PasOnto: Ontology for Learning Pascal Programming Language

Baboucar Diatta
Department Information and Communication Technology
University Alioune Diop
Bambey Sénégal
baboucar.diatta@uadb.edu.sn

Adrien Basse
Department Information and Communication Technology
University Alioune Diop
Bambey Sénégal
adrien.basse@uadb.edu.sn

Samuel Ouya
Department Computer Engineering
University Cheikh Anta Diop
Dakar Sénégal
Samuel.ouya@gmail.com

***Abstract*— The Pascal programming language is one of the most pedagogical languages to learn coding. The best way to improve coding skills is through practice. In that regard, we give in our programming course the opportunity to practice the Pascal programming language concepts by providing exercises with solutions. To enhance learners' understanding, research suggests to represent exercises querying using ontologies. Nevertheless, such research focuses mainly on object-oriented programming (Java, C#) and C programming language. For this reason, this work proposes PasOnto ontology to describe exercise solutions essentially in Pascal programming. We also propose a web and mobile interface around this ontology so as to query and highlight Pascal keywords in displayed exercise solution.**

Keywords—programing language, Pascal programming, ontologies, distance education.

I. INTRODUCTION

Computer programming skills can open countless information technology careers including web and software developers, computer systems analysts, computer and information research scientists. Our first-year students start learning programming through algorithm and Pascal programming language course. The procedural Pascal language was developed as a language suitable for the teaching of programming with concepts that easily translate into algorithm. Thus, Pascal is easy to learn and produces efficient and reliable programs.

Improving programming skills requires practicing rules, basics, and core concepts of Pascal programming language (how to declare variable, to use selective or loop structure, to define functions and procedures , …). In that regard, our programming course proposes many exercises with solutions ranging from basic to more complex exercises. So, student have the possibility to practice programming by visualizing solutions and comparing them with their work. Nevertheless, proposed solutions are not semantically structured to be queried efficiently.

To better exploit the potential of exercise databases, research suggests to represent their content using ontologies. Indeed, ontologies defined as "explicit formal specifications of the terms in a domain and relations among them" [1], provide not only semantic description of domain concepts but also new knowledge. But still, research about exercise databases focuses mainly on representing and querying exercises as well as programming domain knowledge [2] or mainly Java, C and C# programming languages ([3][4], [5] and [6]).

In this work, we propose PasOnto ontology to describe the exercise solutions essentially in Pascal programming language so as to enhance learners' querying. We also provide a web and mobile interface to retrieve available Pascal exercises and to highlight Pascal keywords in displayed solution. This interface provide new knowledge like exercice difficulty level and syntactic prerequisites from exercice solutions. Such new knowledge helps to categorize exercice by difficulty level and to guide learners on requisite programming course chapters during solution writing process.

Section II of the paper contains a review of the literature on programming language ontologies. Section III describes the adopted development process of ontologies and designed ontology for Pascal programming language. Section IV shows an example of how to use this ontology and section V is the conclusion.

II. RELATED WORK

The development and deployment of ontologies have many benefits including "sharing common understanding of the structure of information among people or software agents and, enabling reuse of domain knowledge" [7]. Therefore, research has proposed many ontologies to enhance understanding and querying of programming domain. Proposed programming language ontologies focus mainly on representing c language elements or object-oriented concepts through Java and C# programming languages.

Among object-oriented languages, research focuses mainly on Java programming language. For instance, to model object-oriented concepts and features, [8] and [9] have proposed two ontologies with a component to automatically populate them from Java code. [4] proposed a Java ontology to capture the semantics of Java concept with the reserved words of the language, the general object-oriented concepts (class, constructor, interface, …), the different types of statements, ... As [4], [5] proposed a Java ontology representing Java language elements with the view to helping teachers design and organize the Learning Objects ("smaller learning unit to combine in order to build course") and help learners to better interact with the computer programming courses. [10] uses an ontology to describe tools used for Java programming and the Java language elements. [3], to facilitate the learning of Java programming to young Bulgarian programmers, capture in a bilingual (English and Bulgarian) ontology the semantic of Java and object-oriented concepts.

Another represented object-oriented language is C#. Because Java ontologies like SCRO ([8]) use elements strongly dependent on Java language like Java.lang.String, [11] decided to construct the new ontology CSCRO to represent object-oriented concepts but dedicated to C#.

Ontologies are also dedicated to C programming language. C is a procedural language like Pascal and is based upon the concept of the procedure call. In addition to proposed Java ontology, [5]captured the semantic of C language elements. As [5], [6] proposed an ontology for

978-1-5386-9506-7/19/$31.00 ©2019 IEEE 9–11 April, 2019 - American University in Dubai, Dubai, UAE
2019 IEEE Global Engineering Education Conference (EDUCON)

teaching/learning C programming. This ontology represents the basic syntactic elements of C language and other knowledge including helpful programming techniques.

The above ontologies focus mainly on object-oriented programming languages (Java and C#) and on C programming language. Because they use elements strongly dependent on a particular language (Java, C#; C), we choose to design pasOnto ontology to describe exercise solutions essentially in Pascal programming language.

III. PASCAL PROGRAMMING LANGUAGE ONTOLOGY

Our first task is to construct Pascal programming language ontology to capture the semantics of Pascal language elements. In this section we describe the adopted methodology to construct our ontology and the resultant ontology.

In the literature, there are many methodologies ([12], [13], [14]) that address the issue of ontology development. In order to achieve PasOnto, we mainly adopt the methodology proposed by [7] and used to construct, for instance, Java ontology [4]. From Noy and McGuinness methodology ([7]) we choose mainly the five following steps:

1. **Determine the scope of the ontology:** This first step helps to limit the scope of the model by clarifying its purpose and the type of expected queries and answers;
2. **Enumerate significant terms**: In this step we write down all significant domain terms which appear in queries or answers defined in step 1;
3. **Define the classes and class hierarchy:** From the list of terms created in step 1, we pick up the classes and organize them into a taxonomic hierarchy;
4. **Define properties of classes with their facets**: At first, this step consists in identifying the properties (datatype properties) of each class and the interactions (object properties) among classes. Then, for each of this two kinds of properties, we define their domain, range and if necessary property restrictions (value constraints and cardinality constraints);
5. **Create instances:** In this step, we populate the ontology with individual instances of classes created in step 2 with their properties.

We construct our ontology by following the five steps listed above. We begin with determining the main goals and intended use of PasOnto.

A. Scope of PasOnto ontology

PasOnto has been proposed to conceptualize Pascal exercise solutions in order to enhance learners' program understanding and querying. Thus, PasOnto is expected to contain exercise descriptions and all Pascal language constructs so as to describe a Pascal program.

B. Significant terms

Since C and Pascal are procedural programming languages and so have a set of concepts in common, we particularly rely on C programming language ontologies during this step. We also consult some Pascal programming courses[1,2]. All those terms have been validated by an expert of the domain that is responsible for the first year programming course in our university.

Figure 1 shows a part of the concept map developed in this step. Concept map and mind map are very useful in this brainstorming step especially with the available conversion algorithm from concept map to ontology ([15], [16]). The represented concepts in the concept map include:

- *exercise* to formulate the problem;
- *Pascal program*, the main node of PasOnto, to represent solution exercise;
- *variable*, *constant, label, subprogram* and *type* declarations which are at the top of the Pascal program and for some of them in subprogram;
- *procedure, function* and *mainSubprogram* for subprogram definition;
- *simple statement* of Pascal language like *inOut* statements, return statement, …;
- *compound statements* (loop and decision statements) of Pascal programming language;
- *Variable, Parameter, Type* and *Constant which represent Pascal language* elements.

Fig. 1. Part of concept map of significant terms

C. Classes and class hierarchy

From this step, we use the Protégé[3] editor to build PasOnto ontology in particular our class hierarchy from the significant terms identified in the previous step. The ontology editor Protégé provides complete OWL 2 [17] editing capabilities and latest rdf specification from the World Wide Web Consortium (W3C). Protégé is an extensible open source environnement with a strong and active community of users and developers.

Figure 2 shows a part of class hierarchy obtained with mainly:

- *Data* class and its subclasses *constant* and *variable*. The class *variable* has a subclass *parameter* linking to subprograms;
- *Declaration* class and its subclasses to make the label, constant, variable, subprogram and type declarations. *DataDeclaration* class has two

[1] https://www.tutorialspoint.com/pascal/
[2] https://www.pascal-programming.info/lesson1.php
[3] https://protege.stanford.edu

subclasses: ConstantDeclaration subclass and VariableDeclaration subclass to declare constants and variables, respectively. SubProgramDeclaration class contains FunctionDeclaration and ProcedureDeclaration subclasses;

- *Statement* class with two subclasses: *CompoundStatement (loopStatement* or *DecisionStatement)* which contain statements and *SimpleStatement* like *assignmentStatement*, *InOutStatement and returnStatement*. *InOutStatement* has in turn two subclasses: *ReadStatement* and *WriteStatement* class. *ForStatements*, *WhileStatement* and *RepeatStatement* are the three subclasses of *LoopStatement*. *DecisionStatement* contains *CaseStatement*, *IfStatement* and *ElseStatement*;
- *SubProgram* class with three subclasses: *DefinedSubProgram* of the programming language, *mainSubProgram* representing the main begin …end block, *UserSubProgram. A subprogram is a block of statements that carries out particular task. DefinedSubPrograms* are the built-in subprogram, which are automatically declared by the Pascal compiler. Pascal standard library proposes numerous built-in subprogram that programs can call. *UserSubPrograms* are User-Defined subprogram and help developers to define and use their own subprograms;
- *Type* class has three subclasses: *PointerType*, *ScalarType* and *StructuredType*. *ScalarType* class contains *StandardType* subclass (with instances *boolean*, *char*, *integer*, *real*), *UserDefinedType* subclass (with subclasses *EnumerateType* and *SubRangeType*). *StructuredType* class has four subclasses: *ArrayType*, *FileType*, *RecordType* and *SetType* subclass.

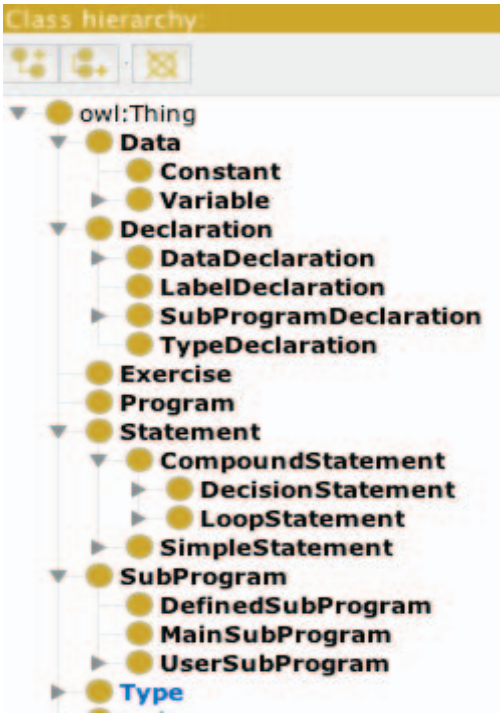

Fig. 2. Part of the class hierarchy.

D. Properties of classes with their facets

This step consists in identifying data properties for each class as defined previously and the object properties between the individual members of such classes. Thus, for each property, we define its domain (class it describes) and its types of values. . Fig. *3* shows part of the object properties obtained such as:

- *concernData* to link data (variable or constant) to statements. concernData property has two subproperties for constant (*concernConstant*) and variable (*concernVariable*). *ConcernVariable* subproperties help assign an expression to a variable (*assignVariable* property), declare a variable (*declareVariable* property), assign a value to a for loop control variable (*hasControlVariable* property);
- *hasDeclaration* property to link a *Program* to a Declaration block. *HasDeclaration* has four subProperties. Each subproperty is for a type of declaration: *hasDataDeclaration*, *hasLabelDeclaration*, *hasSubProgramDeclaration*, *hasTypeDeclaration*;
- *hasParameter* and its inverse *isParameterOf* to assign a parameter to UserSubprogram instance. The *has* properties generally have corresponding inverse *is…of* properties to represent bi-directional relationships. For instance, we have *hasType* and its inverse *isTypeOf*, *hasParameter* and *isParameterOf*, *hasSolution* and *isSolutionOf*;
- *hasSolution* to link an exercise with its solution (Pascal source code);
- *hasStatement* to link a block (*MainSubProgram, a UserSubProgram* or a *CompoundStatement)* to statements. It means block contains statements;
- *hasSubProgram* property to link a program to *MainSubProgram* or *UserSubProgram*. *HasSubProgram* has two subproperties: *hasMainSubProgram* and *hasUserSubProgram*;
- *read* property to link a *ReadStatement* class to *Variable* class to represent reading process;
- *use* property to link a *Program* class to *Unit* class like *Crt*.

Fig. 3. Part of the object properties

Figure 4 includes data properties such as:

- *assignDataValue* property to link a data (variable or constant) to a value. *AssignData*Value has many sub-properties to assign initial (*assignInitialValue*) and final (*assignFinalValue*) value to index variable of loop *for*; to assign value (*assignValue*) for assignment statement. *AssignConstant* property is used during constant declaration to assign a value to a constant.
- *difficulty* data property of exercise with instances *easy*, *medium* or *hard*;
- *direction (to or downto)* of loop *for;*
- *name* data property of a program to assign a name to a source code;
- *passBy* data property to give the parameter call type. By default Pascal uses call by value to pass parameter. Pascal Keyword *var* is used to pass parameter by reference.
- *Text* for the exercise description*;*
- *title* for the exercise title*;*
- *write* property to assign an *expression* to a write statement.

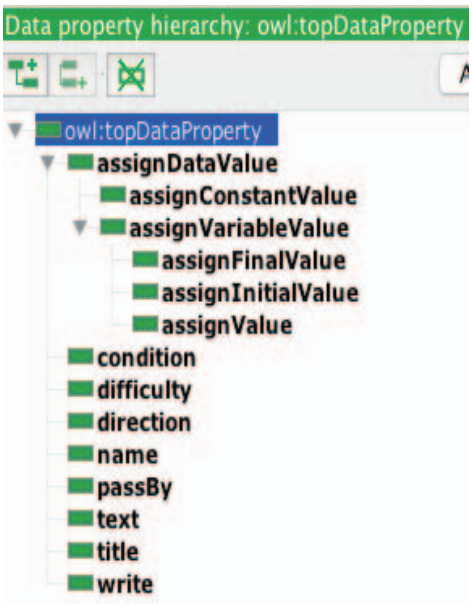

Fig. 4. Part of the data properties

E. Instances

The last step consists in creating individual instances of classes in a hierarchy with their properties. Thus, we have created many instances of exercises with their complete solutions using protégé. For instance, Fig. ***5*** shows *programFact* individual with three variable declarations, an inclusion of *crt* unit and a main function. Each of this may have, in turn, property assertions. Fig. ***6*** shows *MainSubProgram* individual with five statements. Individual *programFact* also has a data property *name* with value *factorial*.

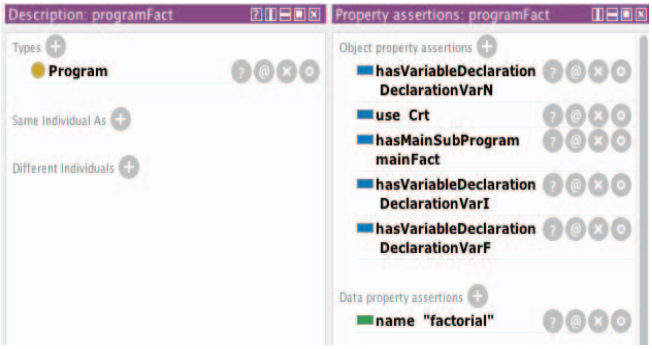

Fig. 5. View of individual pseudocode assertions

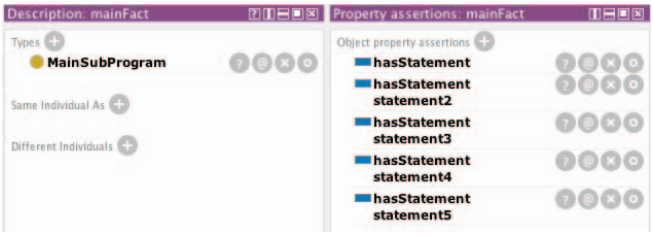

Fig. 6. View of individual mainSubprogram assertions

IV. Implementation

At the end of PasOnto ontology design, we developed a mobile and web interface to illustrate how to exploit it through an information querying module. To build web

application for storing and querying RDF data, we used Owlready [18] module with Python programming langage. Owlready helps to load and manipulate owl ontologies as Python objects. It also helps to perform SPARQL [19] query through integrated HermiT reasoner 0. Bootstrap[4] with its "Mobile First" approach and JQuery [5] are also used to facilitate the development of our responsive interface for mobile and desktop devices.

Our information querying module helps to query our program data using directly SPARQL (SPARQL Protocol And RDF Query Language). It also provides a user-friendly interface to retrieve available Pascal program and to highlight Pascal language keywords in displayed solution.

To help learners to browse the exercise database we categorize exercises by difficulty level (easy, medium and hard). Difficulty level is automatically generated from exercise solution information. Exercise solutions just containing simple variables (integer, real, boolean, char, …) and/or simple statements (write, read, assignment, if, case, … statements) are categorized easily. Medium exercises has solutions containing structured variables (array, set, files, …) pointers, and/or conditional statements (for, while and repeat), fonctions and procedures. To be declared hard, an exercise has solution containing for example multidimensional arrays, array of pointers and/or imbricated loop statements.

We also automatically generated from exercise solutions some syntactic prerequisites (how to declare an integer, an array, how to use loop statements, …?). Thus, learners know what part of Pascal courses they need to learn before writing down the solution.

Figure 7 shows the list of Pascal exercises with their title, description, generated difficulty level and the possibilities to display solutions and prerequisites. Factorial exercise has medium difficulty level due to loop statement used in solution. Other exercises are easy to solve because corresponding solutions contain read, write, conditional statements and simple variables. For instance, *Hello word* exercise only uses *write* statement without any variable.

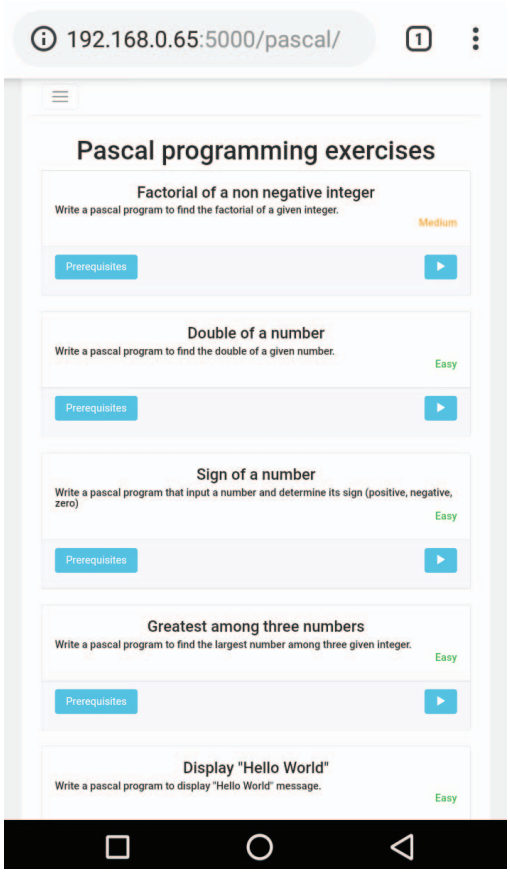

Fig. 7. List of Pascal exercises

The solution (see Fig. ***8***), highlighted with Prism[6] and corresponding to *factorial of a non negative number,* uses *crt Unit*, declares three variables and has a main block with *write*, *read*, *for* and *assignment* statements. Prism is a lightweight, extensible syntax highlighter, built with modern web standards.

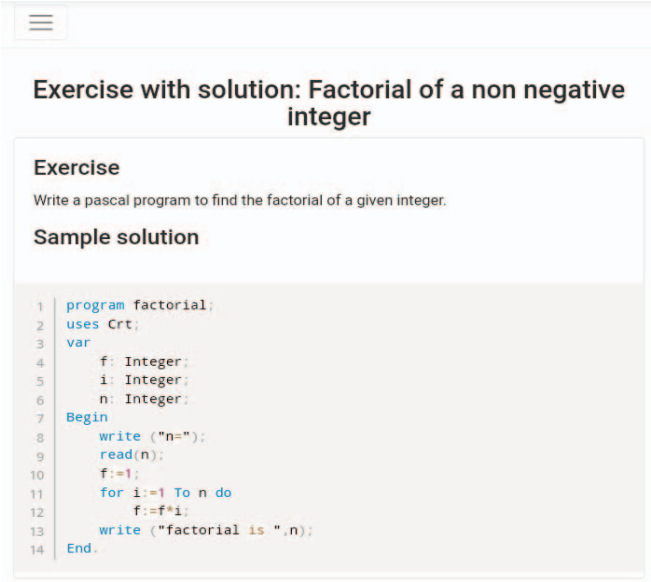

Fig. 8. Solution of exercise "Factorial of a non negative number"

[4] https://getbootstrap.com
[5] https://jquery.com
[6] https://prismjs.com

Figure 9 and 10 show prerequisites respectively for *Fatorial* and *Hello world* exercises. Thus, before writing *Factorial* exercise solution learners need to know how to declare, assign, read and write simple variables and how to use loop control statements. Each of these prerequisites points learners to part of Pascal programming course that explain how and why a particular statement is used.

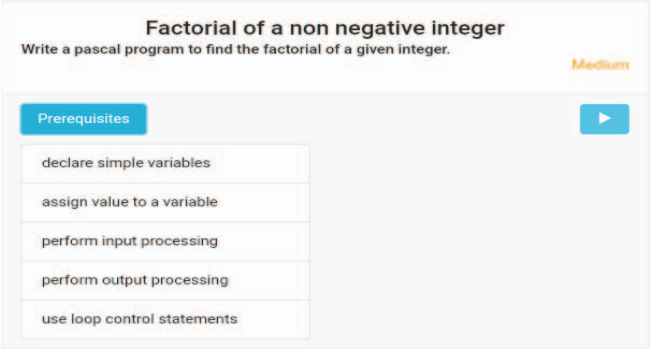

Fig. 9. Factorial exercise prerequisites

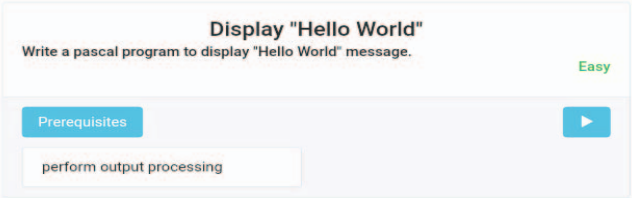

Fig. 10. Hello word exercise prerequisites

V. CONCLUSION

The best way to improve programming skills is through practice. In that regard, we give in our programming first-year course the opportunity to practice the Pascal programming language concepts by providing exercises with solutions. Nevertheless, proposed exercises and solutions are not semantically structured to be queried efficiently. For this reason, we propose in this paper, to construct an ontology to structure and describe source code in Pascal language. We also propose to develop a web and mobile interface around this ontology so as to query and highlight Pascal language keywords in displayed solutions. To help learners to know what exercises are suitable for them, we automatically generate for each exercice its difficulty level from solution. We also generate exercise prerequisites from exercise solutions to link each exercise with needed Pascal programming course chapters to learn before writing down solution.

This approach blazes the trail for further exploration. Based on the current approach, our future work seeks to study how to automatically populate our ontology from textual Pascal programs at first and more generally from procedural source code. We also plan to deeply analyse exercise description and exercise solution to generate more information like algorithmic prerequisites, exercise objectives for example in order to facilitate solution writing process.

ACKNOWLEDGMENT

We would like to thank CEA-MITIC for partially supporting this work.

REFERENCES

[1] Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing?. *International journal of human-computer studies*, *43*(5-6), 907-928.

[2] Su, X., Zhu, G., Liu, X., & Yuan, W. (2005, November). Presentation of programming domain knowledge with ontology. In *Semantics, Knowledge and Grid, 2005. SKG'05. First International Conference on* (pp. 131-131). IEEE.

[3] Ivanova, T. (2014). Bilingual Ontologies for Teaching Programming in Java. In *R. Romansky (Ed.), Proceedings of the International Conference on Information Technologies* (pp. 182-195).

[4] Kouneli, A., Solomou, G., Pierrakeas, C., & Kameas, A. (2012, September). Modeling the knowledge domain of the Java programming language as an ontology. In *International Conference on Web-Based Learning* (pp. 152-159). Springer, Berlin, Heidelberg.

[5] Pierrakeas, C., Solomou, G., & Kameas, A. (2012, October). An ontology-based approach in learning programming languages. In *Informatics (PCI), 2012 16th Panhellenic Conference on* (pp. 393-398). IEEE.

[6] Sosnovsky, S., & Gavrilova, T. (2006). Development of educational ontology for C-programming. International Journal on Information Theories & Applications, 13(4), 303-308.

[7] Noy, N. F., & McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics.

[8] Alnusair, A., & Zhao, T. (2010, August). Component search and reuse: An ontology-based approach. In *Information Reuse and Integration (IRI), 2010 IEEE International Conference on* (pp. 258-261). IEEE.

[9] Atzeni, M., & Atzori, M. (2017, October). CodeOntology: RDF-ization of Source Code. In *International Semantic Web Conference* (pp. 20-28). Springer, Cham.

[10] Ganapathi, G., Lourdusamy, R., & Rajaram, V. (2011). Towards ontology development for teaching programming language. In*World Congress on Engineering*.

[11] Epure, C., & Iftene, A. (2016). Semantic Analysis of Source Code in Object Oriented Programming. A Case Study for C#. *Romanian Journal of Human-Computer Interaction*, *9*(2), 103.

[12] Bachimont, B., Isaac, A., & Troncy, R. (2002, October). Semantic commitment for designing ontologies: a proposal. In *International Conference on Knowledge Engineering and Knowledge Management* (pp. 114-121). Springer, Berlin, Heidelberg.

[13] Fernández-López, M., Gómez-Pérez, A., & Juristo, N. (1997). Methontology: from ontological art towards ontological engineering. In: Proceedings of AAAI97 spring symposium series, workshop on ontological engineering, Stanford, CA, pp 33-40.

[14] Uschold, M., & King, M. (1995). Towards a methodology for building ontologies. Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing held in conjunction with IJCAI, Montreal.

[15] Graudina, V., & Grundspenkis, J. (2011, June). Algorithm of concept map transformation to ontology for usage in intelligent knowledge assessment system. In *Proceedings of the 12th International Conference on Computer Systems and Technologies* (pp. 109-114). ACM.

[16] Yao, J., & Gu, M. (2013). Conceptology: Using Concept Map for Knowledge Representation and Ontology Construction. *JNW*, *8*(8), 1708-1712.

[17] Motik, B., Patel-Schneider, P. F., Parsia, B., Bock, C., Fokoue, A., Haase, P., ... & Smith, M. (2009). OWL 2 web ontology language: Structural specification and functional-style syntax. *W3C recommendation*, *27*(65), 159.

[18] Lamy, J. B. (2017). Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*, *80*, 11-28.

[19] Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (January 2008), http://www.w3.org/TR/rdf-sparql-query/

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, *53*(3), 245-269.