Auto-generative Learning Objects in Online Assessment of Data Structures Disciplines

Ciprian-Bogdan Chirila
University Politehnica Timişoara,
Piaţa Victoriei 2, Timişoara 300006 Romania
chirila@cs.upt.ro

Abstract

Nowadays, regional IT industry lacks human resources because of the pressure created on the labor market by the high-value economic projects. Tutors tend to be more and more loaded with teaching, research, and administrative tasks. Students tend to use more and more electronically devices like laptops, tablets, and mobile phones in their learning sessions. In this context, universities should rely more on technologies like: LMSs (Learning Management Systems), MOOCs (Massive Open Online Courses), and why not GLOs (Generative Learning Objects) or evenAGLOs (Auto-generative Learning Objects). Auto-generative learning objects are reusable pedagogical patterns to be instantiated with generated content based on random numbers to fulfill the learning objectives. Many online e-learning resources are available containing interactive presentations, gamifications of several learning objectives. Such e-learning resources are hard to reuse and even harder to modify and adapt to; each discipline needs this because it needs access to the source code, programming knowledge to change, test and deploy etc. In this paper, we will focus on computer science disciplines needed in the regional IT industry, namely data structures and algorithms. We will show how a tutor can build several auto-generative learning objects in order to assess the knowledge of a class of students. We will start with the design of the generic models, then we will assess the generated content created with the help of a tool based on meta-programming, afterwards, we will deploy the content to a webserver to be consumed by the students. Finally, we will evaluate the assessed results and discuss the approach both from the student's and the tutor's perspective.

Keywords: generative learning objects, auto-generative learning objects, online assessment

1. Introduction

The nowadays IT industry is in a human resources crisis due to the pressure created on the labor market by the high economic value projects. The teaching staff from universities tends to be more and more loaded with teaching, research, and administrative tasks.

Moreover, students rely very much on technology using laptops, tablets, and smartphones in their daily routine and also when they study. They are much present on social networks, accessing social media content.

Such a context forcesuniversities into relying on e-learning technologies such as microformats enhanced learning management systems (LMS) (Ermalai & Dragulescu & Ternauciuc & Vasiu, 2013), massive open online courses (MOOCs) (Naaji & Mustea & Holotescu & Herman, 2015), generative learning objects (GLOs) (Boyle, 2003; Boyle, 2006), and also auto-generative learning objects (AGLOs) (Chirila & Ciocarlie & Stoicu, 2015).

In this paper, we will present a concrete instantiation of a set of auto-generative learning objects that help students grasp easier basic programming knowledge that is taught in data structures and algorithms discipline. We worked on a set of data structures namely: generalized trees, binary ordered trees, graphs, and weighted graphs.

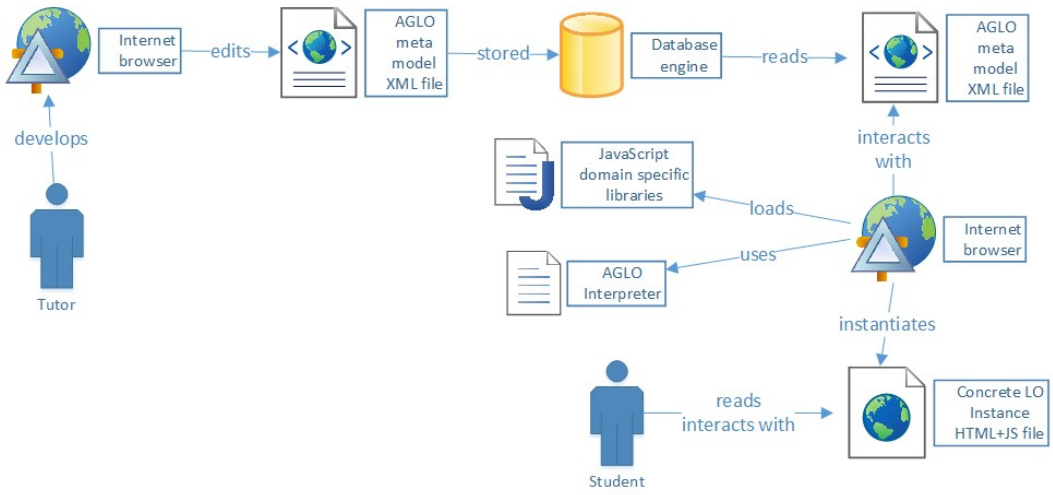

Figure 1. The AGLO online assessment approach

In Figure 1, we present the lifetime of AGLOs in the context of online student assessment following a set of steps. In the backend, the tutor develops an AGLO model respecting a predefined meta-model. The model is intuitive, it has a few sections where symbols are defined using formulas and random numbers and then used in a section of a presentation for the student. When such models are created they are stored in a storage facility like a database to be selected by the student through the web application frontend. In the frontend, the students access the web application using a web browser from a workstation, tablet or smartphone. In the assessment process, the student will access several AGLOs. At this step, the accessed AGLOs are instantiated with random numbers, formulas are evaluated to fulfill the designed learning or testing scenario and to create the presentation content for the student. Nevertheless, the instantiated symbols will be used for the automatic assessment of the answers correctness.

In order to perform complex tests in a certain domain, a library design is proposed for two purposes. The first purpose is to generate randomly complex structures controlled by parameters, while the second purpose is to implement operators to exploit the details of these structures. For example, in the field of computer programming, we designed several abstract data types together with random generation algorithms and also the classic operators to able the exploit the created structures. More specifically, we designed anabstract-data-type tree equipped with random generation algorithms controlled by parameters like a number of nodes, tree degree, key types etc.

The instantiated object is presented to the student. The student will have to read the sentence and start buildinghis/hers answer. The student will fill in the answeron the web page;his/hers response will be evaluated and stored in a learning record store (LRS). The correct answer is shown to the student along with the feedback when possible. In order to make the connection with social media, the positive consolidated results are published on the walls of social networks like the most popular nowadays: Facebook.

From the cognitive point of view, this paper relies on the learning repetition principle of the same concept in different contexts. Aristotle said that it is possible for repetition to create a natural tendency. The test model that we propose is based on a repeated activity intended to build strength and precision and to enable a higher level of performance for the student.

The paper is structured as follows. Section 2 presents related works. Section 3 presents the AGLO formal model, in a nutshell. Section 4 presents the design and implementation of 10 AGLOs used in online evaluation with students. Section 5 discusses the main encountered issues. Section 6 concludes and sets the perspective.

2. Related works

In this section, we will present related works in the area of learning objects.

Learning objects are considered deliverable electronic components having a specific learning objective. In this sense, several models were standardized like LOM (IEEE Learning Technology Standards Committee, 2000).

In Boyle (2003), Boyle (2006), Jones (2007), the GLO term was coined and considered to be the second generation of learning objects. The reuse of pedagogical template patterns was inspired bythe reused concept of software engineering as implemented with object-oriented programming.

In Stuikys, Burbaite, and Damasevicius (2013) is presented a generative model for teaching computer science disciplines using Lego robots. Their work is similar to ours, having common elements like presentation, parameters, generation by metaprogramming, etc. While their work uses embedded code loaded into robots, we implement a web-oriented approach which handles the complexity of exercises through domain libraries. In Damasevicius & Stuikys (2009), they present a learning object model organized and sequenced by feature diagrams and generated by metaprogramming. Our work uses a manually crafted competence model for sequencing.

The works of Bogdan (2016) and Bogdan and Ancusa (2016) present principles for designing e-learning tools dedicated to the local automotive industry that we target with our AGLO model. Their work implies mostly static content having dynamic presentation enhanced by modern tools like TT Knowledge Force. They use a based-development template for the manual creation of learning materials while our approach uses web-oriented metaprogramming. The common aspect of their and our work is that both are oriented towards the automotive industry where basic programming skills are required in writing embedded codes.

Karpova, Shmelev, and Dukhanov (2016) present a similar model to our AGLO approach controlled by parameters but enhanced with dynamic learning and evaluation functionalities. The model was applied to master disciplines dealing with optimization and simulation.

In Chirila (2013) and Chirila (2014) are presented several web-oriented generative models for primary and middle school students who learn basic mathematical concepts. The proposed models include dialogs implemented using automate states and mathematical formulae, generic games, etc. implemented using JavaScript programming language.

3. Auto-generative learning objects

In this section, we will present the structure of AGLOs in the context of our approach.

The AGLO meta-model is structured in XML as in *Figure 2*,a refinement from Chirila, Ciocarlie, and Stoicu (2015). The AGLO definition contains several sections like name, scenario, theory, question, answers, and feedback (line 01).

The name element contains the name of the AGLO, possibly a small description in the human language (line 02).

The section of the scenario (line 03) contains a comment (line 04) followed by a set of symbol definitions. The comment should describe the imagined scenario in details and it has the same role as code comments. The symbol is the central element of the AGLO model. The symbol has a name and is very similar to programming language variables. Symbols may be called also parameters since they control the content of the AGLO content in the process of instantiation.

The symbol XML element has attached a typeattribute which is useful for the AGLO designer, only. The symbol type may be a primitive type like integer, character, string, array, but also another user defined types like tree, graph etc.

```
01 AGLODef ::= <action> Name Scenario [Theory]
Question Answers Feedbacks </action>
02 Name ::= <name> (ID)* </name>
03 Scenario ::= <scenario> [ Comment ] Symbol* </scenario>
04 Comment ::= (ID|CT)*
05 Symbol ::= <symbol SymbolName Type> Expression </symbol>
06 SymbolName ::= name = ID
07 Type ::= type =
 (boolean | int | float | double | string | array)
08 Expression ::=
<expr> Function ( ExpressionList ) </expr>
09 ExpressionList ::= Expression (, Expression)*
10 Function ::=
(item from composed functions and operators list of JavaScript
 using random numbers)
11 Theory ::= <theory> (ID)* </theory>
12 Question ::= <question> (ID | Value)* </question>
13 Value ::= <value name = ID />
14 Answers ::= <answers> (Answer)+ </answers>
15 Answer ::= <answer id = Index >
(ID | Value)* Correctness </answer>
16 Index ::= INTEGER_LITERAL
17 Correctness ::= <correct> (true | false) </correct>
18 Feedbacks ::= <feedbacks> (Feedback)+ </feedbacks>
19 Feedback ::= <feedback> AnswerIdList (ID | Value)* Active
</feedback>
20 AnswerIdList ::= <AnswerIdList> (INTEGER_LITERAL)+
</AnswerIdList>
21 Active ::= <active> (true | false) </active>
```

Figure 2. Auto-generative Learning Object Model Definition

The symbol has also an initialization expression which relies on ECMA JavaScript (ECMA International, 2011) functions, operators, and previously defined symbols (lines 08-10). In this point, the AGLO designer should use random number generators in order to change the values at each instantiation. Of course, that functionsand operators can be composed altogether and also with domain-specific libraries in order to create a complex content.

The symbols type is dynamic since all the initialization expressions are evaluated dynamically at run time and because JavaScript programming language is dynamically typed.

Line 11 defines a theory section which is designed to be static and to present theoretical aspects of the problem which the student has to solve in the context of the AGLO. The content is filled by the AGLO designer with HTML content.

Line 12 defines the section of the questions where the problem is presented to the student. This section, benefits from the symbols defined in the scenario section. In this section, symbol values are catenated with static text to form the sentence to be presented to the student. Each time the AGLO is instantiated, the student will receive a different content based on the variability induced by the symbols depending on random numbers. This section creates a variable linguistics model based on templates including words and also figures.

In order to refer a value of a symbol, we use the XML element <*value*> (line 13) together with the name attribute. Such a technique is very common in PHP, JSP, ASP web pages where static content is mixed with dynamic content. The difference is that in such contexts usually, they do not use the randomly generated dynamic content.

The answers section (lines 14-17) contains one or more declared answers which may include dynamic content based on symbol values. Answers can be open and single choice or

multiple choices, some being true and some being false. Depending on the number of answers the AGLO interpreter will handle each case separately.

Lines 18-21 define the syntax of feedbacks which accompanies the answers.

4. The design and implementation of the online assessment AGLOs

In this section, we will present the specification, design, and implementation of the learning objects in order to be used in the automatic online assessment.

We will focus on a set of 10 tests from the area of trees and graphs used inlaboratory evaluation of the student and we will show how they can be implemented with AGLOs, thus benefitting from content openness, automatic instantiation, and assessment.

In the first test, we will give the student an array of parent indexes and will ask him to draw the tree. Our AGLO model will contain the following:

1. a random symbol for the number of nodes;
2. a generalized tree created randomly;
3. the parent index representation of the tree;
4. an online diagram editor for the student to represent graphically the tree;
5. a comparison operator to compare the student-drawn tree with the generated one.

In the second test, the student should write the indexes for the first child and the right sibling of each node of a tree. The AGLO model should contain the following:

1. a random symbol for the number of nodes;
2. a generalized tree created at random;
3. the first child and right sibling representation of the tree.

In the third test, we will give the student an array of integers randomly generated and representing a balanced binary tree in the sense that the number of nodes in the left and right subtrees differs by a few units. The student will have to represent graphically the tree. Our AGLO model will contain the following:

1. a random symbol for the number of nodes;
2. an online diagram editor for the student to represent graphically the binary tree;
3. a comparison operator to compare the drawn-tree with the generated one.

In the fourth test, the student will have to represent another two trees obtained by deleting sequentially the roots two times.

1. a random symbol for the number of nodes;
2. a balanced binary tree created randomly by the number of nodes;
3. a node deletion operator in order to be applied on the root;
4. an online diagram editor for the student to represent graphically the binary tree;
5. a comparison operator to compare the drawn tree with the generated one.

In the fifth test, we will present to the student an adjacency matrix with a set of nodes denoting a graph. The student will have to represent graphically the graph. Our AGLO model for this test will contain the following:

1. a random symbol for the number of nodes;
2. a generated adjacency matrix of 0 and 1 with the first diagonal set to 0;
3. an online diagram editor for the student to draw the graph.
4. a comparison operator to compare the drawn-graph with the generated one.

In the sixth test, we will present to the student a graph and we will ask him to write the graphs' degree. The AGLO model will contain the following:

1. a random symbol for the number of nodes;
2. a generated adjacency matrix of 0 and 1 with the first diagonal set to 0;
3. a graph representation operator in SVG format;
4. a graph operator to compute its degree.

In the seventh test, we will present to the student a graph and we will ask him to list the nodes in a depth-first search order. The AGLO model will contain the followings:

1. a random symbol for the number of nodes;
2. a generated adjacency matrix of 0 and 1 with the first diagonal set to 0;
3. a graph representation operator in SVG format;
4. a graph operator to compute the depth first search order of nodes.

In this case, special indications are needed for the student because there are multiple correct node orders. In order to be able to automatically and easily assess the student's answer, we impose two restrictions: first, to set the starting node and second, to choose the adjacent nodes in a certain order. For example, we can choose the starting node to be the first in a lexicographical order and to choose alphabetically the adjacent nodes.

In the eighth test, we will present to the student a graph and we will ask him to list the contents of the queue sequentially to illustrate the breadth first search. The AGLO model will contain the followings:

1. a random symbol for the number of nodes;
2. a generated adjacency matrix of 0 and 1 with the first diagonal set to 0;
3. a graph representation operator in SVG format;
4. a graph operator to compute the breadth first search order of nodes.

For the last two tests, we will present in detail the variables and their initialization expressions (see *Figure 3*) from the AGLO model.

```
<scenario>
<symbol name="n" type="integer">random(5,9,0);</symbol>
<symbol name="g" type="graph">random_graph(
      {"nNoOfNodes":v("n"), "nMin":0, "nMax":1});</symbol>
<symbol name="gSVG" type="string">v("g").toSVG()</symbol>
<symbol name="gdfs" type="string">
      v("g").depthFirstSearch()</symbol>
</scenario>
```

Figure 3. Symbols definition for a graph-based test

In this scenario, we intend to generate a random graph and compute a deep first-search node list. The first defined random symbol is *n*, namely the number of nodes in the graph as an integer from 5 to 9. The next symbol is named *g* and denotes the graph object created randomly using 3 parameters: the number of nodes, the minimum, and the maximum value for the weight. For the number of nodes, we used the previously computed value of *n*, whereas for the weights, we used two constants 0 and 1 since the graph is not weighted.

The next symbol is named *gSVC* and is of type string. The semantic for this symbol is to represent the graph as a string using the SVG (Scalable Vector Graphics) representation. SVG is a markup language, like HTML (Hyper Text Markup Language), used to represent vector graphics based on graphical primitives. We initialized the symbol with the *toSVG()* graph operator from the JavaScript domain library.

In *Figure 4,* we present the result of the model instantiation. The number of nodes was generated randomly at value 7. The graph was generated with 9 edges and with random node names like [C, E, G, H, I, J, L]. The nodes are generated randomly, in alphabetical order. The SVG representation of the graph is visible inside *Figure 4*. The representation is generated putting the nodes on an imaginary circle with a radius computed as such to provide good visibility and then edges are drawn.

For the student answer, in this test two restrictions were imposed:

1. the starting node has to be selected alphabetically, namely, the first in the ordered list;
2. the adjacent nodes when it is the case will be selected also in alphabetical order.

Thus, applying these restrictions the computed solution is C, E, G, J, L, H, I and is unique. Node C is the starting node since it is the first from the lexicographical point of view. The first step CE is the only choice coping with the restrictions from the [CE, CG, and CJ] edges. Next, EG is the first edge in the list of [EG, EJ]. The next step is GJ which is the only choice. Edge JL is another unique choice. Edge LH is the next step from the list [LH, LI]. Finally, the last edge is obtained by backtracking to node *L* and then taking edge *LI*.
These restrictions allow us to drive the student to build only one solution from the possible set of solutions. This will determine an easier way of comparing the student's answer with the answer of the computer. Another more general solution is to use validation functions which require implementation in domain libraries written in JavaScript.

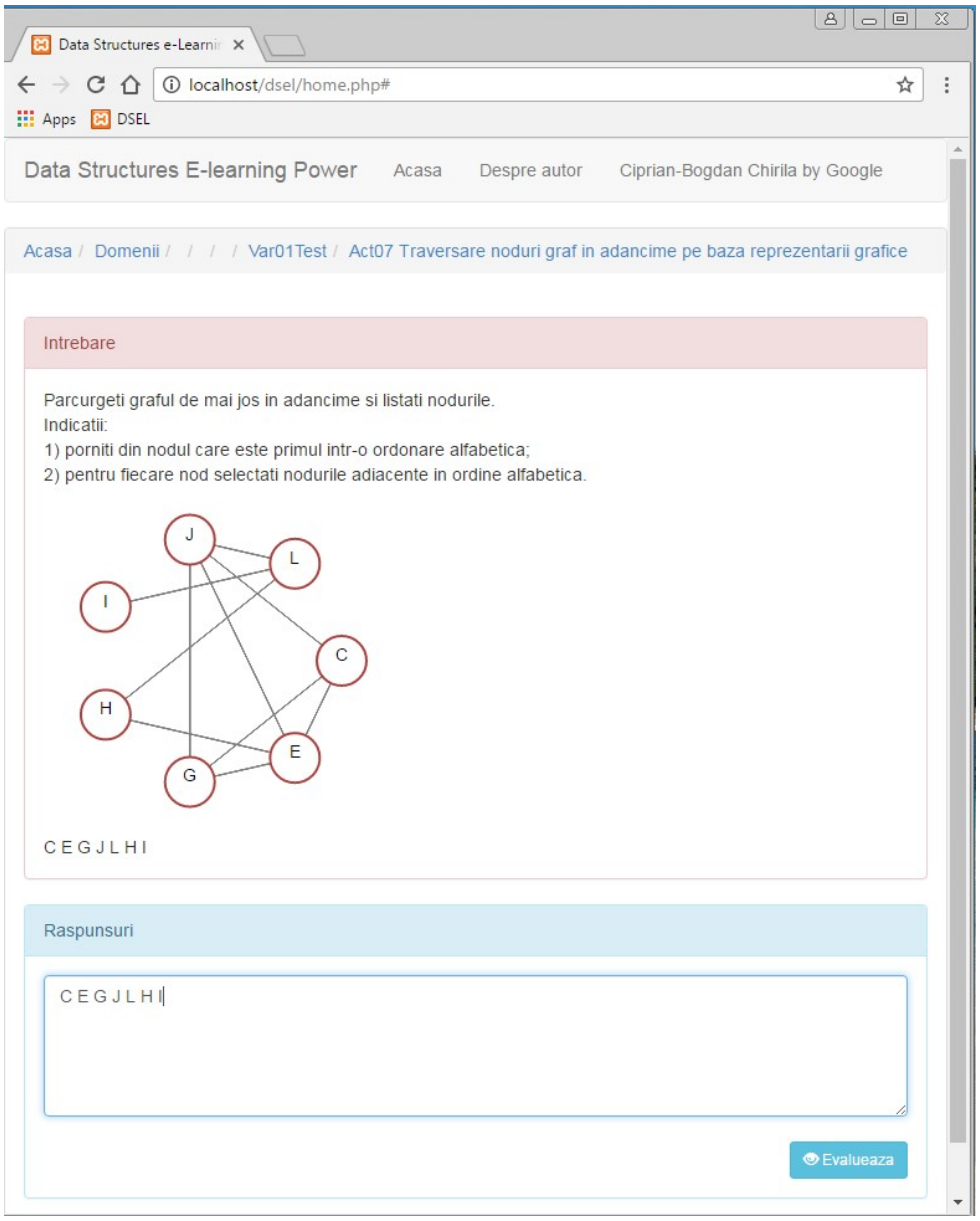

Figure 4. Online test assessment example

In the ninth test, we will present to the student a weighted graph and we will ask him to write on each line the content of the visited and unvisited sets of nodes as the Prim algorithm drives it to build the minimum spanning tree. The AGLO model will contain the following:

1. a random symbol for the number of nodes;
2. a generated adjacency matrix of random weights with the first diagonal set to 0;
3. a graph representation operator in SVG format;
4. a graph operator to compute the steps of the Prim's algorithm.

In the tenth test, we will present to the student a weighted graph and we will ask him to write on one line the ordered edges and then on the second line the edges subset forming the minimum spanning tree, added in order without creating a cycle. The AGLO model will contain the following elements:

1. a random symbol for the number of nodes;
2. a generated adjacency matrix of random weights with the first diagonal set to 0;
3. a graph representation operator in SVG format;
4. a graph operator to compute the steps of the Krushkal's algorithm.

5. Discussion

In this section, we will discuss the complexity based on the number of symbols used in the design of the AGLO test battery. We will analyze three aspects:

1. symbols count;
2. symbols percentages;
3. parameters count for the creation of structures.

Table 1. Symbols count

Test number	Symbols initialized with functions computing random scalar values	Symbols initialized with functions create random structures	Symbols initialized with ADT operators	Total number of symbols
1	2	1	3	6
2	0	2	5	7
3	3	1	2	6
4	3	2	1	6
5	1	1	2	4
6	1	1	2	4
7	1	1	2	4
8	1	1	2	4
9	1	1	2	4
10	1	1	2	4
Total	14	12	23	49

In table 1, we can see the number of symbols used in the design of the 10 tests. The symbols were distributed in three categories:

1. symbols that were initialized with functions that compute random scalars used in the instantiation process (14), e.g. integers;
2. symbols that were initialized with functions or operators that create random structures to exploited later by abstract data type (ADT) operators (12);
3. symbols that were initialized with ADT operators for computing domain specific values used in the presentation or assessment part (23).

Table 2. Symbols percentage

Test number	Symbols initialized with functions computing random scalar values %	Symbols initialized with functions creating random structures %	Symbols initialized with ADT operators %
1	33.33%	16.67%	50.00%
2	0.00%	28.57%	71.43%
3	50.00%	16.67%	33.33%
4	50.00%	33.33%	16.67%
5	25.00%	25.00%	50.00%
6	25.00%	25.00%	50.00%
7	25.00%	25.00%	50.00%
8	25.00%	25.00%	50.00%
9	25.00%	25.00%	50.00%
10	25.00%	25.00%	50.00%
Total	28.33%	24.52%	47.14%

The total number of symbols used is 49, so we used in average almost 5 symbols for each test. From this measure, we can draw the conclusion that AGLO models complexity for this battery of tests is kept low.

In table 2, we look at the percentages of the three categories of symbols. On average, the first and second categories of symbols are around one-fourth each of the totals, while the third category is around half of the total. Briefly, half of the symbols manage the random creation and the other half manages the classic or standard ADT operators.

In table 3, we look at the number of parameters used for the instantiation of the AGLO models from the test battery. On average, for a generated structure there are necessary 3 parameters.

Table 3. Parameters count

Test number	Number of parameters used in structures creation	Average number of parameters on functions creating random structures
1	2	2
2	3	1.5
3	4	4
4	5	2.5
5	3	3
6	3	3
7	3	3
8	3	3
9	3	3
10	3	3
Average	3.2	2.8

6. Conclusions and perspectives

In this section, we conclude and set the perspectives. The conclusion will consist in several advantages and drawbacks.

Students benefit from the AGLO models since the content is dynamic, having a large set of generated contexts to experiment with. AGLO tests are reusable at a higher level since they are unlike static tests which have the same answers anytime the student answers them. Another advantage of the approach is that the student may test individually its understanding of algorithms by re-instantiation until it is well-learned. We have no solutions yet in evaluating code implementations written in some programming language, except some black box techniques, where the e-learning environment has no control over the details and cannot offer support.

For tutors, one advantage of the AGLO enabled dynamic content approach over static content e-learning tools is that the content is structured and thus modifiable and adaptable at two levels. At the first level, the tutor will edit the AGLO model where the presentation layer of the test is defined. At this level, the tutor can make several adaptations, e.g.: to change the language of the test while keeping the same business logic, to adjust the topic of the test sentences for better understanding or for increased difficulty, to adjust the computed answer in the same business logic but still relying on the predefined domain libraries.

One major drawback for the GLO designer is that the model is dependent on the domain specific libraries and when he needs a specific functionality he needs to ask a programmer to write it. In the area of computer science, probably, will not be the case, like in other domains.

As future work, we intend to multiply the first category of variables and to introduce levels of difficulty, and also, adaptiveness. One possible choice is to use different initialization expressions producing values in different ranges which will increase the size of the created structures and thus, the difficulty level. Using artificial intelligence algorithms, the student's answers are evaluated and easier or harder tests are generated in order to fulfill the goal of the learning objective. Another important aspect is the implementation of contextual feedback when a wrong answer occurs. Some variable based models experimented but we consider them still young. The prototype was only piloted in the laboratory environment so we intend to test it on a larger pool of students.

References

Bogdan, R. (2016), Guidelines for developing educational environments in the automotive industry, 1st International Conference on Smart Learning Ecosystems and Regional Developments, Timisoara, Romania.

Bogdan, R.; Ancusa, V. (2016), Developing e-learning solutions in the automotive industry. *World Journal on Educational Technology*, *8*(2), 139-146. doi:10.18844/wjet.v8i2.823

Boyle, T. (2003), Design principles for authoring dynamic, reusable learning objects, *Australian Journal of Education Technology,volume 19*(1), pp. 46-58.

Boyle, T. (2006), The design and development of second generation learning objects, Invited talk at Ed Media 2006, World Conference on Educational Multimedia, Hypermedia and Telecommunications, Orlando, Florida, June 28.

Burbaite, R.; Bespalova, K.; Damasevicius, R.; Stuikys, V. (2014), Context Aware Generative Learning Objects for Teaching Computer Science, *International Journal of Engineering Education, volume 30*(4), pp. 929-936.

Chirila, C.B. (2013), A Dialog Based Game Component for a Competencies Based E-Learning Framework, *In proceedings of SACI 2013 8-th IEEE International Symposium on Applied Computational Intelligence and Informatics*, pp. 055-060, Timisoara, Romania, May.

Chirila, C.B. (2014), Educational Resources as Web Game Frameworks for Primary and Middle School Students, In proceedings of eLSE 2014 International Scientific Conference eLearning and Software Education, Bucharest, Romania, April.

Chirila, C.B.; Ciocarlie, H.; Stoicu-Tivadar, L. (2015), Generative Learning Objects Instantiated with Random Numbers Based Expressions, *BRAIN - Broad Research in Artificial Intelligence and Neuroscience, vol. 6*,no. 1-2, Bacau, Romania, October.

Damasevicius, R.; Stuikys, V. (2009), Specification and Generation of Learning Object Sequences for e-Learning Using Sequence Feature Diagrams and Metaprogramming Techniques, *In proceedings of 2009 9-th International Conference on Advanced Learning Technologies*, pp. 572-576, Riga, Latvia.

ECMA International (2011), Standard ECMA-262 ECMAScript Language Specification Edition 5.1, http://www.ecma-international.org/publications/standards/Ecma-262.htm.

Ermalai, I.; Dragulescu, B.; Ternauciuc, A.; Vasiu, R. (2013), Building a module for inserting microformats into Moodle. Adv. Electr. Comput. Eng. Jan 1;13(3):23-6.

Florea, A.; Gellert, A.; Florea, D. (2016), Teaching programming by developing games in Alice, *The International Scientific Conference eLearning and Software for Education*, Bucharest: "Carol I" National Defence University.

Han, P.; Kraemer, B.J. (2009), Generating Interactive Learning Objects from Configurable Samples, emphIn proceedings of International Conference on Mobile, *Hybrid and On-line Learning*, pp. 1-6, Cancun, Mexico, February.

IEEE Learning Technology Standards Commitee (2000), LOM working draft v4.1, http://ltsc.ieee.org/doc/wg12/LOMv4.1.htm.

Jones, R.; Boyle, T. (2007), Learning Object Patterns for Programming, *Interdisciplinary Journal of Knowledge and Learning Objects, vol. 3*.

Jung, H.; Park, C. (2012), Authoring Adaptive Hypermedia using *Ontologies International Journal of Computers Communications & Control, ISSN 1841-9836, volume 7*(2), pp. 285-301, June.

Karpova, M.; Shmelev, V.; Dukhanov, A. (2016), Information resource based on scientific software as a core of interdisciplinary learning resources, 2016 IEEE Frontiers in Education Conference (FIE), pp. 1-5, Eire, PA, USA.

Naaji A. & Mustea A. & Holotescu C. & Herman C. (2015). How to Mix the Ingredients for a Blended Course Recipe, *Journal of Broad Research In Artificial Intelligence and Neuroscience, 6*(1-2), ISSN 2067-3957, Bacau, Romania, October.

Stuikys, V.; Burbaite, R.; Damasevicius, R. (2013), Teaching of Computer Science Topics Using Meta-Programming-Based GLOs and LEGO Robots, *Journal of Informatics in Education, volume 12*(1), pp. 125-142, Institute of Mathematics and Informatics, Vilnius, Lithuania.