DOI: 10.18372/2306-1472.71.11751

Nikolay Sidorov[1]
Nika Sidorova[2]
Alexander Pirog[3]

ONTOLOGY-DRIVEN TOOL FOR UTILIZING PROGRAMMING STYLES

National Aviation University
Kosmonavta Komarova avenue 1, 03680, Kyiv, Ukraine
E-mails: [1] nikolay.sidorov@livenau.net; [2] nika.sidorova@gmail.com; [3]pirogman@gmail.com

Abstract

Activities of a programmer will be more effective and the software will be more understandable when within the process of software development, programming styles (standards) are used, providing clarity of software texts. ***Purpose:*** *In this research, we present the tool for the realization of new ontology-based methodology automated reasoning techniques for utilizing programming styles. In particular, we focus on representing programming styles in the form of formal ontologies, and study how description logic reasoner can assist programmers in utilizing programming standards.* ***Our research hypothesis*** *is as follows: ontological representation of programming styles can provide additional benefits over existing approaches in utilizing programmer of programming standards.* ***Our research goal*** *is to develop a tool to support the ontology-based utilizing programming styles.* ***Methods:*** *ontological representation of programming styles; object-oriented programming; ontology-driven utilizing of programming styles.* ***Results:*** *the architecture was obtained and the tool was developed in the Java language, which provide tool support of ontology-driven programming styles application method. On the example of naming of the Java programming language standard, features of implementation and application of the tool are provided.* ***Discussion:*** *application of programming styles in coding of program; lack of automated tools for the processes of programming standards application; tool based on new method of ontology-driven application of programming styles; an example of the implementation of tool architecture for naming rules of the Java language standard.*

Keywords: coding standard; description logic; ontology; programming; programming style; reasoner; software engineering; the Java language.

1. Introduction

Currently, methods and means of software development and maintenance, which are based on multiple use and reuse, are spreading [1]. Application of these methods and means requires a programmer to read and understand a significant amount of source texts, and major requirement to the software is its understandability. Activities of a programmer will be more effective and the software will be more understandable when within the software development process the programming styles (standards) will be used, providing clarity of software texts. Programming stylistics problems arose in the period before the structured programming, but nowadays they remain relevant [2]. Within the previous years, many programming standards have been developed [3-7]. But today the issue of solving the problem of their effective use receives little attention. The existing problems of using the standards such as [8]: opposition of development team to use standards; developers "forgetting" to use standards; management thinking that the implementation of standards is too expensive, are resolved by developing and using tools that automate the corresponding processes. In the paper [9], new method of programming styles application based on the ontology has been proposed. In this paper, the task of implementing the ontology-driven tool for utilizing programming styles based on the proposed method is considered.

2. Analysis of latest research and publications

Programming (coding) style (standard) is a set of rules and requirements regulating the conditions for the development of software text. Typically, standards are composed for a particular programming language [3-5], but usually cover four aspects [10]:

1. Formatting - requirements for the general form of code text lines;
2. Naming - requirements for the names in programming text (literals, variables, methods, modules, classes);
3. Commenting - requirements for the commenting in programming text;
4. Coding - requirements for the use of the language statement basis, definitions and initialization.

Generally, coding standard is represented in the form of text document, which sets out the requirements and rules for the writing of program text. The set of style rules consists of three types of rules – syntactic, semantic and pragmatic. For example, a syntactic rule [3]:

Synopsis: Do not use an underscore in identifiers
Language: C#
Level: 8
Category: Naming

Team of programmers should go along the following steps of path in order to implement a coding standard [8]:

- reach a consensus as to which coding standard to implement;
- study the coding standard and modify the standard (if required);
- follow the coding standard during the coding process;
- execute monitoring and fixed bugs made within the implementation of the standard.

Automation of the processes of studying, following and monitoring (fixing bugs) makes the implementation of programming standards beneficial, cheap and reliable. That is why there is a number of tools aiming at these processes [10,11]. Two key processes that should implement such tools are to be denoted. The first process is the presentation of programming standard requirements and rules to a programmer (the first two steps of the path), and the second one is checking whether the programmer has been following these requirements and rules during the coding (the last two steps of the path).

Nowadays, the first process is not automated, paper documentation manuals providing the following information are being used:

- description of a rule;
- motivation for the existence of a rule;
- example of where and how a rule should be used;
- example of where a rule should not be used;
- example of where and how a rule can be verified.

In the paper [12], the results of the implementation of the first process with the help of automating bringing information to a programmer using ontologies are given.

To implement the second process, static code analysis tools are being developed [13]. These tools are usually work by scanning the code with the help of the configuration file, which stores the information about the programming standard, usually in HTML or XML, which makes the setup of tools expensive and inconvenient. The task of implementing this tool basing on the ontology is considered in this paper.

3. Purpose and objectives of the research

In this research, tool for the realization of new software methodology that utilizes ontologies and automated reasoning techniques for utilizing programming standards is presented. In particular, the focus is made on the representation of programming standards in the form of formal ontologies, and the study of how description logic reasoner can assist programmers in utilizing programming standards. The research hypothesis is as follows: ontological representation of programming standards can provide additional benefits over existing approaches in utilizing programmer of programming standards. The research goal is to develop a tool to support the ontology-driven utilizing programming standards.

4. Tool of ontology-driven application of programming styles

To apply the programming style, a programmer should decide two tasks: study the description of the style; use and check the style during the coding. Thus, it requires two tools - one for studying the style and the other one to check the use of this style. Both tools are based on the representation of the style. That is why the form of this representation affects the efficiency of processes performed by a programmer and the efficiency of tool. It is proposed to use the ontology as a form of knowledge representation about programming style [12] (Fig. 1).

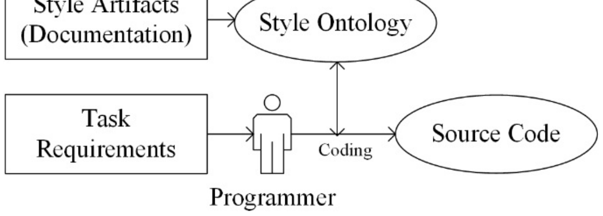


Fig. 1. Ontology of style in programming

Using appropriate tool (e.g. Protégé [14]), a formal representation of programming style - an ontology is developed. A programmer for coding uses ontology. Therefore, two tools are required - one for creating an ontology and assisting the programmer, the second one to control the implementation of style during the coding (Fig. 2).

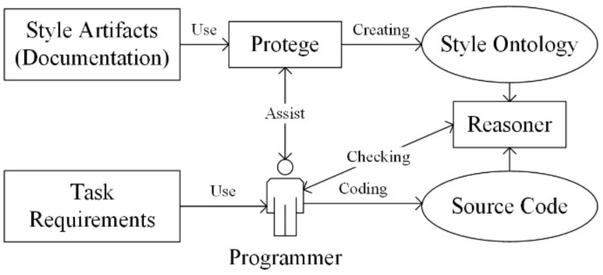


Fig. 2. Ontology-used tools

The first tool creates the ontology template, which is defining, general programming standards properties. Style analyst using Protégé setup template on programming standard. After that the programmer uses ontology to study programming standard.

The second tool is a reasoner [15]. In terms of descriptive logic, the reasoner solves one major problem – verifies consistency of the ontology [16]. This problem has certain features for the task of programming style implementation (Fig. 3).

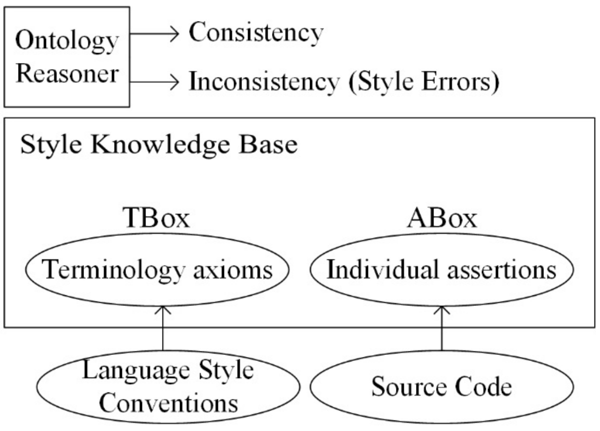


Fig. 3. Model of style knowledge Base

Protege is used to create TBox, which includes terms describing programming style. The assertions about the source code (ABox) are created according to the source code that is written by a programmer. Reasoner provides appropriate service based on TBox and ABox. However, it should be not only assertion about knowledge base consistence, i.e. compliance of ABox assertions regarding TBox, but also indications of specific stylistic errors in the source code in case of the knowledge base inconsistency. Thus, “regular” reasoner will not fully satisfy this service. Therefore, the implementation of corresponding Style Ontology Reasoner (SOReasoner) is described below.

As far as the ABox assertions for the operation of SOReasoner are not generated to TBox, the style ontology should be implemented in a format for SOReasoner. This format is recorded in a pattern on OWL, which is used for creating style ontology. Means for the creation of OWL template (OWLParser) and ABox (SourceCodeParser) are united in SOReasoner (Fig. 4).

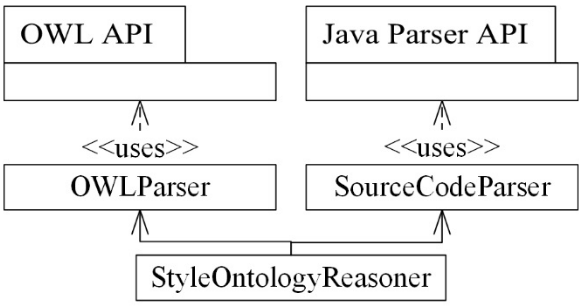


Fig. 4. Style Ontology Reasoner scheme

SOReasoner consists of two main parts: OWL Parser and Source Code Parser. First part will be achieved by means of the OWL API and will generate OWL template when the ontology is created and it will generate a set of TBox rules when programming style of source code is checking. Second part will use Java Parser API to analyze Java source code, checking consistency style knowledge base and give feedback. It is also important to design user-friendly interface. The OWL API is a Java API and reference implementation for creating, manipulating and serializing OWL ontologies in the Java language [17]. Java Parser API is a set of tools to parse; analyze; transform and generate of Java source code [18]. The activity diagram of SOReasoner is given in the Fig. 5.

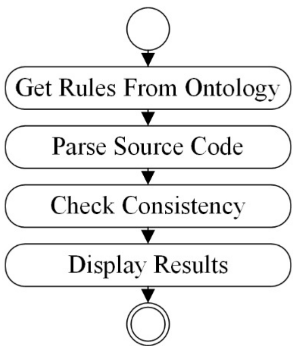


Fig. 5. SOReasoner activity diagram

Before source code file is parsed in order to check the style, it is required to obtain rules from the ontology. OWLParser provides functionality to parse ontology in order to obtain rules for checking code style (Fig.6). After OWLParser successfully loads the ontology, it starts to go through all axioms of TBox in order to build set of rules. If this process goes well, OWLParser returns a list of rules.

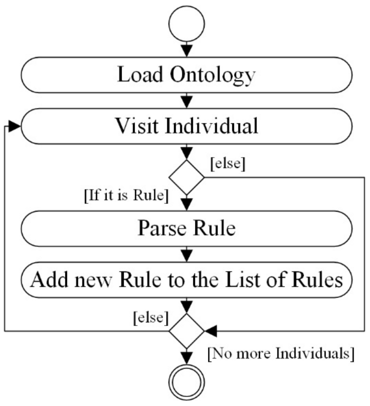


Fig. 6. OWLParser activity diagram

OWLParser can also be used to generate template style ontology with all required structures for specifying general rules of programming style conventions. The process of adding new rules to the ontology (setting) is very simple because all required structures have been added to the template. In Fig.7, a part of hierarchy of concepts of the style ontology template for naming is given.

In Fig. 8, the subclass of the corresponding class and the description of this subclass are given. It includes the first symbol of the denotation, separator, denotation type, and symbols which may combine the denotation (except the first one). Specific samples for the denotation will be indicated when a template is setting in Protege. It is required to create new individual for each rule and add object property assertions to specify exact aspects of rule. After setting the template it is required to check the ontology in reasoner in order to double check newly added rules.

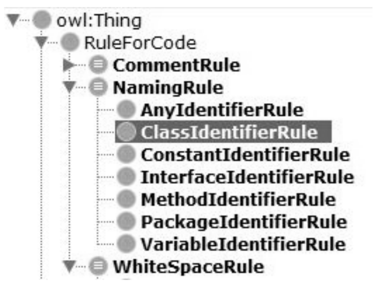
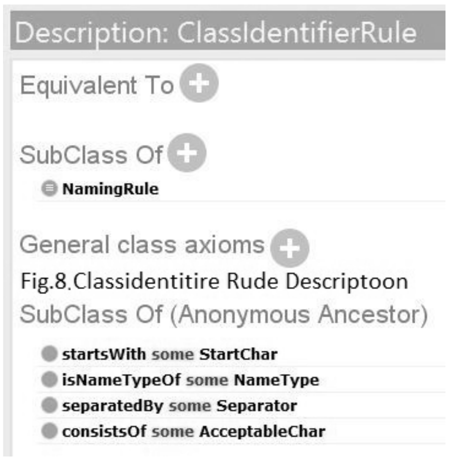


Fig. 7. Part of Concept Hierarchy in style Ontology Template


Description: ClassIdentifierRule

Equivalent To

SubClass Of
- NamingRule

General class axioms

Fig.8.Classidentitire Rude Descriptoon

SubClass Of (Anonymous Ancestor)
- startsWith some StartChar
- isNameTypeOf some NameType
- separatedBy some Separator
- consistsOf some AcceptableChar

Fig. 8. Class identifier Rule description

Source Code Parser provides functionality to parse source code using the list of rules (TBox) from ontology and obtaining ABox assertions about instances from source code. For parsing source code, it is required to set a path to desired Java file and then obtain rules (if any) for checking. After the preparation is finished, it is possible to start parsing. General process is as follows (Fig. 9).

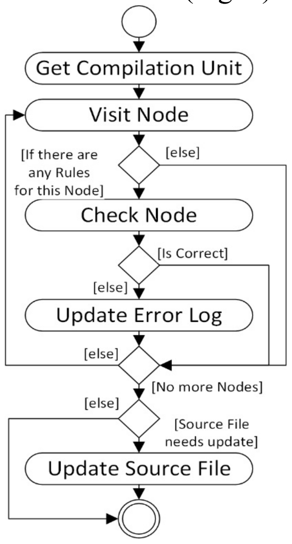


Fig. 9. Source code Parser Activity diagram

Firstly, SourceCodeParser parses compilation unit and gets information about all language constructions from nodes. Compilation unit is the abstract syntactic tree. While doing so, all nodes of compilation unit are also checked (if rules were specified). Then, if the update flag is set as true, SourceCodeParser will update source file via adding new comments after all problematic construction. The sequence diagram of ontology-driven using programming style is given in Fig. 10.

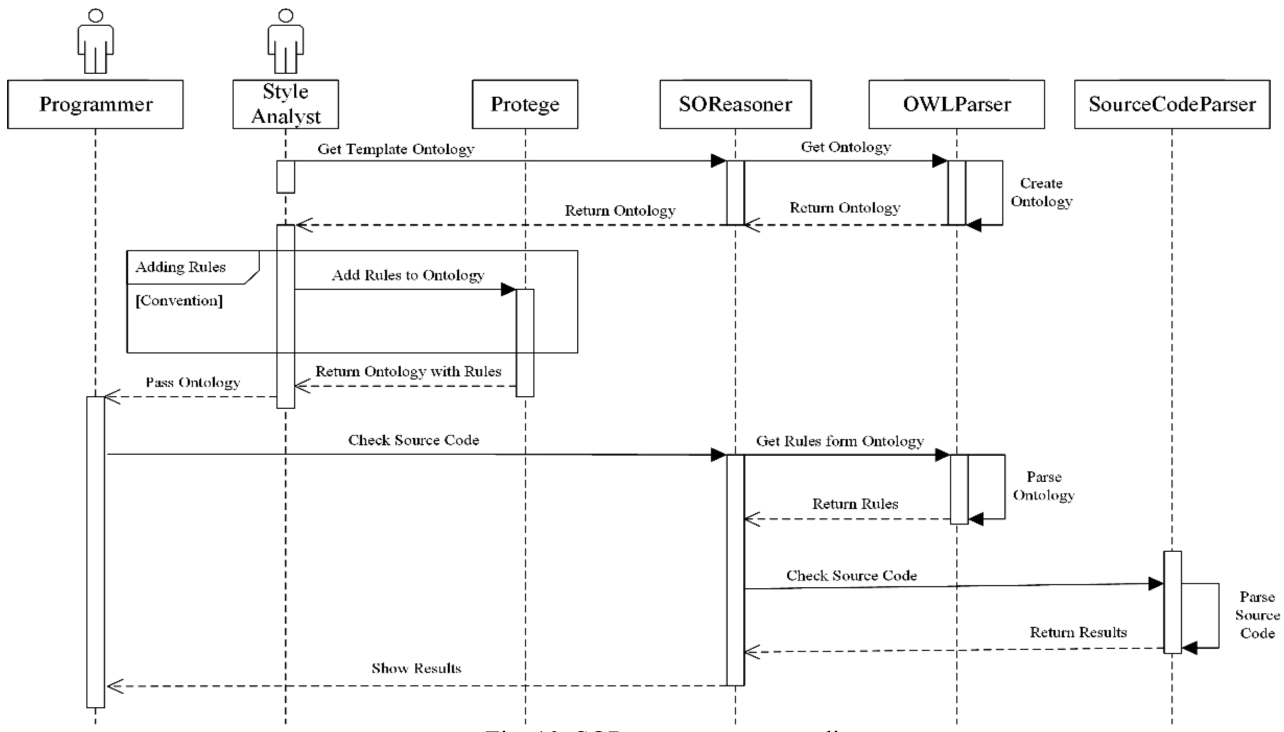


Fig. 10. SOReasoner sequence diagram

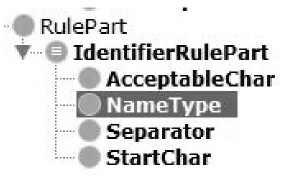
In this sequence diagram it is shown how objects operate with one another and in what order. Diagram shows the scenario of using SOReasoner through all its functionality.

Firstly, style analyst runs SOReasoner to get template ontology. For this purpose, OWL Parser creates a template and returns it back to style analyst. Secondly, style analyst runs Protégé and sets up the template (adds specific rules from convention language standard to the template). After that, style ontology is checked for the consistency. If the ontology is consistent, style analyst passes it to the programmer, who runs SOReasoner again to check source code file. This process can be separated into two main sub-processes: parsing ontology and parsing source code. First one is performed by OWLParser which takes ontology and returns a list of rules (TBox). Second one is performed by SourceCodParser, which takes source file (ABox), checks it against TBox rules and returns results. Finally, SOReasoner shows feedback to the programmer.

5. Case study

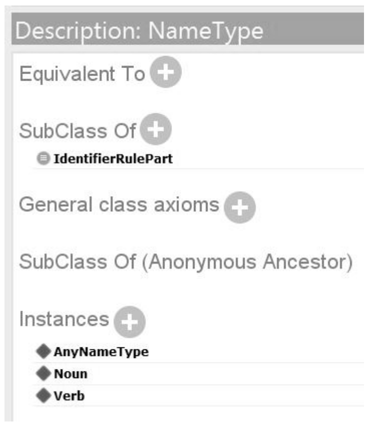
It is proposed to consider the application of SOReasoner on the example of naming rules from Java Convention [5].

With the use of OWLParser a template ontology for programming style is created. It builds all required classes, individuals, objects and data properties (Fig. 11, 12). After that, a list of TBox rules for the code is created.



RulePart
IdentifierRulePart
AcceptableChar
NameType
Separator
StartChar

Fig. 11. Fragment of ontology Template



Description: NameType
Equivalent To
SubClass Of
IdentifierRulePart
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
AnyNameType
Noun
Verb

Fig. 12. NameType

For example, in descriptive logic:

NamedObject – concept of all object names. Then:

$$\text{Class} \sqsubseteq \text{NamedObject}$$

$$\text{Method} \sqsubseteq \text{NamedObject}$$

$$\text{Parameter} \sqsubseteq \text{NamedObject}$$

$$\text{NamedObject} \sqsubseteq {=}1\,\textit{hasName}.\text{Identifier}$$

$$\text{Identifier} \sqsubseteq \textit{hasCharSet}.\{\text{US-English}\}$$

$$\text{Class} \sqsubseteq 1\,\textit{hasName}.\text{ClassIdentifier}$$

$$\text{ClassIdentifier} \sqsubseteq \text{Identifier}$$

$$\text{ClassIdentifier} \sqsubseteq \textit{ConsistsOf}.\text{Noun}$$

$$\text{Identifier} \sqsubseteq \text{hasFirstChar}.\text{SmallChars}$$

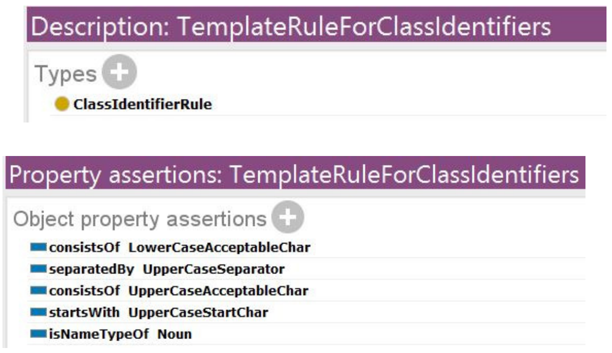
$$\text{Identifier} \sqsubseteq \text{hasOtherChar}.\text{AllChars}$$

$$\text{AllChars} \sqsubseteq \text{BigChars} \sqcup \text{OtherChars}$$

For example, there is a naming rule from Java convention [5] (Fig. 13). Style analyst can open this ontology template in Protege and use the rule description from convention to setup template (Fig. 14). In this case, OWLParser will definitely parse the ontology correctly.

Example of rule from Java Convention: *Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words—avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).*

Fig. 13. Rule description


Description: TemplateRuleForClassIdentifiers

Types
● ClassIdentifierRule

Property assertions: TemplateRuleForClassIdentifiers

Object property assertions
consistsOf LowerCaseAcceptableChar
separatedBy UpperCaseSeparator
consistsOf UpperCaseAcceptableChar
startsWith UpperCaseStartChar
isNameTypeOf Noun

Fig.14. Setup ontology template

Parsing source file for names (identifiers) is one of the functions of SourceCodeParser. In Fig. 15, a part of SOReasoner class diagram for names is given.

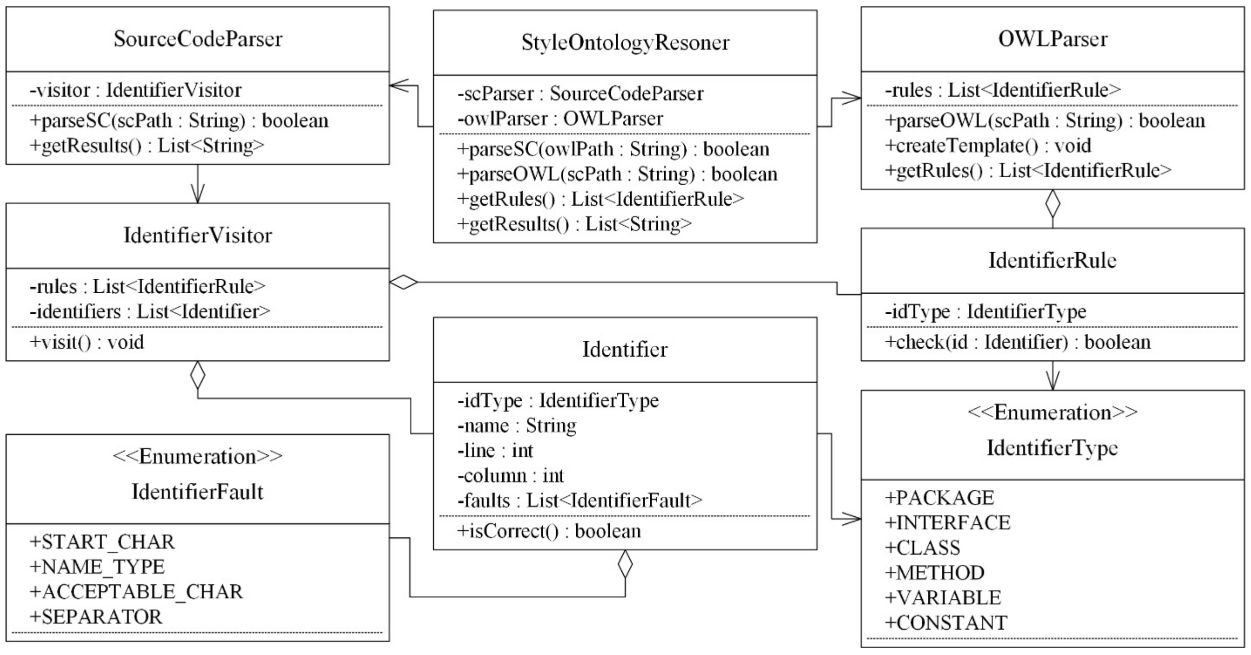

SourceCodeParser
-visitor : IdentifierVisitor
+parseSC(scPath : String) : boolean
+getResults() : List<String>
StyleOntologyResoner
-scParser : SourceCodeParser
-owlParser : OWLParser
+parseSC(owlPath : String) : boolean
+parseOWL(scPath : String) : boolean
+getRules() : List<IdentifierRule>
+getResults() : List<String>
OWLParser
-rules : List<IdentifierRule>
+parseOWL(scPath : String) : boolean
+createTemplate() : void
+getRules() : List<IdentifierRule>
IdentifierVisitor
-rules : List<IdentifierRule>
-identifiers : List<Identifier>
+visit() : void
IdentifierRule
-idType : IdentifierType
+check(id : Identifier) : boolean
Identifier
-idType : IdentifierType
-name : String
-line : int
-column : int
-faults : List<IdentifierFault>
+isCorrect() : boolean
<<Enumeration>>
IdentifierFault
+START_CHAR
+NAME_TYPE
+ACCEPTABLE_CHAR
+SEPARATOR
<<Enumeration>>
IdentifierType
+PACKAGE
+INTERFACE
+CLASS
+METHOD
+VARIABLE
+CONSTANT

Fig. 15. Fragment of SOReasoner class diagram

In this diagram, all custom classes that are involved in our case study example are shown. The main class is SOReasoner, which uses OWLParser and SourceCodeParser to perform its functions. OWLParser loads ontology and parses it to obtain rules, in our case rules from naming section of Java convention. SourceCodeParser uses IdentifierVisitor to go through all identifiers from Java source code. While doing so, IdentifierVisitor also checks each identifier for a set of rules. After that, SOReasoner returns results to the user. In code, all identifiers are presented as instances of Identifier class. It stores value of name (identifier), its type (class, variable, etc.), location in source code (line and column) and list of faults (if any). It also has several constructors and other methods. Naming rules is presented as instances of IdentifierRule class. It provides all required functionality to check identifiers. IdentifierRule is capable of checking start and acceptable characters, determine whether identifier is a noun or a verb and can separate compound identifiers into simple words. For example: MyFirstClass = My + First + Class. It can also be used to directly update source file with comments that identify problematic areas.

It is proposed to consider updating a file in detail. Firstly, SourceCodeParser reads source file line-by-line (Fig. 16). Secondly, SourceCodeParser adds comments to all lines with identifiers that do not follow rules (Fig. 17). Finally, SourceCodeParser writes all lines back to the file.

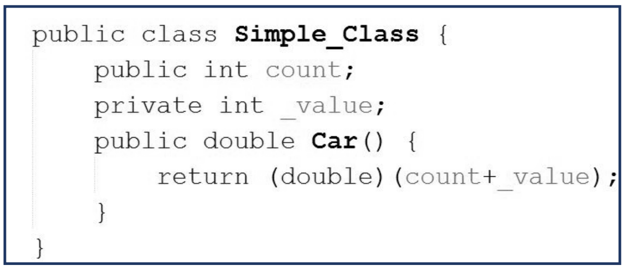

```
public class Simple_Class {
    public int count;
    private int _value;
    public double Car() {
        return (double)(count+_value);
    }
}
```

Fig. 16. Source code

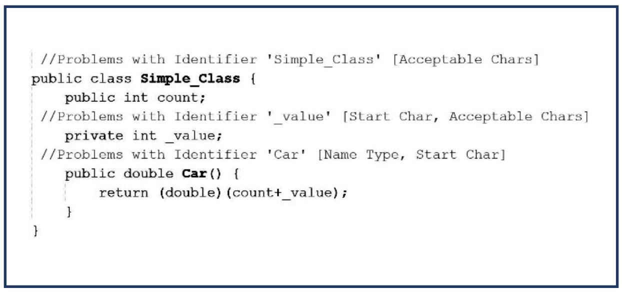

```
//Problems with Identifier 'Simple_Class' [Acceptable Chars]
public class Simple_Class {
    public int count;
//Problems with Identifier '_value' [Start Char, Acceptable Chars]
    private int _value;
//Problems with Identifier 'Car' [Name Type, Start Char]
    public double Car() {
        return (double)(count+_value);
    }
}
```

Fig. 17. Source code with comments

6. Conclusion

In this research, the implementation of tool for ontology-driven utilizing of programming styles is considered. Solutions for the development of general architecture of tools and main components are provided. Implementation details are given based on the example of naming styles for the Java convention.

References

[1] Sidorov M.O., (2007) *Software engineering,* Kyiv, NAU, 135p. (In English)

[2] Sidorov N.A., (2005) *Software stylistics*. Proc. of the National Aviation University, no.2, pp.98-103. **doi**: 10.18372/2306-1472.24.1152

[3] Philips Healthcare (2009) *Philips Healthcare – C# Coding Standard*, Philips Healthcare, p. 57.

[4] Blake J., Cirtis P. (2007) *CERT C Programming Language Secure Coding Standard Document* N1255, Carnegie Mellon University, 488 p.

[5] ESA BSSC (2005) *Java Coding Standards, Prepared by: ESA Board for Software Standardization and Control*, Issue 1.0, PARIS CEDEX, France, 113p.

[6] Rosen J-P. (2008) *A comparison of industrial coding rules.* Ada User Journal, vol. 29, no. 4, pp. 1-5.

[7] Stallman R. (2016) *GNU Coding Standards*, July 25, pp.80.

[8] *Coding Standards in the Real World* http://submain.com/codeit.rigth

[9] Sidorova N.N. (2015) *Ontology-driven method using programming styles,* Software engineering, no.2 (22), pp 19-29. (In English)

[10] Haijie Z. (2009) *Developing a System to Help Programmers Achieve a Good Coding Style* [A dissertation submitted in partial fulfillment of the requirements of Dublin Institute of Technology for the degree of M.Sc. in Computing (Information Technology)], Dublin, 124 p.

[11] Levinson H. L., Librizzi R. M. (2013) *Using Software Development Tools and Practices in Acquisition*, Technical note CMU/SEI-2013-TN-017, Carnegie Malone University, Software Engineering Institute, 28p.

[12] Sidorov N.A., Sidorova N.N. (2016) [Programming style ontology-driven tools]. Abstracts of the International Scientific and Practice Seminar for Young Scientists and Students, Луцьк – НТУ, с.100. (In English)

[13] Black P. (2009) *Static Analyzers in Software Engineering, CrossTalk*, The Journal of Defense Software Engineering 16-17 March-April, Software Engineering, pp. 1617,

[14] Protégé at: http://protege.stanford.edu/

[15] Dentler K., Cornet R., Teije A., Keizer N. *Comparison of Reasoners for large Ontologies in the OWL 2 EL* Profile at: http://www.semantic-web-journal.net/sites/default/files/swj120_2.pdf

[16] Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P.F. (2003) *The Description Logic Handbook*, Cambridge University Press, 320p.

[17] OWL API at: http://owlapi.sourceforge.net/

[18] Java API Specifications, at: http://www.oracle.com/technetwork/java/api-141528.html

Received 23 May 2017

М.О. Сидоров^1, Н.М. Сидорова^2, О.О. Пирог^3

Керований онтологією інструмент для застосування стилей програмування

Національний авіаційний університет, просп. Космонавта Комарова, 1, Київ, Україна, 03058
E-mails: ^{1} nikolay.sidorov@livenau.net; ^{2} nika.sidorova@gmail.com; ^{3}pirogman@gmail.com

Діяльність програміста буде більш ефективною, а програмне забезпечення зрозумілим коли при створенні програмного забезпечення застосовуються стилі (стандарти) програмування, які забезпечують зрозумілість програмних текстів. **Мета:** В цьому дослідженні представлено засіб для реалізації нового методу, який застосовує онтології і автоматизує процеси використання стилів програмування. Зокрема розглядається представлення стандартів в формі онтології і застосування ризонеру дескриптивної логіки для програміста. Метою статті є розробка засобу для підтримки керованого онтологією застосування стилів програмування. **Методи дослідження:** онтологічне представлення стилів програмування; об'єктно-орієнтоване програмування; кероване онтологією застосування стилів програмування. **Результати:** отримано архітектуру та мовою Java реалізовано засоби які забезпечують інструментальну підтримку методу керованого онтологією застосування стилів програмування. На прикладі іменування стандарту програмування мови Java надано особливості реалізації і застосування засобу. **Обговорення:** Застосування стандартів програмування при конструюванні програмного забезпечення, відсутність засобів автоматизації процесів застосування стандартів програмування, засіб на основі нового методу керованого онтологією застосування стилів програмування, приклад реалізації архітектури засобу для іменувань стандарту мови Java.

Ключові слова: дескриптивна логіка; інженерія програмного забезпечення; онтологія; програмування; ризонер; стандарт кодування; стиль програмування.

Н.А. Сидоров[1], Н.Н. Сидорова[2], А.А. Пирог[3]

Управляемый онтологией инструмент для использования стилей программирования

Национальный авиационный университет, просп. Космонавта Комарова, 1, Киев, Украина, 03058
E-mails: [1] nikolay.sidorov@livenau.net; [2] nika.sidorova@gmail.com; [3]pirogman@gmail.com

Деятельность программиста будет более эффективной, а программное обеспечение понятным когда при создании программного обеспечения применяются стиле (стандарты) программирования, которые обеспечивают ясность программных текстов. **Цель:** В этом исследовании представлены средство для реализации нового метода, который применяет онтологии и автоматизирует процессы использования стилей программирования. В частности рассматривается представление стандартов в форме онтологии и применения ризонеру дескриптивной логики для ассистирования программиста. Целью статьи является разработка средства для поддержки управляемого онтологии применения стилей программирования. **Методы исследования:** онтологическое представление стилей программирования; объектно-ориентированное программирование; управляемое онтологии применения стилей программирования. **Результаты** получены архитектуру и на языке Java реализовано средства, которые обеспечивают инструментальную поддержку метода управляемого онтологии применения стилей программирования. На примере именования стандарта программирования языка Java предоставлено особенности реализации и применения средства. **Обсуждение** Применение стандартов программирования при конструировании программного обеспечения, отсутствие средств автоматизации процессов применения стандартов программирования, средство на основе нового метода управляемого онтологии применения стилей программирования, пример реализации архитектуры средства для именований стандарта языка Java.

Ключевые слова: дескриптивная логика; инженерия программного обеспечения; онтология; программирование; ризонер; стандарт кодирования; стиль программирования.

Sidorov Nikolay. Doctor of Engineering. Professor.
Head of software Engineering chair of the National Aviation University.
Education: Taganrog Radio Technik, Institute, Taganrog, Russia.
Research area: software engineering
Publications: 130.
E-mail: nikolay.sidorov@livenau.net, sna@nau.edu.ua

Sidorova Nika. Postgraduate student.
Department of Software Engineering. National Aviation University, Kyiv, Ukraine
Research area: Software Engineering
Publications: 12.
E-mail: nika.sidorova@gmail.com

Pirog Alexander. Undergraduate student.
Department of Software Engineering, National Aviation University, Kyiv, Ukraine (2014)
Research area: Software Engineering
E-mail: pirogman@gmail.com