ТЕХНОЛОГІЇ РОЗРОБКИ ТА СУПРОВОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

UDC 004.683.01

Sidorova N.M.
National Aviation University

ONTOLOGY-DRIVEN PROGRAMMING STYLE ASSISTANT

Programming style is the set of guidelines, and practices, applicable to a specific language, that are used while writing the source code and which are intended to introduce the universal look and feel of the code, improve understandability of the code and assist software engineers to not introduce more mistakes while writing code. Programming styles are different for different languages.

To develop the ontology, one must first perform the extensive research on the topic, basically a domain analysis on the problem, identify the main concepts of programming styles, build a hierarchy of them and define the relationship between the concepts and their members.Ontology-driven programming style assistant was developed with the help of Protégé. Ontology reasoning is a process in any ontology development to ensure that the ontology is of high-quality and does not contain any contradictory relations.

For programming styles ontology was choice HermiTreasoner, because it proved to be one of the fastest, reliable and widely used reasoners in the current state of things in the ontological domain.Protégé tool also automatically collects metrics about the ontologies, which are mostly concerning the numbers of class axioms, object property axioms and other important numbers.Ontology development using modern software tools is one of the key problems ontology domain. It is important that the users are able to create ontologies with ease and operate with description logic expressivity in the very clear and concise manner.

Застосування стилів програмування при написанні програм зараз, у зв'язку з вирішенням завдань супроводу і повторного використання программного забезпечення, як і раніше актуально. У статті пропонується інструмент для асиствування програміста в процесі конструювання программного забезпечення, заснований на представленні стилю програмування онтологією. Для реалізації інструменту використовується Protégé. Розглянуто технічні питання організації онтології. Виконано дослідження реалізованого інструменту і показана його працездатність.

Применение стилей программирования при написании программ сейчас, в связи с решением задач сопровождения и повторного использования программного обеспечения, по-прежнему актуально. В статье предлагается инструмент для ассистирования программиста в процессе конструирования программного обеспечения, основанный на представлении стиля программирования онтологией. Для реализации инструмента используется Protégé. Рассмотрены технические вопросы организации онтологии. Выполненоисследованиереализованногоинструментаипоказанаегоработоспособность.

Keywords:*softwareengineering, programming, programmingstyle, ontology, Protégé, reasoner.*

Introduction

Programming style is the set of guidelines, and practices, applicable to a specific language, that are used while writing the source code and which are intended to introduce the universal look and feel of the code, improve understandability of the code and assist software engineers to not introduce more mistakes while writing code [1]. Programming styles are different for different languages.

What works for one language may not work for another due to the language specifics and the corresponding code structure. Even within the same language there can be a variety of different programming styles used for different situations and different programmer and differentsoftware teams. For example, the particular project team (Mozilla developers) may rely in their work on the Java official programming style, but they do have the list of concepts that is different from the official guide and that is universally adopted within their project and is fulfilled. This makes up for their own programming style and it is no less correct one than the official Java style guide.

During the evolution of the programming styles, the following list of relevant concepts was formed: white space, indentation, vertical alignment, naming conventions, comments, portability, meaningfulness, consistency, file structure, complexity. Generally, programming style is something that software engineers enjoy not to address in order to save time and effort while developing software applications, as it can be challenging to follow the fruitful variety of boring guidelines and it usually slows down their work significantly.

Yet, the years and years of the software development have led the community to believe that adhering to programming styles is a highly-beneficial practice which allows to save a lot of time and money, since software programs are usually supported and used for quite a broad periods of time [2]. The more impactful the software project is, the more people involved in the development are, the more programming styles are coming in handy.

Ontology Design

To develop the ontology, one must first perform the extensive research on the topic, basically a domain analysis on the problem, identify the main concepts of programming styles, build a hierarchy of them and define the relationship between the concepts and their members.

The research session has provided the basic notions about the programming styles and the variety of actual standards to work through [3,4]. This allowed to gather actual concepts that later will be formed into the programming styles ontology and to trace the connections between these concepts. We proposed the list of programming styles ontology high-level concepts (is called ‘classes’) that is the following: Programming Languages, Projects, Standards, Style Rules.

Programming Languages

It is clear that the programming styles are influenced by the languages. A lot of languages have code conventions developed for them, available in free access and ready for usage. The search for programming style conventions allowed composing a list of languages, which have official or unofficial code conventions developed for them.

In terms of the ontology, the ‘Programming Language’ is the class, while each specific language is considered to be the member of the class. For example, Action Script, Ada, C, C++, C#, D, Dart, Erlang, Flex, Java, JavaScript, Lisp, MATLAB, Mono, Object Pascal, Perl, PHP, Python, Shell are the members at the class.All these languages have the programming styles associated with them.

Projects

Programming style guides are not limited to languages exclusively.It is also easy to find open-source standards for various software projects. In a lot of cases, the standards for the projects are heavily relying on some official language guides, but there are cases when the project themselves are setting the coding conventions that influence pretty much everyone who are working with the language. There is quite a big number of programing styles and code conventions for the projects as well.

In terms of ontology, ‘Projects’ are the ontology class, while particular projects are treated like the members of the class. The list of projects with found standards that are being included in the ontology is the following: Apache, Drupal, Zend, GNU, Google, Linux, Modular, Liquid Web CMS, Mozilla, Road Intranet, NetBSD, OpenBSD, GNAT,ZeroMQ.

Standards

Next step of gathering the data for the ontology was choosing the standards and getting familiar with their structure and rules, so the domain analysis could be finished.

The standards have different document structures, they can be presented to the end user in different forms and they can contain the different level of detail within them. Such an analysis resulted in the need to classify the standards further and to introduce sub-classes into the ontology.The final structure of the ‘Standards’ ontology class is the following (Fig.1):

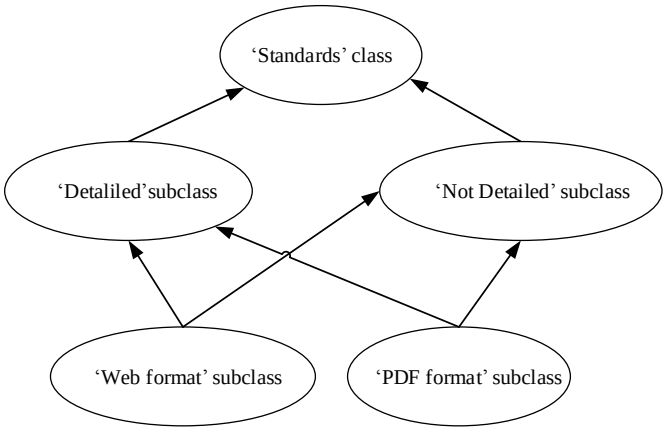

Fig. 1. The structure of the “Standards” ontology class

The actual standards are bound to become the members of the ‘Web format’ or ‘PDF format’ subclasses. It is important to note that the whole point of the ontologies is about defining consistent knowledge bases, so this requires to put a disjoint restriction between the subclasses of the same level while designing the ontology using ontology editorthe example ofthe final choice for the detailed ontology analysis is on the tab. 1.

Table 1
Detailed Analysis Standards

Standard	Language	Project
C# Coding Conventions (C# Programming Guide)	C#	–
C# Brad Adams		
Philips Healthcare		
Code Conventions for the Java Programming Language	Java	–
Mozilla Coding Style Guide		Mozilla
Apache Developers' C Language Style Guide	C	Apache
GNU Coding Standards		GNU

The specific rules and guidelines that have made it into the ontology for this work are taken from the abovementioned standards [5].

Style Rules

The work with actual style guide rules is one of the most important parts of the ontology development, since it is necessary to analyze the rules and draw connections between them. The code conventions usually have the similar structure and the rules themselves are concerning the same topics. For example, [5], rule 3@101-use us-English for naming identifiers. There are a lot of cases where the same rule is described with the help of different words and it is highly important that for the ontology it is clear whether the rule is unique or is there the same rule within any other standard. We are to classify the rules. This helps to keep the consistency within the ontology (which is one of its most important quality characteristics) and make sure that there are no synonymous members.

The important step is also to classify the programming style rules into concepts, since there is clearly a variety of different groups the rules. The classification from Philips Healthcare standard was taken into account for this ontology [5], since it does provide the most comprehensive level of logic and detail for the ontology concepts, although it was modified a bit, as the standard itself missed the file structure section. Majority of other standards are using the same structure and naming as well.

The final list of style rule types consists of 10 elements and is the following: General, Naming, Comments, Object Lifecycle, Control Flow, Object Oriented, Exceptions, Delegates & Events, Data Type, Coding Style & File Structure.The style rules themselves are further classified in accordance with above mentioned style rule types and serve as ontology members or objects within the rule types in ‘Style Rule’ class.

The problem with style rules is such that the descriptions of the rules usually require quite a rich amount of sentences. It is clearly not possible to use the description themselves because of it as the names for the style rules. Also, it is extremely hard to choose the short and meaningful name for the rule without losing the sense of the rule itself and without the risk of the rule sounding too similar to the other or being redundant. This has called for the need to adopt the numbering convention to be able to properly describe the style rules within the ontology.

The table 2 presents some examples of style rules with regards to the numbering convention, style rule types, their descriptions and connections to standards, languages and projects.

These numbering conventions are further used when the ontology is being developed in Protégé editor, just as the connections between the rules and other ontology elements are established as well.

Table 2
Style Rules Examples

#	Description	Standard	Language	Project
General				
1.1	Do not mix code from different providers in one file	Philips	C#	–
Naming				
2.1	Use US-English for naming identifiers	Philips	C#	–
		GNU	C	GNU
Comments				
3.1	Each file shall contain a header block	Philips	C#	–
Object Lifecycle				
4.18	Avoid assigning several variables to the same value in a single statement	Apache	C	Apache
Control Flow				
5.1	Functions should be declared with ANSI-style arguments	Philips	C#	–
Object Oriented				
6.1	Do not use embedded assignments in an attempt to improve run-time performance.	Philips	C#	–
Exceptions				
7.14	Do not throw an exception from inside an exception constructor	C# Conv	C#	–
Delegates & Events				
8.10	Each subscribe must have a corresponding unsubscribe	C# Conv	C#	–
Data Types				
9.12	Explicitly declare the types of all objects	C# Conv	C#	–
Coding Style & File Structure				
10.29	Do not use spaces inside brackets	Adams	C#	–

Ontology Model

Prior to developing the ontology using Protégé editor, it is vital to establish the proper connections between all the classes. In general, it is a popular way to model the ontologies using basic UML class diagram concepts, so this method was chosen to present the ontology structure [6].

Using the relations, the model for the programming styles ontology was developed(Fig.2). Note please, that it deals with the level of ontology classes and their sub-classes, without going down to the detail of the particular class members

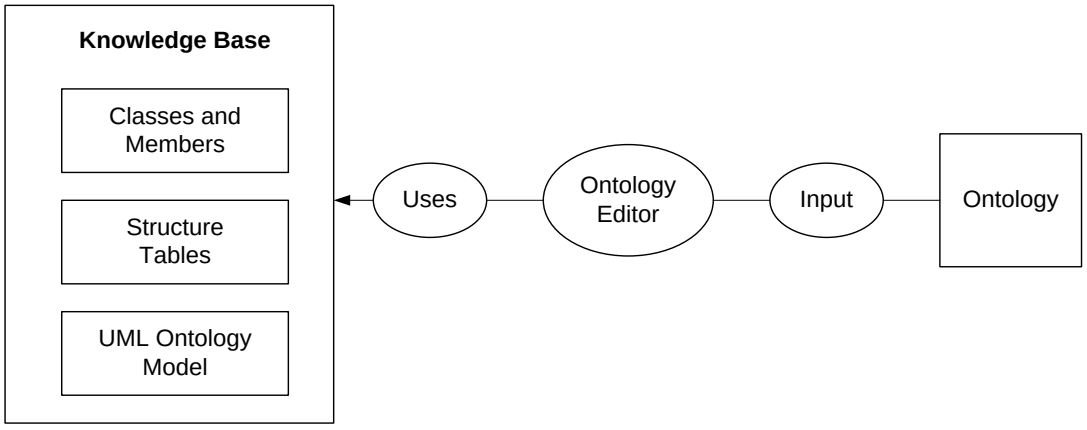

Fig. 2. Programming Styles Ontology Model

So, the ontology was presented in the way of design tables, as well as in the form of UML ontology model that shows the relations of classes between each to other. The obtain information is significant enough and is comprehensively described, that allows to build the programming styles ontology using the Protégé ontology editor.

Case study. Ontology Development Using Protégé

Ontology-driven programming style assistant was developed with the help of Protégé[7,8].

Protégé is an open-source knowledge acquisition system, developed by Stanford University. Protégé is considered to be the leading ontology engineering tool. It provides the means of visual ontology modeling. Users can interact with the help of the interface to create classes, add members, create and assign properties between ontology concepts. Protégé also allows generating Java code from the ontology models and supports a number of languages and ontology representations, such as RDF, OWL, XML and so on.

Just as Protégé allows creating ontologies and editing them, it also provides the unique opportunities to actually present ontologies in a very user-friendly form. Description logic ontology form is great for ontology logic purposes, but it is hard to read and navigate through [9]. The UML ontology form mostly presents the model of the ontology, but it certainly lacks any Description logic expressivity and in general is lacking details. Protégé capabilities allow presenting ontologies no worse than a PDF document of web page can. This is vital when programmer needs to work with the programming styles ontology as the knowledge base for his coding needs.The scheme on fig.3 present the logic of ontology development.

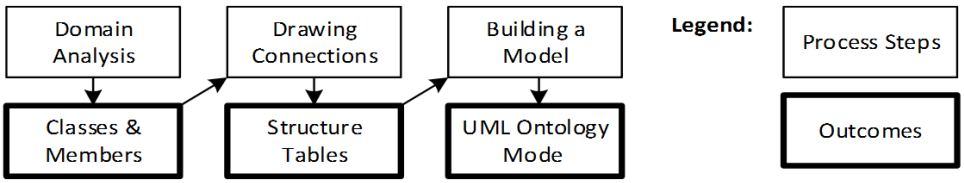

Fig. 3. Ontology creation process scheme

While creating the ontology in Protégé, there are several the following ground rules:

- classes are sets or collections of objects;
- members or individuals are the instances of the classes – basic components of the ontology;
- restrictions or assertions help define relations between classes and between members;
- it is impossible for a class to be directly related to any other class member;this interaction happens solely on the level of class members;
- object properties are used to set restrictions or assertions;
- annotations help presenting any information about ontology the user needs, usually it is used to handle descriptions, comments or even some examples for the ontologies.

Protégé also automatically collects metrics, which will be presented later.

Creating the ontology consists of any steps.

The first step of creating the ontology is defining its classes. Fig. 4 presents the programming style ontology classes in the view of their defined hierarchy.

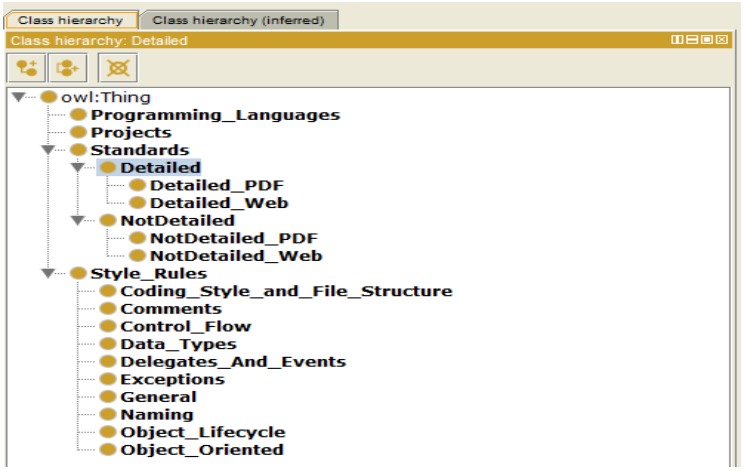

Fig.4. Programming styles ontology classes hierarchy

The ontology contains classes and sub-classes as well. There is no restriction as for the class nesting, as long as there are no contradictions or inconsistencies caused by this.The specific class objects are stored at the level of class members. It is worth noting that not all classes contain members in this programming styles ontology – members usually follow on the lowest level of detail.The screenshot is also presenting the members view from so-called 'Classes' view in the editor (Fig.5).

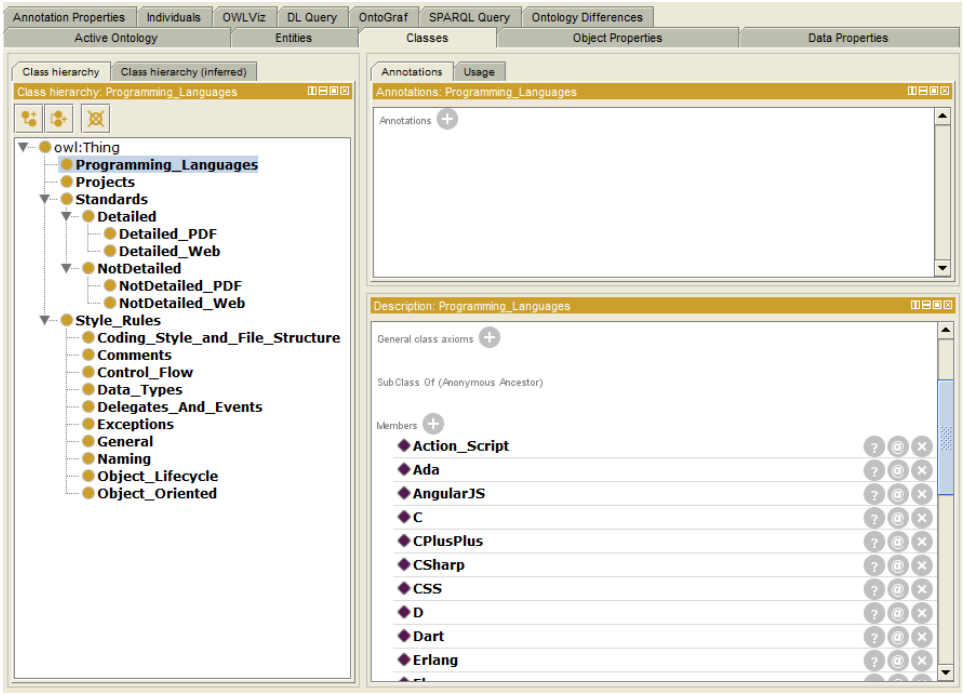

Fig.5. 'Class' ontology view with members

Class view within Protégé ontology editor deals with all kinds of properties and relations, connected in particular to the class view. The editor holds a great variety of different interaction options for ontology classes. The structure of options is the same for each ontology class, whereas user himself defines and adds the needed options which are logical for this particular class. Protégé ontology editor is not bounded by class view only. There are a lot of the cases where it is much easier to work with members view, and the tool presents it with the same easy and intuitive user interface. To work with members, ontology editor needs to use the 'Individuals' menu, which allows seeing the list of members and their specific properties but still not losing the connection with the classes.

There is an important point to note about members of any subclass of class 'Style Rules'. It was mentioned before that introduced numbering was the only way of somehow maintaining the structure of the ontology and do not overload it with long and potentially non-meaningful or redundant member names. Protégé as an instrument contains the powerful feature called 'Annotations' that allows using 'comment' property, where ontology editor can store any information about the member or the class. This functionality was successfully utilized when describing various style rules (Fig.6).

The final view of style rules concepts in ontology editor is such that the name of the member consists of '<Style Rule><chosen designation number>'.

The important point not to miss while developing the ontology is also the examples of code usage, in case that is actually present within the standard. Since the goal of the programming styles ontology is to allow programmers to easily navigate through it and facilitate learning, it is a vital point to contain some examples. Protégé ontology editor contains the functionality to include the examples. Just as ontology should resemble the knowledge base structure and relations between its elements, it should also be informative enough to allow easy learning process for any user.

Protégé also allows creating user-defined properties and does not restrict editor with the list of available properties. It also separates user-defined properties from default ones using bold

font. This allowed introducing the special, editor-custom ‘example’ property.

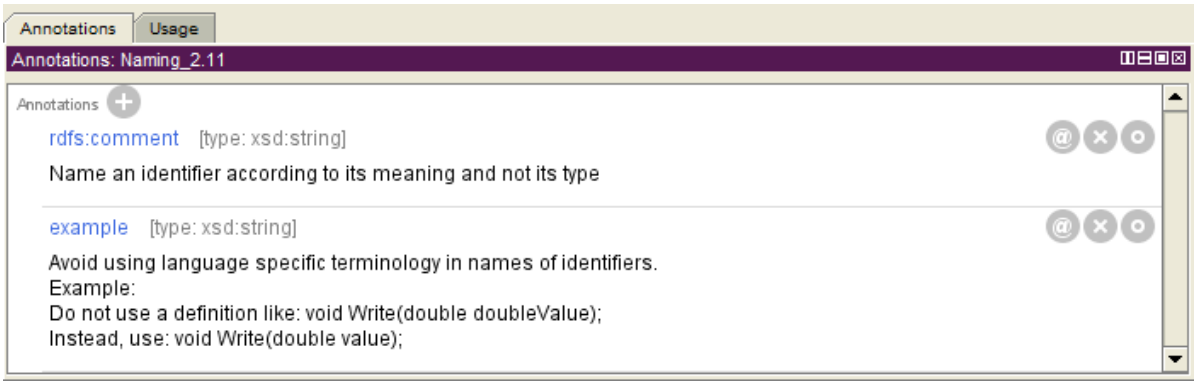

Fig. 6. Comment and example for ‘Naming’ class member

Certain properties are related to the classes and certain ones are related to members of these classes and establish the connection between them. It is important that these object properties are basically an analog of description logic expressivity within the Protégé tool [9]. Instead of writing the description logic queries, user can easily see all the needed axioms in the form that is merciful to a human mind and easily understandable. It is important that subclasses are inheriting the object properties relations from their parent classes or ancestors, which means that there is no class in this ontology without the relations. The tab.3 presents example of the object properties for classes and members defined for the domain of programming styles.

Table 3
Object Properties for Classes

Class	Property
Programming Languages	containStyleRules some Style_Rules useStandards some Standards

Another powerful feature of Protégé tool is the so-called ‘usage’ view, which allows displaying instantly where the particular member or class are actually used within the ontology (Fig.7). The ‘usage’ menu captures all kinds of connections between elements and presents them in a user-friendly form, also allowing a bit of sorting for distinguishing between different usage types. It also automatically calculates the number of occurrences of the particular class within any type of relation.

Ontology Reasoning

Ontology reasoning is a process in any ontology development to ensure that the ontology is of high-quality and does not contain any contradictory relations.

The Protégé tool supports a rich variety of different ontology reasoners as plug-ins, ready to download directly using the tool. Once installed, any ontology opened in Protégé can be reasoned.

For programming styles ontology was choice HermiTreasoner, because it proved to be one of the fastest, reliable and widely used reasoners in the current state of things in the ontological domain [10].

In general, a HermiTreasoner is capable of performing such checks as:

- determine whether a description of the concept is not contradictory;
- determine whether members in ABox do not violate descriptions and axioms described by TBox;
- check whether the member is an instance of a class;
- find all members that are instances of a class;
- find all concepts which the member belongs to.

To invoke the reasoner using Protégé, one should go to ‘Reasoner’ menu, choose the needed reasoner in the list of available ones and click button ‘Start reasoner’.

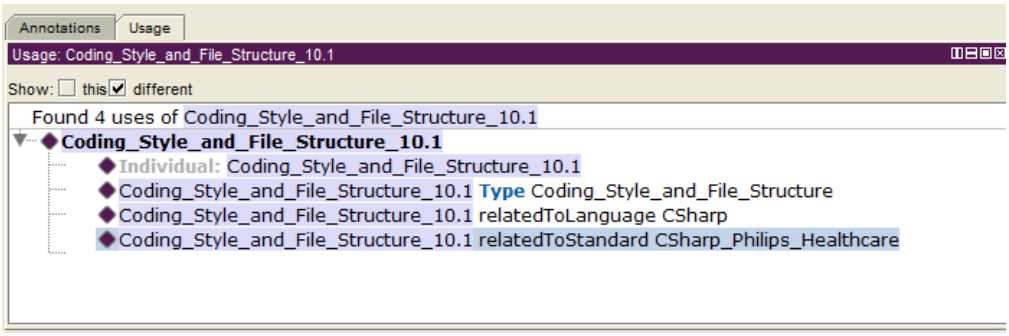

Fig. 7. Example of 'usage' view for the ontology member

Because all 'Style Rules' subclasses are later set to be disjoint with each other to make that there are no duplicates within the classes, the reasoner catches the illogical expression and immediately renders the obtained ontology as the inconsistent one (Fig.8).

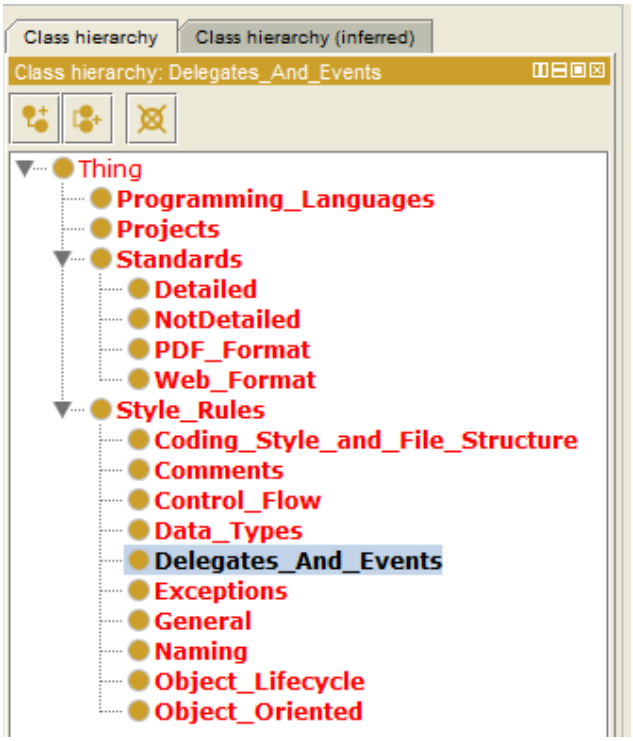

Fig. 8. Inconsistent ontology class hierarchy

Protégé ontology editor also provides the error messages in case the ontology is inconsistent, as it logs all issues that occur while tool is being run (Fig.9).

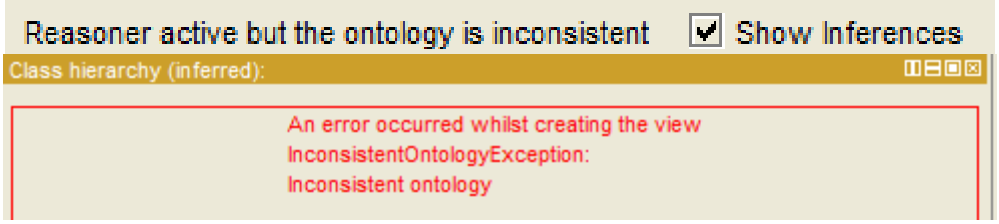

Fig. 9. Error in inferred ontology

An important point about reasoners is that it also allows drawing connections between objects in case there is a sufficient description logic basis for it. Such a case was uncovered while working with the programming styles ontology as well.While the ontology should be consistent and ideally there should be no synonyms, there are certain cases where elements can be similar in their nature, but still be different objects in terms of the ontology.

Protégé tool also automatically collects metrics about the ontologies, which are mostly concerning the numbers of class axioms, object property axioms and other important numbers.

The programming style ontology metrics summary can be found in tab. 4.

Table 4
Programming Style Ontology Metrics

Metrics

Axiom	1878
Logical axiom count	1223
Class count	20
Object property count	8
Data property count	0
Individual count	344
DL expressivity	ALCHO

Class axioms

SubClassOf axioms count	33
EquivalentClasses axioms count	0
DisjointClasses axioms count	4
GCI count	0
Hidden GCI Count	0

Individual axioms

ClassAssertion axioms count	344
ObjectPropertyAssertion axioms count	838
DataPropertyAssertion axioms count	0
NegativeObjectPropertyAssertion axioms count	0
NegativeDataPropertyAssertion axioms count	0
SameIndividual axioms count	1
DifferentIndividuals axioms count	2

Annotation axioms

AnnotationAssertion axioms count	283
AnnotationPropertyDomain axioms count	0
AnnotationPropertyRangeOf axioms count	0

Conlusions

Ontology development using modern software tools is one of the key problems ontology domain. It is important that the users are able to create ontologies with ease and operate with description logic expressivity in the very clear and concise manner.

Another important notion is making sure that the ontology is consistent, does not contain any contradictory notions and hence is high-quality and useful for the community. Ontology reasoners, usually in the form of plug-ins for the ontology editors are helping precisely with that. Ontology development itself is more fruitful in case the editor is familiar with general principles of reasoning and is able to eliminate most of the ontology inconsistencies by keeping in mind how description logic and reasoning themselves are functioning.

As the result of ontology development and reasoning, the consistent programming styles ontology was developed, containing a substantial amount of concepts, members and having the meaningful connections drawn and verified via the help of HermiTreasoner using the Protégé ontology editor tool [3, 5]. Even more importantly, the developed ontology contains the capabilities to use it as the tool to learn programming styles, as it supports the description of a programming style using annotations, that allows it to contain the code snippets examples, connected to the rules.

References

1. Sidorov N.A. Software stylistics [текст] /Sidorov N.A.// Proc. of the National Aviation University, 2005. – №2. – С.98–103.

2. Railich V., Wilde N. et. al. Software cultures and evolution Computer. – 2001, Sept. – P. 25 – 28.

3. Sidorova N. N. Programming styles taxonomy [текст] /Sidorova N. N.// Наук. журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво" – Луцьк.: Луцькийнаціональнийтехнічнийуніверсите, № 19.– 2015.– С. 79–85.

4. Sidorova N. Ontology-Driven Method Using Programming Styles [текст]/ Sidorova N./ – "Інженерія программного забезпечення". – 2015. – № 2 (22). –С. 19 – 28.

5. C# Coding Standard, Version 2.0, Philips Healthcare, 2009.– 57p.

6. Grants E.S. Roadmap to a DO-178C Formal Model– Based software Engineering Methodology.Proc. of the Intern. Multiconf. Of Eng.Comp. Sci. – V.1, IMECS. –March.– 2015. – 6p.

7. Sidorov N., Sidorova N. Programming style ontology-driven tools.[текст]/ Sidorova N., Sidorova N. //Programmable logic integrated circuits and microprocessorteehnignein education and manufacturing. -abstr. of the Intern. Scient. and Pract. Workshop Young Scientists and students. –Луцьк. – 28-29.04.2016. – P.100–101.

8. The description logic handbook Theory, implementation, and applications, Ed. by F. Baader, Cambridge University Press. – 2003. – 320 p.

9. SidorovaN.M.Theprogrammingstyleontology assistant. - Тези доповідей Міжнародної науково-практичної конференції аспірантів і студентів. –Київ. – 2016.–P. 16.

10. S. Abburu, "ASurveyonOntologyReasonersandComparison", InternationalJournalofComputerApplications (0975 – 8887), Volume 57– No.17, November 2012.

Information about author:

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

E-mail: nika.sidorova@livenau.net