The Ontological Programming Paradigm

Valeriya Gribova, Alexander Kleschev
Intelligent Software Laboratory
Institute of Automation and Control Processes,
Far Eastern Branch of RAS
Vladivostok, Russia
gribova@iacp.dvo.ru, kleschev@iacp.dvo.ru

***Abstract* — This paper is devoted to the new ontological programming paradigm, an evolution of the declarative programming paradigm. The data model, basic structures of the language, and example are described.**

Keywords - programming paradigm; data model; semantic network; ontology.

I. INTRODUCTION

Software development is a time consuming process. Even more difficult is software maintenance. They both require new approaches from developers to resolve these problems. The declarative programming paradigm and languages realized in this paradigm were proposed as an approach to the problem mentioned above.

In general, a program in a declarative language is a description of an abstract model of the task to be solved, or, in accordance with [1], an executing specification of a result of computation. The programmer does not have to describe a process to control computation; it is the function of the language processor. Among other advantages is the fact that it is easier to write, to understand, and to maintain programs using languages of this paradigm in comparison with those written in imperative languages.

Declarative languages comprise logical and functional languages. However, the main idea of declarative programming in the modern logical and functional languages has not been realized completely yet. As a result, programs written in Prolog [2] and Lisp [3] are not considerably easier to develop, understand, and maintain than programs written in imperative languages. Among other drawbacks of declarative languages are poor facilities for user interface realization, and the difficulty of including imperative operators, if necessary.

The aim of this report is to suggest a new programming paradigm, called the ontological programming paradigm, as a further evolution of the declarative programming paradigm and to suggest an ontological programming language (OPL) satisfying the declarative programming definition, where a program is an ontology of results of computation. Mechanisms for user interface realization and facilities for including imperative structures are proposed.

The paper has the following structure. The problem statement is discussed complication of processes of development, modification, and understanding a program in the imperative, functional, and declarative paradigms; the ontological programming paradigm and its basic principles are described. Then the data models of the paradigm, basic structures of the ontological programming language are suggested. At the end of the paper the expert system of medical diagnosis using OPL is presented.

II. PROBLEM STATEMENT

Output data of task solving are the result of a computation process. The complications of program writing for solving a certain task are the following: the developer has to understand a set of computation processes to obtain results (extension of a task) for various possible input data, and to specify this set in a programming language (to write a program) [4]. The complications of a program modification during the life cycle are the following: the maintainer must recover extension of the task and comprehend why the computation processes result in exactly these output data. Then he or she must understand how to change these processes in order to obtain new output data and then modify the program.

We will discuss how computation processes to obtain results are connected with programs for obtaining these results in the imperative, functional, and declarative paradigms.

The basis of the imperative paradigm is computational models [5, 6]. The process of obtaining results in these models is a sequence of states, the first of them is generated from input data using an input procedure, every follow-up state is generated from the previous one using an operator of direct processing; the terminal state gives the output result. All the states of the computation process, except for the terminal state, are only indirectly connected to the output result. In currently-used imperative languages, the state of a computation process is a set of variable values, and the operator of direct processing is the assignment operator. Using this operator, the next state is a modification of a variable value; the remainder variables keep their values. Sequences of operator execution in programs written in imperative languages are defined by linear fragments of the program, conditional operators, and cycle operators (and also a procedure call).

The basis of the functional paradigm is the lambda calculus [5, 6]. The process of obtaining the result can be represented by an oriented marked network of a function call. The label of every terminal vertex is input data, the label of every non-terminal vertex is a function value. Arguments of this function are labels of arcs outgoing from this vertex (the arc orientation is from the result to the argument). The computation result is the label of the network root. All temporary values (labels of non-terminal vertexes), except

 ISBN: 978-1-61208-217-2

for the root label, are indirectly connected to the output result. In current functional languages there is a set of basic functions and facilities for designing of new functions from basic and already defined (among them being conditional terms and recursion).

The basis of the logical paradigm is the first order predicate calculus [5, 6]. The process of obtaining result can be represented by an oriented marked network of result inference. The label of every terminal vertex is a relationship tuple representing input data; the label of every non-terminal vertex is a relationship tuple representing the result of applying a rule to premises. Premises are labels of arcs outgoing from this vertex (the arc orientation is from the consequence to premises). The computation result is the label of the network root. All temporary values (labels of non-terminal vertexes), except for the root label, are indirectly connected to the output result. A program in a logical language is an inquiry and a set of rules (implications) and facts (relationship tuples).

The computation result in each of the three paradigms is obtained at the last step of the computation process, all remainder steps are indirectly connected to the output result and are temporary values.

Thus, to simplify the program development and its understanding and modification, it is necessary to suggest a programming paradigm where processes of obtaining result, represented by oriented graphs are direct. It means that a fragment of a result is formed at the every step of the computation process. In this case a program is an executable specification of a set of results of computation (but not a set of indirect processes of obtaining them). Developing such a program the programmer must only specify a set of results of computation; analyzing such a program the programmer must only conceive a set of results obtained; modifying such a program the programmer must only understand how to change the specification of the set of computation results to obtain required changes of these results, and every changing is local. The specification of a set of results, according to the up-to-date view in the artificial intelligence is an ontology of computation results. So, we name the suggested paradigm the ontological programming paradigm. It is considered as a more complete realization of the declarative programming paradigm (functional and logical paradigms).

III. THE DATA MODEL OF THE PARADIGM

All data in the ontological programming paradigm are semantic networks. The semantic network is an oriented graph without cycles; all arcs of the network have labels, vertexes can be simple and structural; the network vertex without entering arcs is the root. Every simple terminal vertex of the network has a label. A label is a constant of a type. The root of the network has two labels. The first of them is the label of a class (the name of a function), the second is the individual label of this network. Every structural vertex is a container comprising an ordered set of hierarchical semantic networks with the same label of a class and different individual labels.

The semantic network may be stored constantly (out of the programs) or temporarily (within the program). Using semantic networks as a data model means that objects of processing are integrated information structures. For example, integrated information structures in the expert system of medical diagnosis are: a case report, a knowledge base, and an explanation represented in the form of semantic networks. It is the difference of the ontological paradigm from others that support processing informational structures divided into some fragments (for example, some objects). Relations between these fragments are only known to programmers.

IV. THE APPLICATION

In general, an application has one or some output semantic networks (results), may have/do not have one or some input semantic networks (input data), and also may have/do not have one or several temporary semantic networks (temporary data). Every temporary or the output semantic network is computed by a function (the name of the function is a label of a class of the semantic network root).

Every function may have/do not have input semantic networks and the only output semantic network and does not have any temporary networks.

Input semantic networks may be stored constantly (out of the application) or temporary (within the application), so output semantic networks of an application or a function may be stored both constantly and temporary.

Among all semantic networks of an application (input, output, and temporary ones) a partial order is defined. Therefore, an application is a superposition of these functions, connected with computation of all semantic networks of an application, except for input semantic networks stored constantly. Thus, application executing is computation of the output semantic network using input semantic networks stored constantly and created before application launching (if any). Every function of an application is an ontology (structure) of an output semantic network created by the ontological programming language.

V. THE ONTOLOGICAL PROGRAMMING LANGUAGE

The OPL is a visual logical language of programming. A function is the ontology of the output semantic network. It is an oriented graph with the marked vertex and arcs. Arc labels are terms which become arc labels of an output semantic network of a function in the process of application executing. A vertex label is logical formulas.

The computation process of an output semantic network of a function built this network starting from the root. During the process of semantic network building one-to-one correspondence between the vertex and arcs of the output semantic network and the vertex and arcs of its ontology (function) is determined. The output semantic network can be built as both automatic and interactive ones.

Figure 1. The simple formula

 ISBN: 978-1-61208-217-2

Logical formulas of the language are: a simple formula, a unary formula, a propositional formula, a simple quantifier formula, a structural quantifier formula and a set of implications.

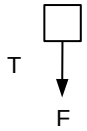

Figure 2. The unary formula

The simple formula (see Figure 1) is a graph comprising the only vertex with the label c. Label c is a constant of any type, or variable v, or variable value v*.

The unary formula (see Figure 2) is a graph comprising the initial vertex and the arc with the label T (term), going out of the initial vertex and coming into the initial vertex of a logical formula F.

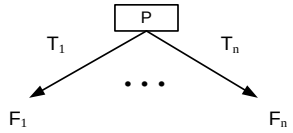

Figure 3. The propositional formula

The propositional formula (see Figure 3) is a graph comprising the initial vertex with the propositional label P and n arcs (two and more) going out of this vertex. Each of these arcs has the label Ti (term; i varies from 1 to n, and all of these arcs must be different) and coming into the initial vertex of a logical formula Fi. The propositional labels P are: & — conjunction, ∨ — disjunction, and | — XOR. The set of propositional labels is extended.

The simple quantifier formula (see Figure 4) is a graph comprising the only vertex with the label QMT, where Q is a quantifier, M is a set, and T is a term. The quantifier is ∀ (universal), ∃ (existential), ∃2 (existential not less than two), ∃? (existential but not for all), ∃! (existential and only), ∃ [] (existential subinterval).

The set is defined by values of the elements (constants), or the type of valid values, or an integer and a real interval, or a variable. The type is "string", "integer", "real", "integer interval ", "real interval ", and "data-time ". The set of types is extended. The determined set and the chosen quantifier define conditions and contingencies during the consequence process of a result fragment.

Figure 4. The simple quantifier formula

The structural quantifier formula (see Figure 5) is a graph comprising the only vertex with the label QMT, where Q is a quantifier, M is a set, and T is a term, and a logical formula F, the initial vertex of which is inside the initial vertex of the structural quantifier formula.

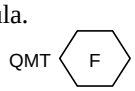

Figure 5. The structural quantifier formula

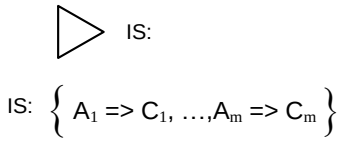

Figure 6. The set of implications

The set of implications (see Figure 6) is a graph comprising the only vertex with the label {A1 ⇒ C1, …, Am ⇒ Cm}, where A1, …, Am are antecedents and C1, …, Cm are consequents of implications. The antecedent of the implication is a finite set of components. They are logical formulas; each of them can have a prefix. The prefix is a name of an application written in the OPL. The consequent of the implication is a logical formula.

Variables declared in logical formulas can be only in antecedents and consequent of implications. The special labels of vertexes of unary formulas can be only in antecedents of implications. If a variable is in the consequent of the implication, it must be in antecedent of the implication.

For realization of complicated calculations computed predicates are added to the OPL. They are intended for describing calculations using operators of an imperative language (for example, Java). A computed predicate has name and a set of formal parameters.

Abstract interface commands define functional of a user interface. They are divided into four classes: output commands, input (edit) commands of a value of a type, input (edit) commands of a set of values of a type, choice commands of a subset from the set of values.

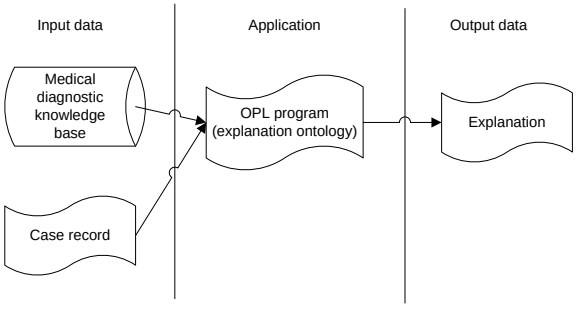

Figure 7. Architecture of the expert system of the medical diagnosis

Using the OPL the expert system of medical diagnosis is described (see Figure 7). Input data for the system are the knowledge base and a case record. The application (the program in OPL) is the semantic network of explanation (forming the explanation). The explanation consists of two parts: hypotheses explanation that a patient is healthy and hypotheses explanation that a patient suffers from a disease from the knowledge base (see Figure 8).

 ISBN: 978-1-61208-217-2

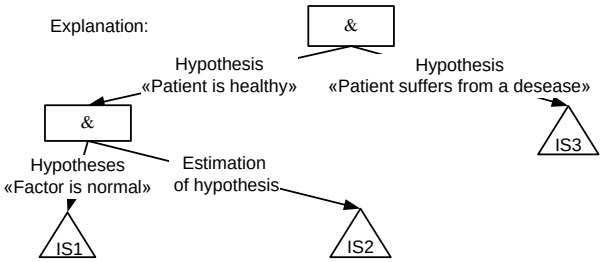

Figure 8. Explanation of the expert system of the medical diagnosis

Hypotheses explanation that a patient is healthy consists of hypotheses explanation that all observed factors of the patient is normal and hypotheses estimation that the patient is healthy.

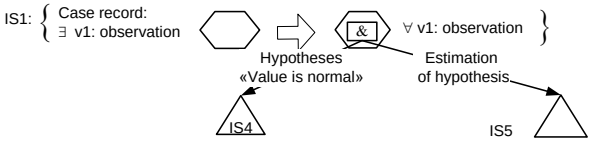

Figure 9. Description of "Factor is normal" hypothesis

Implication IS1 (see Figure 9) forms variable value v1. It is a set of all observed factors in the case record. Hypotheses explanation consists of two parts. The first part is hypotheses explanation that all observed values of this factor is normal. The second part is hypotheses estimation that this factor is normal. Similarly to IS1 implications IS2, IS3, IS4, and IS5 are described.

VI. CONCLUSION

The new paradigm is intended for reducing labor-intensiveness of development and maintenance of intelligent systems. The main idea is to describe an ontology of results using the visual logical language of programming. The programmer does not have to describe a process of obtaining result; it is the function of the language processor. All data in the ontological paradigm are semantic networks. The language has facilities for user interface realization and for including imperative structures.

ACKNOWLEDGMENT

The research was supported by the Russian Foundation for Basic Research, the project 12-07-00179-a and the Far Eastern Branch of Russian Academy of Science, the project 12-I-ОНИТ-04

REFERENCES

[1] J. Lloyd Practical advantages of declarative programming. In: Proceedings of the 1994 Joint Conference on Declarative Programming, GULP-PRODE'94, Springer Verlag, 1994, Vol. 94, pp. 1-15.

[2] I. Bratko Prolog programming for artificial intelligence. Harlow, England ; New York: Addison Wesley, 2001, 678 p.

[3] C. Orlov Technology of software development – SPb: Piter, 2002, 464 p. (in rus.)

[4] UspenskiyV.A. Semenov A.L. The theory of algorithms. Fundamental discoveries and applications - M.: Nauka, 1987, 288 p.(in rus.)

[5] Sebesta R.W Concepts of Programming Languages. - AW: 2001, 672 p.

[6] Floyd R.W. The Paradigms of Programming// Communications of the ACM, 1979, 22(8), pp. 455-460.

 ISBN: 978-1-61208-217-2