidence of the name of the action is not necessary.

In the second case, an attempt is made to find similar situations, when the solving process is divided into blocks – the reference points of the overall strategy. For example, *Move(WhereFrom, WhereTo)* through the cells of the field with

obstacles will be considered as a subroutine (cyclic action, work) within the overall strategy of solving the problem.

According to our description of the problem, the following precedents of actions have been found in the knowledge base:

1. Take, Grab.
2. Climb on.
3. Approach to, Move to.
4. Push.
5. Raise.
6. Put on.

In addition, precedents of other actions have been revealed, which can provide the action "Grab" by reducing the distance between the monkey and the banana using available resources: *Bounce*; *Shoot down (ByWhat = Subject)*.

V. CONCLUSION

The ontological knowledge base provides accumulation of program implementations, their semantic classification and searching for similar precedents "actions, resources → implementation".

In the first approximation, the search is similar to searching by keywords; in this case these are the names of attributes of actions and resources (subjects). The difference is that attributes in the ontology of concepts are related to each other by role relations and the correctness of the terminology is controlled by the logical inference module (reasoner).

Thus, the main principles of organizing the ontology of programming in the proposed approach are:

- semantic modeling of the problem description: actions, resources, restrictions;
- ensuring the relevance of results of searching for program implementations by semantic models of precedents.

Semantic models of actions and resources make the process of structuring and algorithmization of a problem more visible and at the same time machine-readable.

In our opinion, the active use of ontologies of programs is useful both at the stage of teaching programming, and during the development of application software in various domains.

REFERENCES

[1] P. Fitsilis, V. Gerogiannis and L. Anthopoulos, "Ontologies for Software Project Management: A Review," Journal of Software Engineering and Applications, Vol. 7, No. 13, pp. 1096-1110, 2014.

[2] S. John, "Development of an Educational Ontology for Java Programming (JLEO) with a Hybrid Methodology Derived from Conventional Software Engineering Process Models," International Journal of Information and Education Technology, Vol. 4, No. 4, pp. 308-312, 2014.

[3] I.A.O. Abuhassan and A.M.O. AlMashaykhi, "Domain Ontology for Programming Languages," Journal of Computations & Modelling, Vol. 2, No. 4, pp. 75-91, 2012.

[4] A. Kouneli, G. Solomou, C. Pierrakeas and A. Kameas, "Modeling the Knowledge Domain of the Java Programming Language as an Ontology," In: E. Popescu, Q. Li, R. Klamma, H. Leung and M. Specht (eds), Advances in Web-Based Learning - ICWL 2012. ICWL 2012. Lecture Notes in Computer Science, Vol 7558, pp. 152-159, Springer, Berlin, Heidelberg, 2012.

[5] D. Ratiu and F. Deissenboeck, "Programs are Knowledge Bases," 14th IEEE International Conference on Program Comprehension (ICPC 2006), 2006.

[6] Y. Zhao, G. Chen, C. Liao and X. Shen, "Towards Ontology-Based Program Analysis," 30th European Conference on Object-Oriented Programming (ECOOP 2016), 2016.

[7] D. Djuric and V. Devedzic, "Incorporating the Ontology Paradigm into a Mainstream Programming Environment," Informatica, Vol. 23, Is. 2, pp. 203-224, 2012.

[8] M. Şensoy, G. Mel, W.W. Vasconcelos and T.J. Norman, "Ontological logic programming," Proceedings of the International Conference on Web Intelligence, Mining and Semantics (WIMS '11), Article No. 44, 2011.

[9] V.N. Kuchuganov, "Elements of Associative Semantic Theory," Upravlenie bolshimi sistemami, Is. 40, pp. 30-48, 2012. (in Russian).

[10] J. Feder, "Plex languages," Information Sci., No. 3, pp. 255-241, 1971.

[11] K.S. Fu, Syntactic Methods in Pattern Recognition, New York and London: Academic Press, 1974.

[12] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P.F. Patel-Schneider (editors), The Description Logic. Handbook, Cambridge University Press, 2003.

[13] C. Lutz, "Description Logics with Concrete Domains - A Survey," In P. Balbiani, N.-Y. Suzuki, F. Wolter and M. Zakharyaschev (eds), Advances in Modal Logics, 4, King's College Publications, 2003.

[14] V.N. Kuchuganov, "Ontology and animation of precedents," Ontologiya proyektirovaniya, Vol. 6, No. 3(21), pp. 287-296, 2016. (in Russian).

[15] J. McCarthy, "Situations, actions, and causal laws," In Semantic Proceedings of the tri-annual IFIP Conf, Minsky (ed), Machine Intelligence, eds: Meltzer and Michie, vars. PublishersT Press, 1968.

[16] V.N. Kuchuganov and D.R. Kasimov, "An Intelligent Environment for Learning Techniques and Strategies of Solving Combinatorial Problems," Proceedings of the III International Scientific Conference "Information Technologies in Science, Management, Social Sphere and Medicine" (ITSMSSM 2016), pp. 204-207, Atlantis Press, 2016.