УДК 001.891.57:004.043(045)

Sidorova N.N.
National Aviation University

ONTOLOGY-DRIVED METHOD USING PROGRAMMING STYLES

Application of experience in software engineering plays an important role in improving the efficiency of development and maintenance of software products. Experience is applied through using of software development methods and life cycle models, based on the use of legacy software, and reuse. In connection with distribution of engineering methods of the software development, the models of life cycle, based on component development and reuse, and extreme programming are put and solved the problems connected with reading of program texts, written on different programming languages and at various times. Application of these methods and models increases the complexity of software and the collective nature of its development and maintenance and requires the use of programming styles. Through collaborative development and reuse, the style has a direct and through training - indirectly related to all processes of the software lifecycle. Application of styles in programming means the improvement of the efficiency of development and maintenance of software. For the first time is proposed a method of application programming style based on ontologies, providing greater efficiency through using of style complete, by the way precise and formal description of the domain ontology and using OWL-DL to automate processes associated with its creation and maintenance. The results of the study programming stylistics domain are presented. The result is the taxonomy. A taxonomy of concepts was used to construct the programming styles ontology. Ontology is a part of the tool that is constructed according to the own method. The tool is created for using in programming processes and called a programmer assistant tool.

Застосування досвіду грає важливу роль в інженерії програмного забезпечення для підвищення ефективності створення і супроводження програмного забезпечення. Досвід застосовується шляхом використання методів і моделей життєвого циклу, які засновані на застосуванні успадкованого програмного забезпечення і повторного використання, що спрямовані на рішення проблем пов'язаних з читанням текстів програм, написаних на різних мовах програмування і в різний час. Завдяки колективної розробці і повторному використанню, стиль має прямий, а завдяки навчанню непрямий вплив на усі процеси життєвого циклу. Застосування стілей веде до підвищення якості і супроводжуємости програмного забезпечення. Вперше пропонується метод застосування стилів програмування на основі онтологій, що підвищує ефективність і автоматизує відповідні процеси. Онтологія є частиною інструменту який автоматизує застосування стилю.

Применение опыта играет важную роль в инженерии программного обеспечения для повышения эффективности создания и сопровождения программного обеспечения. Опыт применяется путем использования методов и моделей жизненного цикла, которые основаны на применении унаследованного программного обеспечения и повторного использования, направленных на решение проблем, связанных с чтением текстов программ, написанных на различных языках программирования и в разное время. Благодаря коллективной разработке и повторному использованию, стиль имеет прямой, а благодаря обучению косвенное влияние на все процессы жизненного цикла. Применение Стилей ведет к повышению качества и супроводжуемости программного обеспечения. Впервые предлагается метод применения стилей программирования на основе онтологий, повышает эффективность и автоматизирует соответствующие процессы. Онтология является частью инструмента который автоматизирует применение стиля.

Keywords: *Software, programming, programming language, programming style, coding standard, ontology, taxonomy, OWL-DL.*

Formulation of scientific problem

Application of experience in software engineering plays an important role in improving the efficiency of development and maintenance of software products. Experience is applied through using of software development methods and life cycle models, based on the use of legacy software, and reuse [1]. In connection with distribution of engineering methods of the

software development, the models of life cycle, based on component development and reuse, and extreme programming are put and solved the problems connected with reading of program texts, written on different programming languages and at various times. Application of these methods and models increases the complexity of software and the collective nature of its development and maintenance and requires the use of programming styles [2]. For example, on the Fig. 1 fulfilling program is presented [2]. This program was written on the special "confusion" way. Therefore, to understand texts of programs, it is often necessary to know either specified features of the time of their writing, or ideologies dominating over this period and ideas of authors. It leads to the necessity for a programmer to be able to represent idea or ideology and to transfer representation together with the program. In various areas, this aspect of activity of the person concerns style, and its research - is a subject of stylistics.

```
  #include <stdio.h>
main(t,_,a)
  char *a;
  {return t>1?t<3?main(-79,-13,a+main(-87,1-_,
  main(-86, 0, a+1 )+a)):1,t<_?main(t+1, _, a ):3,main ( -94, -27+t, a
  )&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?main(_,
  t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
  ,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
  +k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')){}#}w')){}{nl]'/+#n';d}rw' i;# ){n\
  l]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
  n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
  ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
  #'rdq#w! nr'/ ') }+}{rl#'{n' )# }'+}##(!!/")
  :t<-50?_==*a?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
  +1 ):0<t?main ( 2, 2 , "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \
  i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

Fig. 1 The text of "confusion" program

Therefore, the study and solution of problems related to the application programming styles for a long time is of particular relevance.

Programming style ensures all processes of creating software, represented by a set of rules expressed by the linguistic resources and reflects prevailing during the software life cycle is not only technical, but also a cultural experience [3 - 5].

Through collaborative development and reuse, the style has a direct and through training - indirectly related to all processes of the software lifecycle. Application of styles in programming means the improvement of the efficiency of development and maintenance of software.

Analysis research

At various times the problem of style programming directly or indirectly was studied by E.Dejkstra, I.Kernigan, F.Plodger, W.Tassel, I.Velbitsky, A.Ershov, I.Pottosin, N.Sidorov. In programming, the concept of style was introduced with the advent of structured programming. I.Kernigan, F.Plodger were the first researches who began to use the style. Later, there were different interpretations of style, for example, by A.Ershov, V.Borovin. N.Sidorov was proposed the programming stylistics subject [2].

There are two approaches to solving problems of application programming style: language-oriented and technology-oriented [2]. The essence of the first approach is based on the assumption that the use of programming style is done by writing the texts of programs by means of a programming language, and hence texts of programs never go beyond language. This approach has the following disadvantages:

- a translator for a changing variety of styles cannot be build;
- some of the rules that describe the style, can not be converted into grammar;
- some of the rules that describe the style, can not be realized only lexical and syntactic computation.

The essence of the second approach is to develop and implement means, processes and methodologies for automation solutions of the style application. In this case, the means to meet the following requirements:

- ensure that the traditional notion of styles;
- implement the necessary actions associated with the use of empirical methods;
- does not depend on the phases of the life cycle.

The database is the basis of the means of applying the programming style. However, the use of databases to represent the domain knowledge shown its limitations and appropriate use of new tools, such as ontologies [6].

Main material and justification of the results

For the first time is proposed a method of application programming style based on ontologies, providing greater efficiency through using of style complete, by the way precise and formal description of the domain ontology and using OWL-DL to automate processes associated with its creation and maintenance.

Fig. 2 shows the organization of the method. Processes are supported by three ontologies. One reflects the knowledge of programming styles, and the second, about the styles of programming languages, and the third, about programming languages.

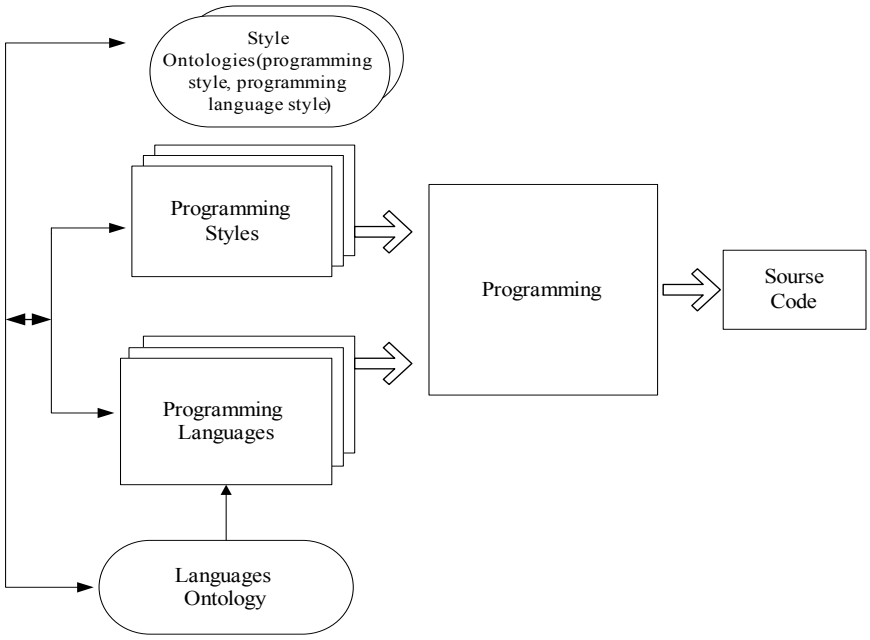

Fig. 2. Ontology-driven using of programming style

The investigation of method requires some work, which consists in the analysis of the subject area and constructing the source structures (thesaurus, taxonomy, dictionaries) (Figure 3).

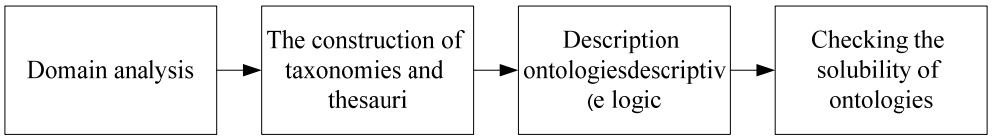

Fig. 3. The processes of preparation

The article discusses the results of processes.

Domain analysis is fulfilled with the help of the domain analysis techniques. As a result, a number of charts are created as the domain knowledge. Domain is called a programming stylistics. Knowledge about the domain is represented by three ontologies - programming style, programming language style, programming language (Fig.4).

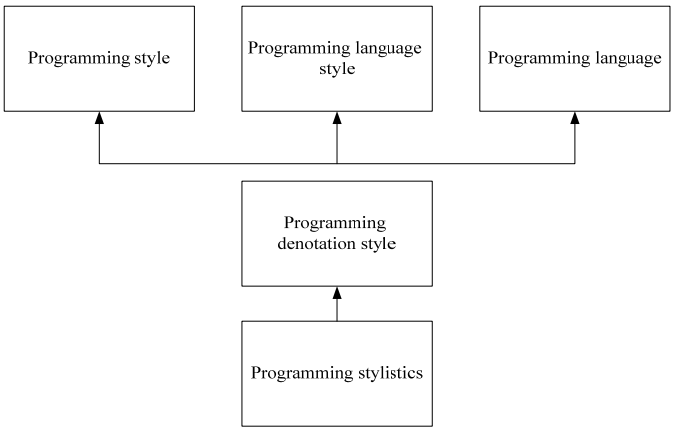

Fig.4. Programming stylistics

Knowledge is based on the definition of programming style. Programming style by the definition is the style that is used in human activity (domain), whose essence consists in programming (Fig. 5).

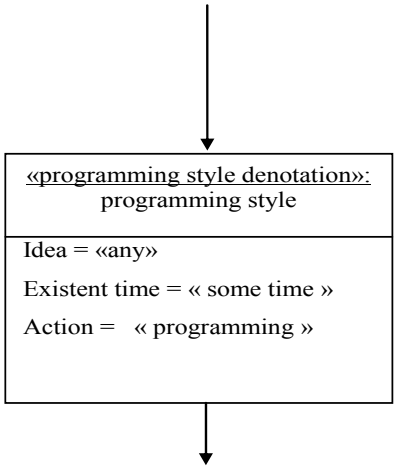

Fig.5. Class-programming style

Representation of style within the limits of the second approach can look like St = <A, S, D>, where A – set of own axioms of style not depending on an essence of expressed ideology, S and D - set of the axioms describing characteristic features of ideology of style in static's and dynamics. The last sets can be used for description of style of programming.

Set A may contain such axioms:

1. Uniqueness of style: if exists ideology (I) and style, based on it, then there are no styles that are also based on this ideology

$$\forall ISt(I) \sim (\overline{\exists} St(I) = St(I));$$

2. Existence of style of human activity: if there exists ideology (I), style St, based on it, and human activity P, then exists style of human activity St_p(St (I), P), based on the style St(I)

$$\forall ISt(I) \sim \forall PSt_p(St(I), P);$$

3. Reflexivity: every style is the sub style of itself

$$\forall St(St = St)$$

4. Antisymmetry: substyle (style, which is derived from some style) cannot be a style for a style, it was based on.

$$\forall St_1 \forall St_2 (R(St_1, St_2) \sim \neg R(St_2, St_1));$$

5. Transitivity: if style (St_2) is substyle of some style (St_1) and style (St_3) is substyle of (St_2), then (St_3) is substyle of (St_1)

$$\forall St_1, \forall St_2, \forall St_3 (R(St_1, St_2) \sim (R(St_2, St_3) \sim R(St_1, St_3))$$

The «programming» domain consists of three essences – subject (programmer), tool (programming language), and product (program) (Fig.5).

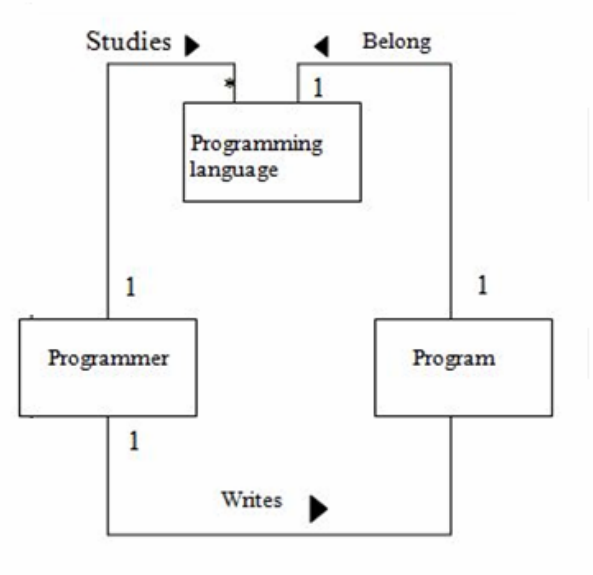

Fig.6. Programming domain

Programming style consists of the rules that apply to parts of the program text (Fig. 7)

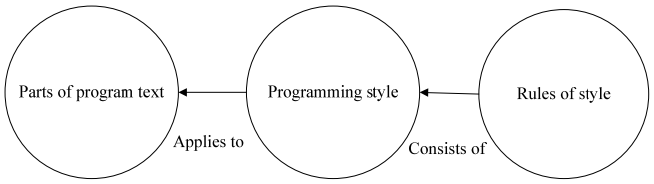

Fig. 7. Programming style with associated terms

The set of style rules consists of three types of rules - syntactic, semantic and pragmatic (Fig. 8)
.

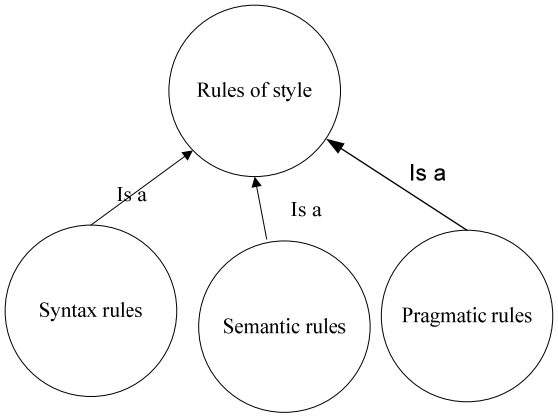

Fig.8. Rules of style

Description of the programming style is represented by the set of style rules. For example [6],

Syntax rule:
Synopsis: Do not use an underscore in identifiers

Language: C#
Level: 8
Category: Naming

Semantic rule:
Synopsis: Do not change a loop variable inside a for loop block
Language: C#
Level: 2
Category: Control flow
Description: updating the variable loop within the loop body is generally considered to be confusing, even more so if the variable loop is modified in more than one location. This rule also applies to foreach loops.

Pragmatic rule:
Synopsis: Name an identifier according to its meaning and not its type
Language: C#
Level: 6
Category: Naming
The rule of style consists of four parts: synopsis – is described essence of rule; language – the language is identified; level – level of using; category – link with language constructions and processes; description – consists of comments.

Programming style is applied to parts of the program text. Part of the program text can be of two types – predefined by syntax parts, and parts, that can be defined (Fig. 9).

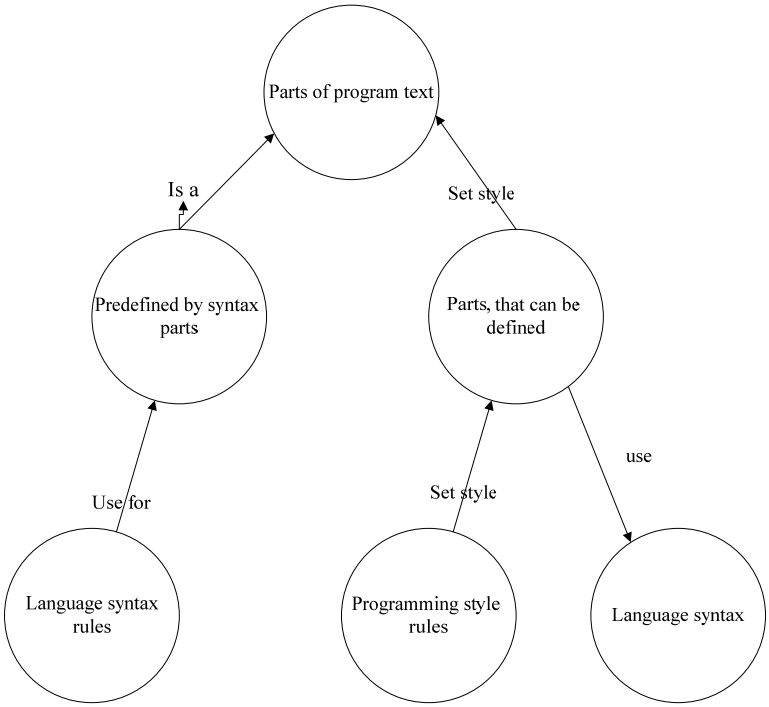

Fig.9 Parts of program text

Programming languages are represented with the help of the encapsulation levels [1]. Each level has its own type of the programming construction (Figure 10). There are lexems (lexical level), operators (operator level), subroutines (subroutines level), modules (module level), classes (class level).

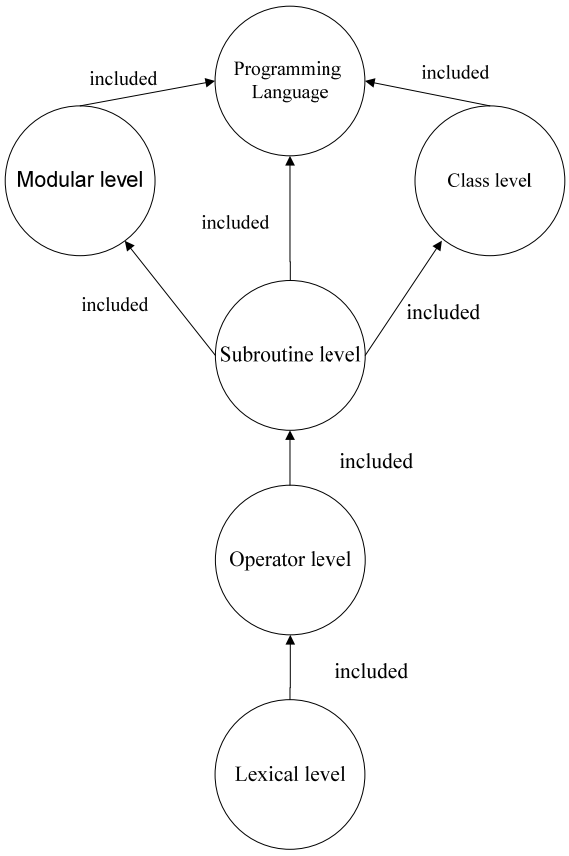

Fig. 10 Levels of encapsulation of programming language

Definition in descriptive logic this chart is the following:

LexicalLevel ⊑ OperatorLevel

OperatorLevel ⊑ SubroutineLevel

SubroutineLevel ⊑ ModularLevel

SubroutineLevel ⊑ ProgrammingLanguage

SubroutineLevel ⊑ ClassLevel

ModularLevel ⊑ ProgrammingLanguage

ClassLevel ⊑ ProgrammingLanguage

Thus, using levels of encapsulation the style rules can be classified as the following (Figure 11).

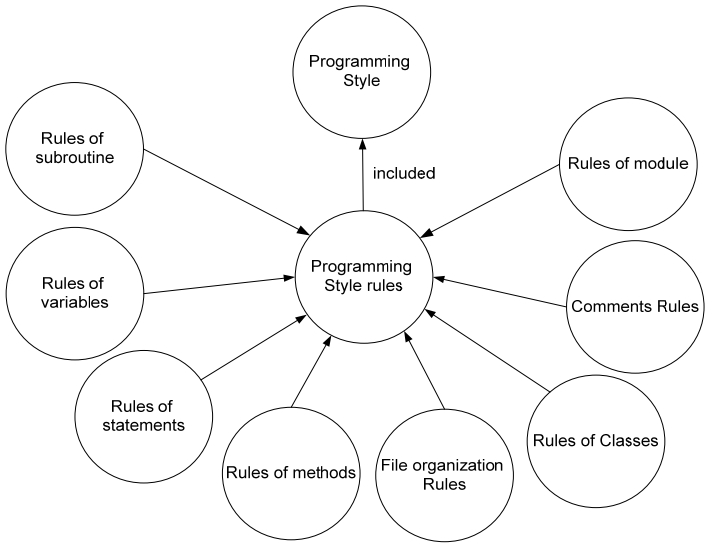

Fig. 11 Ontology of programming style's rules

Definition in descriptive logic this chart is the following:

ProgrammingStyleRules ⊑ ProgrammingStyle

RulesOfSubroutine ⊑ ProgrammingStyleRules

RulesOfVariables ⊑ ProgrammingStyleRules

RulesOfStatements ⊑ ProgrammingStyleRules

RulesOfMethods ⊑ ProgrammingStyleRules

FileOrganizationRules ⊑ ProgrammingStyleRules

RulesOfClasses ⊑ ProgrammingStyleRules

CommentsRules ⊑ ProgrammingStyleRules

RulesOfModule ⊑ ProgrammingStyleRules

The concrete programming styles are a created for concrete programming languages (Figure 11).

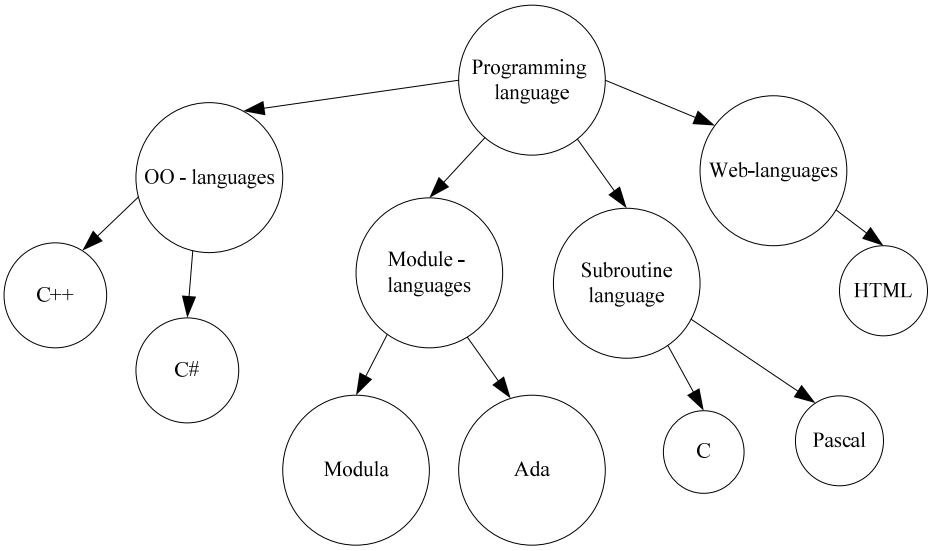

Fig. 11 Programming language styles

Definition in descriptive logic this chart is the following:

OOLanguages ⊑ ProgramingLanguage

ModuleLanguages ⊑ ProgramingLanguage

SubroutineLanguages ⊑ ProgramingLanguage

WebLanguages ⊑ ProgramingLanguage

C++ ⊑ OOLanguages

C# ⊑ OOLanguages

Modula ⊑ ModuleLanguages

Ada ⊑ ModuleLanguages

C ⊑ SubroutineLanguages

Pascal ⊑ SubroutineLanguages

HTML ⊑ WebLanguages

Fragment taxonomy rules for object-oriented language (C#) is shown in the Figure 12. The taxonomy consists of two blocks – naming convention and statements rules. There are also declaration rules, comment rules, white space rules and file organization rules.

For example, naming convention rule is “Do not use an underscore in identifiers” or “Use an -ing and -ed form to express pre-events and post-events”. The ontology is created on the base of this taxonomy.

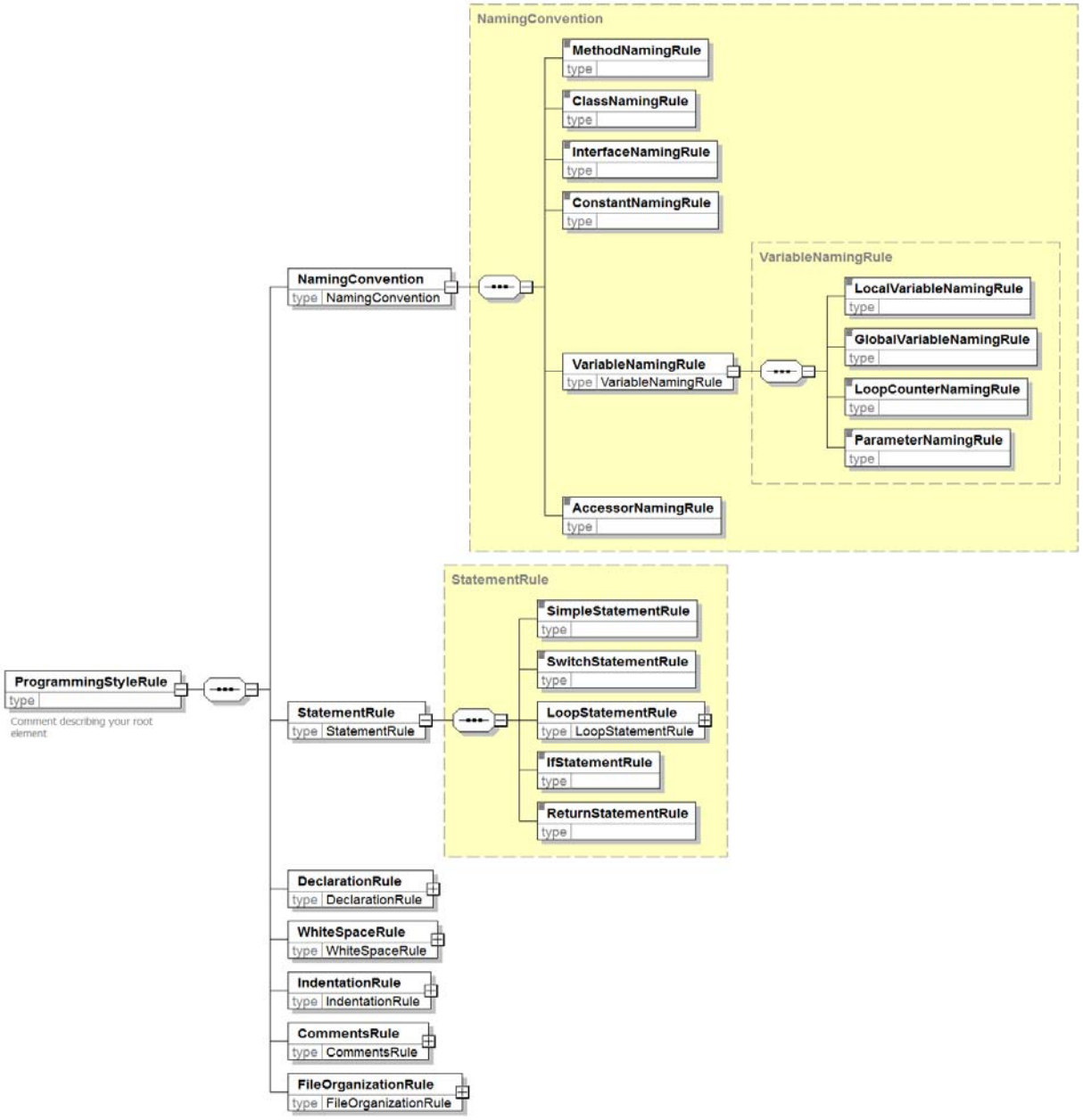

Fig. 12 Taxonomy of programming rules

The method was used for created the tool – "Programmer Assistant" (Fig. 13). The main part of tool is ontology. The tool realized with the help of Protégé platform. The ontology was included using Protégé editor.

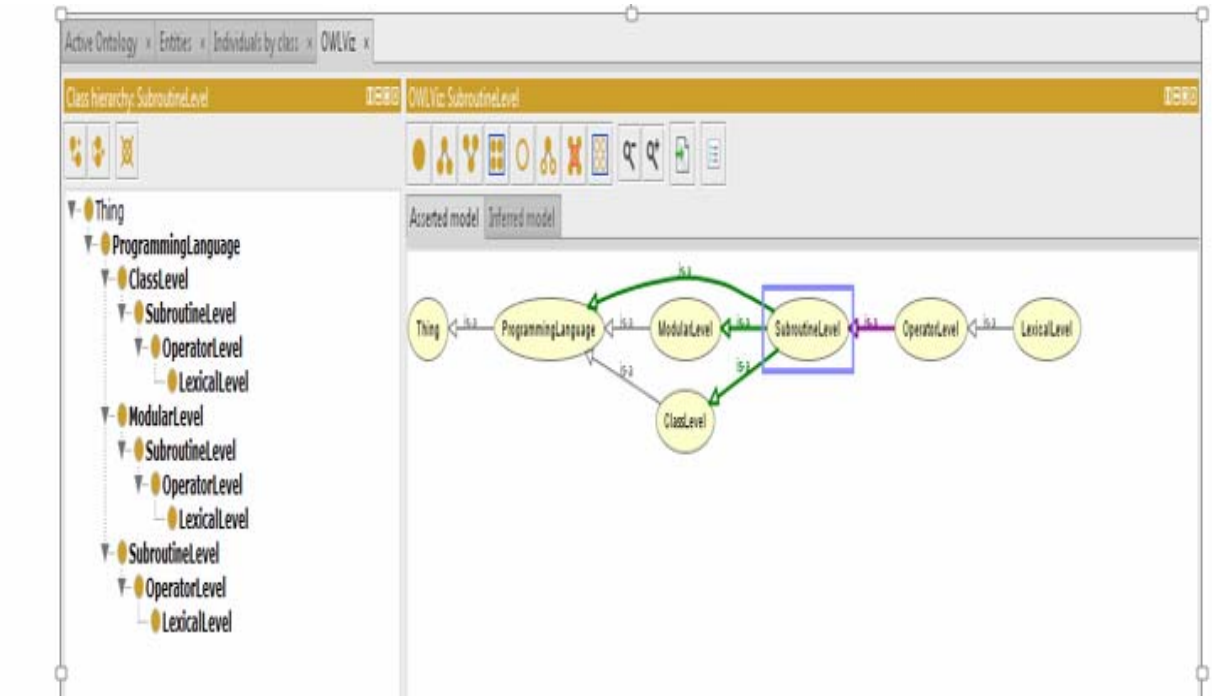

Fig.13 Screenshot of interface

Results and future researches

For the first time is proposed a method of application programming style based on ontologies, providing greater efficiency through using of style complete, by the way precise and formal description of the domain ontology and using OWL-DL to automate processes associated with its creation and maintenance.The results of the study programming stylistics domain are presented. The result is the taxonomy. A taxonomy of concepts will be used to construct the programming styles ontology. Ontology is a part of the tools that is constructed according to the own method. The future researches are creating programming style ontology and programmer assistant tool.

References

1. Sidorov M.O. Software engineering. [текст] / Sidorov M.O. // – 2007. – Kyiv. – NAU.- 135 p.
2. Sidorov N.A. Software stylistics [текст] / Sidirov N.A. // Proc. of the National Aviation University – 2005. - №2. – P. 98 – 103.
3. Goldberg A. Programmer as Reader [текст] / Goldberg A. // IEEE Software – 1987. Sept. – P. 62 – 70.
4. V. Railich Software cultures and evolution / V. Railich, N. Wilde, M. Buckellew // Computer. – 2001. – Sept. – P. 25 – 28.
5. Knuth D.E. Literate are programming [текст] / Knuth D.E. // Computer Journal. – 1984. – Vol. 27, N 2. – P. 42 – 44.
6. Sidorova N.M. Ontology of programming style [текст] / Sidorova N.M., Kramar Y.M. // – Proc. the sixth world longest "Aviation in the XXI-st Century. – 2014. – v.1. – P.1.13.28 – 1.13.36
7. Philips Healthcare – C# Coding Standart [текст]. – Philips. – 2009. – 57p.

Information about author:

Sidorova Nika Nikolaevna – postgraduate student of Software Engineering Department of the National Aviation University. Scientific interests: software engineering, education.

E-mail: nika.sidorova@livenau.net