PROCESSING OF KNOWLEDGE ABOUT OPTIMIZATION OF CLASSICAL OPTIMIZING TRANSFORMATIONS

Irene L. Artemjeva, Margarita A. Knyazeva, Oleg A. Kupnevich

Abstract: The article describes the structure of an ontology model for Optimization of a sequential program. The components of an intellectual modeling system for program optimization are described. The functions of the intellectual modeling system are defined.

Keywords: Knowledge based system; Program optimization; Domain ontology

Developing knowledge-based systems for any domain needs constructing its ontology [Kleshchev, 2002]. Ontology is an explicit description of domain notions and contains terms for describing reality and knowledge and agreements restricting the interpretations of these terms. The ontology of a domain defines the structure of knowledge and the structure of domain reality.

The problems in the "Program optimization" domain are mainly grouping around the unification of the notion system for describing program schemes in terms of which one could describe optimizing transformations and standardization of notion system for describing optimizing transformations. The other set of problems has to do with how to effectively use the accumulated knowledge in the problematic area, i.e. is connected with the special training in program optimization, the development of skills of putting theoretical knowledge about program optimization into practice.

The ontology model of the knowledge domain "Sequential program optimization" and its using when developing computer knowledge banks on program optimization is presented in this work.

The ontology model of the "sequential program optimization" domain

The formal description of the terminology of a knowledge domain together with definition of meanings of terms is called "ontology model" [Kleshchev, 2001]. The terms of the knowledge domain "Program optimization (classical optimizing transformations)" can be divided into two groups: (i) the terms for describing programs (the terms of this group will be called the terms for describing the optimization objects), and (ii) the terms for describing the optimization process. Therefore the ontology model of this knowledge also consists of two parts.

The optimization object is a program. The characteristics of a program are always formulated in terms of a mathematical model of this program. The characteristic of a program is the language (a set of programs) this program belongs to. The characteristics of the language are also formulated in terms of a mathematical model of this language. Thus, a number of terms for describing the object of optimization can be divided into two groups: (i) the terms for describing the language model, (ii) the terms for describing the program model. Before optimizing a program, the language, this program belongs to, must be determined. Consequently, the terms for describing the language model are parameters of the ontology model, and the terms for describing the program model are the unknowns of the ontology model. Then the language model is represented by the values of the parameters, and the program model – by the values of the unknowns.

It is evident that the program model describes not one program but a set of programs that have the same characteristics; the language model describes a set of languages that have the same characteristics. Therefore the following requirements are set on the language model: it must allow to present basic constructs of imperative programming languages essential for describing sequential OTs; it must be flexible and extensible in order to expand the class of modeled programs, if necessary; the form of presenting program models must be convenient for analyzing both information flows and control flows in the program.

The ontology model of the knowledge domain "Sequential program optimization" consists of two modules. The first module contains the terms for describing the optimization object, the second one – the terms for describing the optimization process.

The first module is an unenriched system of logical relationship with parameters written in sentences of a many-sorted language of the applied logic. Any program consists of fragments that in their turn consist of

other fragments [Kasyanov, 1988] [Knyazeva, 1999], i.e. any program has its syntactic structure that is reflected by a mathematical model of this program.

Each fragment – as an element of the program – has a number of characteristics. First of all, it has *the address of the fragment* – the unique characteristic unambiguously defining each fragment in the program.

All the fragments can be divided into three groups: declarative statements, imperative statements, and entries of statements. Its class characterizes each fragment, e.g. the fragment can have the class of assignment statement, iteration statement, declarative statements of functions, etc. The function *FragClass* returns the class of this fragment for the indicated address of the fragment. A set of names of fragment classes (of each group) in the program assigns the values of the parameters D*eclarative statements, Imperative statements, Entries of statements.*

Control arcs assign the syntactic structure of the program. Each control arc connects two fragments of the program. Each control arc has its label identifying a type of connection between the fragments. The function the name of which coincides with the arc label is used to assign the control arc. The value of the parameter *Names of control arcs* assigns what labels can be owned by control arcs in the program. The control arc area is defined for each control arc. This area is assigned by the value of the functional parameter *Control arc area* – a function that matches each arc label with a pair consisting of two sets of names of fragment classes: the first set determines what fragments classes can be arguments, the second one determines what fragments classes can be results of the control arc with this label.

There are always a number of various *Identifiers* in the program. Identifiers can be of different types, e.g. identifiers of variable, functions, constants, and data types. In the program identifiers of each type make a set the name of which coincides with the name of the type. The value of the parameter *Types of identifiers* assigns what types of identifiers can be present in the program.

Functions and relationships identifying the structure of the program and some of its characteristics are defined on a set of fragments and identifiers of the program.

Functions with one argument (a fragment or an identifier) are called attributes. Each attribute has its name. The value of the parameter *Names of attributes* assign what attributes can be used for describing the characteristics of identifiers or fragments in the program. The definitional domain and the range of attribute values are established for each attribute. They are assigned by the values of the functional parameters *Definitional domain* and *Range of attribute value.*

The value of the parameter *Names of functions* assigns what functions with two or more arguments can be used for describing the characteristics of fragments or identifiers of the program. The definitional domain and the range of values are established for each function. They are assigned by the values of the parameters *Definitional function domain* and *Range of function values.*

The value of the parameter *Names of relationships* assigns what relationships can be defined on a set of fragments and identifiers of the program. The values of the parameters *Determination of Relationship* for each relationship name assign a formula of truth determination of a relationship between fragments and identifiers of the program.

Correctness is another characteristic of the fragment. The value of correctness is assigned by the predicate *Correctness* that matches the address of the fragment with the truth if it has all the control arcs and attributes necessary for this fragment. The value of the parameter *Determination of correctness* assigns control arcs and attributes for each class of fragments.

The value of the parameter *Elementary types* assigns for the language a set of identifiers of data types that are basic for all the rest data types of this language.

The value of the parameter *Modes of generation* assigns a set of names of constructors of data types in the language. The values of such characteristics as a base type and the method of constructing a type from the base one are set for each identifier of data type in the program. The value of the first characteristic is assigned by the function *BaseType* that matches each identifier of the type with a chain of identifiers of types that are used when constructing this data type. The value of the second characteristic is assigned by the function *ConstructMethod* that matches each identifier of the type with the name of the constructor. The value of the parameter *Compatibility of types* is a predicate defining the possibility of implicit transformation of one type to another.

The value of the parameter *Reserved names* assigns a set of names of roles that can be played by fragments or identifiers of the program. The value of the parameter *Area of reserved name value* matches each name of a role with predicate defining proprieties for a fragment or an identifier playing this role in the program.

The value of the parameter *Operators* assigns a set of symbols used in the program for identifying operations in expressions (arithmetic, logical, of a transformation, etc.) of the program. The functional parameters *Arity* and *Priority* match each symbol of the operation with a number of arguments and priority.

Optimization is understood as a chain of steps at each of them one transformation is applied to an optimized program. Each step will be called a step of the optimization history. The program model written in terms of the language model is the optimization object. The program model changes at each step of optimization: the rule of transformation that is established by the optimizing transformation that is used at the current step of optimization is applied to it. Thus, at each step of optimization there is its own version of the program model with its set of fragment addresses, its set of identifiers, etc. Therefore all the terms defined in the work [Artemjeva, 2002] are functions the first argument of which unambiguously identifies the current program model (i.e. the current set of fragment addresses, the current set of identifiers, etc.). A number of a step of the optimization history plays a role of this argument.

Let us define the terms for describing optimization process. Each optimizing transformation has the following characteristics: the saving block, the context condition, the indicative function, and the application strategy.

Normally an optimizing transformation (OT) is not applied to the whole program but to one of its blocks not necessarily continuous. The set of program fragments necessary for making a decision about optimization will be called candidate for saving blocks and the set of fragments where the context condition is true will be called saving block. Thus the saving block is a candidate for which the context condition of an optimizing transformation is true.

An ordinary saving block is a fixed set of program fragments for which the number of fragments, their types and their mutual location in the program are known. However the saving block consists of two parts. The first part is a tuple of fragments where each element's type and location in the saving block are known.

The second part is a set of tuples of fragments. The number of tuples belonging to the second part of the saving block is variable for different saving blocks of one program but the number of elements of each tuple, types and mutual location of fragments included in the chain are fixed, i.e. these tuples of fragments own certain identical characteristics. When applying an optimizing transformation, all the tuples of fragments of the second part of the saving block change the same way. For example, the saving block contains both declarative statement of a procedure and all its invocations for certain optimizing transformations of procedures (functions). The declarative statement of a procedure is always single but invocation operators can be numerous. All the elements of the set of invocations are assigned by declarative statement of a procedure. Thus, after applying an optimizing transformation, all operators of invocation of a procedure (function) change the same way.

The saving block in the model is represented as a pair the first element of which will be called the simple part of the saving block; the second one will be called the multiple part of the saving block. The first element of the pair is a tuple of addresses of fragments. The second element of the pair is a set of tuples of addresses of fragments. For the example given above, fragments of declarative statement of a function make the simple part of the saving block and a set of tuples of fragments of invocation operators makes the multiple part of the saving block.

Each optimizing transformation has simple and multiple parameters. Each fragment of the simple part of the saving block will be called the value of the simple parameter of the saving block corresponding to it.

The set of fragments included in the set of tuples of fragments of the multiple part of the saving block will be called the value of multiple parameter of the saving block. The function *Number of simple parameters of OT* assigns the number of elements of the tuple of the simple part of the saving block. The function *Number of multiple parameters of OT* assigns the number of elements of each chain of the multiple part of the saving block. The functions *Classes of simple parameters of OT* and *Classes of multiple parameters of OT* assign chains of classes of fragments forming the simple part of the saving block and chains of classes of fragments forming each tuple of the multiple part of the saving block.

The context condition of the optimizing transformation describes the characteristics of the saving block this optimizing transformation is applied to. In this ontology model the context condition is presented as a predicate the arguments of which are the number of a step of optimization history and a candidate to saving blocks.

There can be several saving blocks in each step of the optimization process. Therefore it is necessary to have a criterion to choose one block form many. *Indicative function* (IF) is a formula the arguments of which are fragments from the saving block. It matches each saving block with its estimate – the rational number.

Optimization strategy is a formula helping to choose one number from the set of rational numbers – results of indicative function for various saving blocks. Correspondingly the saving block with the chosen characteristic undergoes optimization on this step of history.

Parameters with the identical context are a set of pairs of numbers of parameters of the saving block before and after optimization contexts of which must coincide.

The chain of application of OT assigns the order of OT application. Each OT is applied until there are valid saving blocks left for it.

All the terms defined above are considered as terms for describing knowledge of program optimization, whereas optimization strategy, indicative function and chain of application of OT allow assigning parameters of optimization; context conditions and transformation formulas assign statistic knowledge about optimizing transformations.

Let us further define terms for describing situations of the knowledge domain. These terms are mainly meant for complete recording of the modeling process of application of optimizing transformations to the program.

The situation consists of a chain of optimization steps (history steps). Sets of fragment addresses, identifiers, and values of all attributes, functions and relations in the program are defined for each step. Some fragments, attributes, arcs and identifiers are defined while constructing the program model and known on the first step of optimization but some are recomputed before the beginning of each next step with the help of a special function *Enrichment of SMP*. The work of this function leads to that all the DSCH class fragments are added to the program model, Begin, End, Parent, SCH arcs are defined for them, all relations and functions are defined, values of all the attributes from a set of *Computable* attributes are also computed.

At each history step there is *a chain of current OT*. At the beginning of optimization a chain of *Current OTs* coincides with a chain of OT application.

The first OT from a chain of current ones for which a set of candidates to the SB is not empty is called *applied OT* and its number in the application chain is written in *Number of Applied OT*. A set of estimates is formed for all candidates to SB for applied OT with the help of indicative function and one estimate is selected from this set and with the help of optimization strategy and becomes the *characteristic of chosen SB*. The saving block corresponding to this characteristic becomes *chosen SB*. Apart from this, *optimized saving block* is defined on each step – it is a combination of fragments from next history step that becomes true when permutated to the transformation formula together with chosen SB. Optimized block appears as a result of OT application to the chosen saving block.

As a result of an applied optimizing transformation a number of fragments of the source saving block can be deleted, a number of fragments can be changed, new fragments (with new addresses)can be added, existing identifiers can be deleted or new ones can be added. For each optimizing transformation it is known how many elements will be included in the simple part of optimized saving block and how many elements will be included in each element of the multiple part of the saving block. This information is assigned by functions *Number of elements of the simple part of optimized SB* and *Number of elements of multiple part of optimized SB.*

Transformation formula is a predicate the arguments of which are two saving blocks from two consequent history steps; this predicate is true if the second SB is the result of OT application to the first SB.

Tasks given in terms of ontology model

As it follows from the previous section, in the ontology model there are groups of parameters defining Programming Language (PL), Optimizing Transformation Description Language (OTDL), Optimizing Transformations (OT); Optimization Strategy and groups of unknowns defining program characteristics before optimization, complete protocol of optimization process and characteristics of optimized program. Besides it is necessary to enter parameter Estimating function. This function will allow comparing various programs and, thus, to estimate optimization results.

Let us mention main classes of tasks that can be specified in terms of ontology model:

1. given PL, OTDL, an optimizing transformation and a program, required to get an optimized program and check the correctness of OT application;
2. given a program, PL and OTDL, strategy, estimating function, required to get an optimized program, estimate its optimality, check the correctness of OT, Strategy, analyze protocol;

3. given a set of programs, PL and OTDL, strategy and estimating function, required to get a set of optimized programs and estimates of their optimality, study the dependence of program optimality estimate on its characteristics;
4. given a set of programs, PL and OTDL, a set of strategies, estimating function, required to get a set of optimized programs and their optimality estimates, study the dependence of program optimality estimate on its characteristics and applied strategy;
5. given a set of programs, PL and OTDL, a set of strategies and estimating functions, required to get a set of optimized programs and their optimality estimates, study the dependence of program optimality estimate on its characteristics and applied strategy for various estimating functions;
6. given a set of programs, PL and OTDL, a set of strategies, an optimization criterion, required to get a set of optimized programs and their optimality estimates, study the dependence of program optimality estimate on its characteristics, applied strategy and programming language.

From the list given above one can see that all tasks are special cases of one task: given a set of PL, OTDL, input programs on these PL, and also a set of optimizing transformations, strategies of their application and functions – optimality estimates, required to build a set of optimized programs, protocols of their optimization and a set of optimality estimates for all programs on each step of optimization.

Method of solving the task of modelling optimization process

It is obvious that solving any task from the given above comes to solving the following task: with PL, OTDL fixed, the only input program, a set of OT, strategy of their application and function – optimality estimate are defined, required to build an optimized program, get the protocol of the optimization process and optimality estimate.

The algorithm to solve the task is given below.

```
BEGIN
Step of history=1;
Current OT (Step of history)= Chain of OT application;
Number of applied OT(Step of history)=1
Last step=False
REPEAT
    First OT application=True
    To analyze SMP(Step of history)
REPEAT
    IF Not First OT application
        THEN Number of applied OT(Step of history)= Number of applied OT(Step of history)+1
    Applied OT= Chain of OT application[Number of applied OT (Step of history)]
    Candidates to SB(Step of history)=To find saving blocks(Step of history, Applied OT)
    First application =False;
UNTIL (Candidates to SB(Step of history)≠∅) or
        (Number of applied OT (Step history)=Length(Chain of OT application))
IF  Candidates to SB(Step of history)≠∅
    THEN
        FOR SB in Candidates to SB(Step of history)
            DO SB characteristic(Step of history, SB) =
                To build SB characteristics (Indicative function(Step of history, SB))
                Characteristic of chosen SB(Step of history)=To realize Strategy (Strategy of OT(characteristic
                of SB(Step of history)))
                Chosen SB=To choose SB (Step of history, Characteristic of chosen SB(Step of history))
                Optimized SB(Step of history)=To build optimized SB(Step of history, Chosen SB, Applied
                OT)
```

To build new SMP(Step of history+1), where exists the only Optimized SB (Step of history) where Transformation Formula (Step of history, chosen SB(Step of history), Step of history+1, Optimized SB(Step of history)), and the rest context coincides.

Step of history= Step of history+1

ELSE Last step=True

UNTIL Last step=True

Number of Optimization Steps= Step of history

END.

This algorithm is simple and obvious enough to serve as a kernel of instrumental system for program optimization. However, the functions applied in it: To analyze SMP, To find saving blocks, to build characteristics of SB, To realize strategy, To build optimized SB and To build new SMP are not that obvious and can be considered as separate subtasks.

The structure of intelligent system on program optimization

The developed ontology, the tasks given in it and proposed methods of solving them provide the basis for the instrumental system for program optimization. Instrumental modeling expert system of program optimization (I_MESPO) is intended to support teaching of classical optimizing transformations.

This system allows the user to describe optimizing transformations, to set their application conditions and transformation rules, to form various sets of optimizing transformations, to assign chains, to trace the program optimization history.

The input data of the system are optimizing transformations knowledge, testing program on an algorithmic high-level language.

The result of the work of the system is the protocol of the optimization history protocol where for each optimization step it is shown what transformation has been applied, what saving blocks have been found on this step, which block has been chosen and what it has been replaced by.

Since this task is connected with the complicated processing of the knowledge given and generation of new one, the given subsystem was done as an expert system that models program optimization process.

The expert system includes a subsystem of visual input of knowledge about program optimization, knowledge base description language translator (Synthesizer), high-level language translator in the model of structured programs, estimating and result visualizing subsystem, integrated shell providing interface among these subsystems.

The work with the system I_MESPO begins with that the researcher defines the system of optimizing transformations, i.e.: assigns a list of OTs, a chain of their applications, and defines context conditions for each OT, transformation formulas, indicative functions and optimization strategies. After assigning the values of these parameters, Synthesizer executes translating knowledge base about program optimization into implemented module, thus creating application expert system (AES).

The functioning of the created application expert system begins with the work of a translator included in the system that transforms the structured program written in a high-level language into the structured program model (SPM). SPM is an internal form of relational presentation of programs that is convenient for analyzing and optimizing. According to this model, the application expert system makes an inference and builds up the optimization history protocol of this program. After receiving this protocol, the estimate and visualizing subsystem allows to estimate the program optimality before and after optimizing with the help of assigned estimating function and to compare the texts of the programs on each optimization step on high-level language and to analyze the implemented changes.

Conclusion and Acknowledgements:

Knowledge processing in the field of program optimization makes it possible to use this knowledge in industry, science and education.

Describing optimizing transformations within the terms of one model must facilitate the unification of different transformations within the framework of one system of OT. Thus, specialists can spend much less effort to study and use optimizing transformations to solve program optimization problems.

The use of the knowledge gives an opportunity to train highly qualified specialists to solve tasks on program optimization.

The access to knowledge through the Internet will attract all specialists interested in knowledge exchange on this problem.

References:

[Artemjeva, 2002] Artemjeva I.L., Knyazeva M.A., Kupnevich O.A. Domain ontology model for the domain "Sequential program optimization". Part 1. The terms for optimization object description. In The Scientific and Technical Information, 2002. (In Russian).

[Kasyanov, 1988] Kasyanov V. N. Optimizing transformations of the programs. Moskow: Nauka, 1988.

[Kleschev, 2001] Kleschev A.S., Orlov V.A. Requirements on a computer bank of knowledge. In Proceedings of the Pacific Asian Conference on Intelligent Systems 2001, Seoul, Korea, 2001.

[Kleshchev, 2002] Kleshchev A.S., Chernyakhovskaya M. Yu. The present state of computer knowledge processing. http://www.dvo.ru/iacp/es/publ/kpe.htm

[Knyazeva, 1999] Knyazeva M.A., Kupnevich O.A. Expert system for simulation of program optimization. Joint NCC&IIS Bull., Comp. Science,12 (1999), 24-28.

Author information:

Irene L. Artemjeva: artemeva@iacp.dvo.ru
Margarita A. Knyazeva: mak@nt.pin.dvgu.ru
Oleg A. Kupnevich: garfield1@yandex.ru
Institute for Automation & Control Processes, Far Eastern Branch of the Russian Academy of Sciences
5 Radio Street, Vladivostok, Russia

A KNOWLEDGE-ORIENTED TECHNOLOGY OF SYSTEM-OBJECTIVE ANALYSIS AND MODELLING OF BUSINESS-SYSTEMS

M. Bondarenko, V. Matorin, S. Matorin, N. Slipchenko, E. Solovyova

Abstract: A new original method and CASE-tool of system analysis and modelling are represented. They are for the first time consistent with the requirements of object-oriented technology of informational systems design. They essentially facilitate the construction of organisational systems models and increase the quality of the organisational designing and basic technological processes of object application developing.

Keywords: Knowledge, systemology, natural classification, objects modelling, conceptual knowledge.

The civilization sustainable development is based on formation of the informational society as a first stage of the noosphere. At the same time, as the transition to the informational society, as economic activity in it become based on knowledge. This knowledge represents the "informational resource". It directly influences at the material factors of progress and ensures the "phase transition of knowledge into a power", i.e. efficiency of business, production and any administrative solutions.

The submission about the tendency of knowledge-oriented development of an alive nature is entered into scientific practice by V.I. Vernadskiy under a title "*the Dan's principle*". The knowledge-oriented development should be considered as the universal tendency enveloping not only biological, but also all other complicated systems. The social (organizational) and information systems also develop *in a direction of increasing of a knowledge role for their sustainable functioning*.

This tendency is exhibited in the unprecedented growth of knowledge and scientific information; increasing of a role of inclusive, depth knowledge; rapid development of methods and means of knowledge processing, analytical activity, acute need of the appropriate experts, influence of informational resources to all sides of the human activity. The technologies and methods of purchase, extraction, submission, processing of knowledge (knowledge management, knowledge engineering) in substantial aspect also develop in the knowledge-oriented direction (data mining, text mining, knowledge discovery, knowledge mining, object