BRAIN. Broad Research in Artificial Intelligence and Neuroscience
Volume 1, Issue 2 , April 2010

Generative Learning Objects Instantiated with Random Numbers Based Expressions

Ciprian-Bogdan Chirila
Computers and Technology Information Department, University Politehnica Timişoara, Romania
chirila@cs.upt.ro

Horia Ciocârlie
Computers and Technology Information Department, University Politehnica Timişoara, Romania
horia@cs.upt.ro

Lăcrămioara Stoicu-Tivadar
Automation and Applied Informatics Department, University Politehnica Timişoara, Romania
lacramioara-stoicu.tivadar@aut.upt.ro

Abstract:
The development of interactive e-learning content requires special skills like programming techniques, web integration, graphic design etc. Generally, online educators do not possess such skills and their e-learning products tend to be static like presentation slides and textbooks. In this paper we propose a new interactive model of generative learning objects as a compromise between static, dull materials and dynamic, complex software e-learning materials developed by specialized teams. We find that random numbers based automatic initialization learning objects increases content diversity, interactivity thus enabling learners' engagement. The resulted learning object model is at a limited level of complexity related to special e-learning software, intuitive and capable of increasing learners' interactivity, engagement and motivation through dynamic content. The approach was applied successfully on several computer programing disciplines.
Keywords: e-learning, learning objects, generative learning objects, meta-programming, compiling techniques, function composition

1. Introduction

E-learning is a key area of research with a great influence on the developments of several industries. For example, the nowadays ITC industry is in a continuous growth because of its applications in almost all industrial domains. Companies tend to lack qualified human resources and because of that they reject high economical value projects. In response to this lack of human resource problem, universities started to develop several alternative study programs, many of them are based on e-learning technology and namely on electronic learning materials. Learning objects (LO) are considered to be digital resources that support learning and can be delivered across networks in large or small sizes [20]. In order to increase the reusability and interoperability of LOs, standards were developed (e.g. LOM [14]) by several organizations (IEEE Learning Technology Standards Committee). In the general family of LOs there is a special subset named generative learning objects (GLOs) based on instantiable pedagogical patterns. GLOs will be presented in detail in section 2. In our approach we refine the general GLO concept and we enhance it with variability and interactivity features. In this article we address the problem of static content of learning materials that is exhibited to learners and are highly used in the current approaches. Most of the resulted materials are presentation slides, textbook chapters and other static resources that are deployed in online learning management systems. The enumerated static resources have several disadvantages like:

i) the lack of content interactivity;
ii) the lack of material engagement;
iii) the lack of answer related feedback;
iv) the lack of context diversity.

In order to solve these problems we used the following approach:

i) we developed static content materials, specifically learning objects;

ii) we analyzed their variability potential;

iii) we abstracted their variability models;

iv) we created parametrization models to generate different values based on random numbers according to their learning objectives.

We propose a solution to these problems in the form of a new generative learning object model having the following features:

i) instantiation based on random number generators - the context of the instantiated learning objects is different for each learning use case;

ii) automatic instantiation of the learning objects - with the values of the already initialized parameters;

iii) automatic answer computation enabling automatic correctness assessment;

iv) automatic contextual feedback generation.

On the other hand we also propose a formalism to support the creation and development of such learning objects together with the runtime environment in order to be exploited in practice.

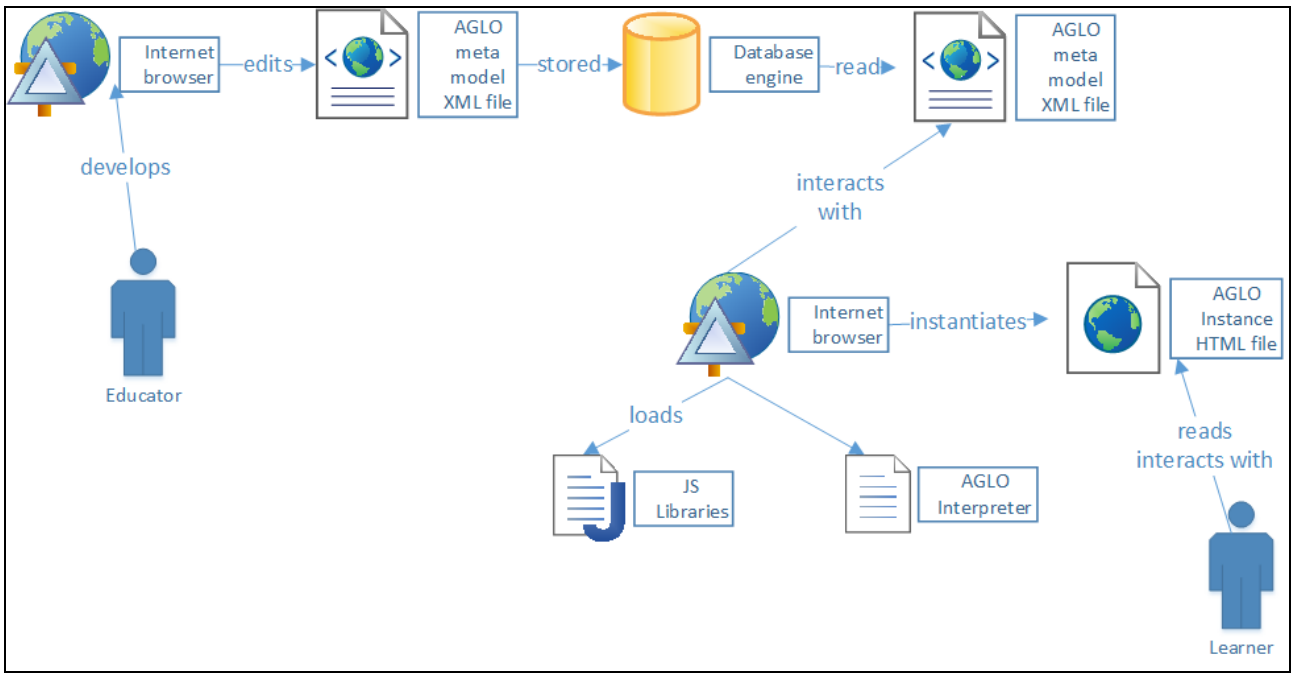

Figure 1. The AGLO Workflow

Figure 1 depicts the workflow of our LOs. Starting from left to right, on the back-end side, the educator develops the metamodel of the autogenerative learning object (AGLO) in a web browser. The editing consists in either writing XML formalisms or using specialized editing tools. The completed model is saved in storage facilities like a file system or database engine. Next, the meta-model file is requested by a learner in his learning process through the help of a web browser. At this point the web browser with the help of an interpreter and auxiliary libraries instantiates the AGLO meta-model resulting in an HTML instance exposing an interactive behavior. The approach was applied successfully on several computer science disciplines. The rest of the article is structured as follows. In section 2 we present previous research in the area of generative learning objects. Section 3 presents our model of the autogenerative learning objects. In section 4 we explain the competence hierarchy which stands as support for the autogenerative learning objects. Section 5 describes a case study of our approach applied to computer programming disciplines. In section 6

we present implementation details of the developed prototype. Section 7 presents the experimental results. Section 8 concludes and sets the perspectives.

2. Related Works

In this section we present related works on GLOs. The GLO concept was conceived by Tom Boyle and is based on software engineering principles like: modularity, cohesion, decoupling and composition [1]. A LO that is cohesive, decoupled and pedagogically rich can be re-purposed for different uses. GLOs are considered to be the second generation learning objects embedding rich pedagogical templates and being structured for reuse [2]. Regular LOs can be abstracted in patterns in order to be reused and to enable new resources to inherit the effectiveness of the originals [12]. In this sense GLOs represent design patterns implemented as reusable templates. GLOs main principle is borrowed from the object-oriented programming paradigm where the object model is described at meta-level through the class concept and afterward at run-time is instantiated into a concrete object. Thus, the reuse is not focused on content but on the design.

The second GLO approach was designed by Damasevicius and Stuikys. [16] presents a knowledge based GLO model composed out of knowledge-based interface, and a knowledge based body. The body contains a rich structure: declarative part, a procedural part, a contextual part and a managerial part. The content is based on commonality-variability analysis for model creation. They present an object for the learning of homogeneous boolean functions where the parameters variability scope determines the scope of the whole LO domain. [17] describes the GLO model through the variability dimension of the LO domain. The result of the domain analysis is expressed with the help of feature diagrams (FD). The FD representation expresses the intention of a concept using mandatory, optional and alternative features. It is considered a generic high level model for the GLO. Some design principles are explained for the creation of GLOs. [17] describes the GLO development process using FDs and generative techniques. FDs are used to represent knowledge, commonality-variability relationships between features and are considered to be high-level specification for generative reuse. [8] focuses on technological aspects of GLO development. The technologies involved in GLO creation are multiple: i) feature diagrams in order to model variability resulting feature models; ii) multilanguage design (separation of concerns) - domain languages to implement domain functionality and external languages for pre-processing, scripting to implement the model generalization; iii) metaprogramming for the manipulation of programs as data - used to implement generalization in the domain; iv) generative specifications development in the form of Open Promol metalanguage - used to describe GLO metamodels. [18] defines GLOs aggregation as internal sequencing of generative contents. [9] presents how LO sequences can be generated from models expressed with feature diagrams using metaprogramming techniques. In their approach the GLO model has the form of a script in Open Promol meta-language editable by the educator, and instantiable into HTML markups in order to be consumed by the learner. [19] presents an application of GLO models in computer science programming disciplines using LEGO robots. Their GLO model includes several modules responsible with the robot manipulation according to its learning objective. [3] introduces context-aware GLOs for computer science topics where a new GLO dimension is found, namely the one of teaching context which may have variants.

[4, 5] present generative techniques for LOs based on random numbers applied to primary and middle school arithmetic discipline. [7] presents non-interactive generative models implemented as C programs and used in computer science disciplines written exams like: data structures, programming language fundamental concepts, compiling techniques, operating systems. [6] presents potential uses of the generative concepts in biomedical informatics, namely in the pharmacology discipline.

In [11] a similar learning object model is presented. Their model is based on the idea of configurable samples, process called pedagogical parametrization, while our model is based on random numbers. The learning object is structured according the Bloom taxonomy cognitive layers, while in our approach the actions are driven by a curricula derived competence tree. The approach is

exemplified with a finite automaton study, while our approach is applied to two computer science disciplines. Their implementation uses Flash and Flex web technologies, while ours uses JavaScript and several associated libraries.

[15] exemplifies an application of the GLO concept in economical sciences, namely in a depreciation LO, based on template reuse.

3. The Autogenerative Learning Object Model

3.1 The Function Composition Mathematical Model

In this section we define the mathematical model of AGLOs instantiated with random numbers. By F_{JS} we denote all functions and operators from the JavaScript language standard [10]. By f_i we denote one function from the set of all available functions F_{JS}:

$$f_i(x): V \to V.$$

Vector $x =< x_1, x_2, ..., x_n >$ is the argument of $f_i \in F_{FS}$ and n is the number of arguments for the x vector, $x_i \in V$.

Set $V = I \cup D \cup St \cup A$ is the set of all possible argument values for f_i functions, composed out of:

i) integers $I = [-L_I - 1, +L_I] \cap Z$, where L_I is the range limit;

ii) doubles (floating point values) $D = [-L_D - 1, +L_D] \cap Q$, where $L_D \in N$ is the range limit;

iii) strings as vectors of character codes $St =< s_1, s_2, ..., s_{n_{St}} >$, where $s_i \in [0,255] \cap Z$ and n_{St} is the length of the string $n_{St} \in N$;

iv) arrays $Ar =< a_1, a_2, ..., a_{n_{Ar}} >$, where $a_i \in I \cup D \cup S$ and n_A is the length of the array $n_{Ar} \in N$.

We define the expression construction function e_i as:

$$e_i(x) = (f_1 \circ f_2 \circ ... \circ f_k)(x), \text{where}$$

$$x_i = \begin{cases} a, & a \in V \text{as constant value} \\ y, & y \in V \text{as random generated value} \end{cases}$$

We define AGLO as a tuple: $Ag =< S, T, Q, A, F >$ where: S is the scenario section, T is the theoretical section, Q is the question section, A is the answers section, F is the feedbacks section.

We define the scenario section as:

$$S =< e_1(y), e_2(y), ..., e_{n_S}(y) >,$$

as a vector of composed functions.

We define the theory section as:

$$T =< t_1, t_2, ..., t_{n_T} >, t_i \in [0,255] \cap Z.$$

We define the question section as:

$$Q =< q_1, q_2, ..., q_{n_Q} >,$$

$$q_i = \begin{cases} a, & a \in V \text{as constant value} \\ e_i(y), & y \in V \text{as random generated value} \end{cases}$$

We define the answers section as:

$$A =< a_1, a_2, ..., a_{n_A} >,$$

$$a_i = \begin{cases} a, & a \in V \text{as constant value} \\ e_i(y), & y \in V \text{as random generated value} \end{cases}$$

We define the feedback section as:

$$F =< f_1, f_2, ..., f_{n_F} >,$$

$$f_i = \begin{cases} a, & a \in V \text{as constant value} \\ e_i(y), & y \in V \text{as random generated value} \end{cases}$$

3.2 The EBNF Model

The proposed model is defined using the EBNF meta-language resulting in the concise grammar from figures 2 and 3. We will not present the full XML Schema because of clarity and space reasons.

Structurally, the AGLO model has six sections: i) name; ii) scenario; iii) theory; iv) question; v) answers; vi) feedbacks. Next, we will explain each section in details.

```
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 "</name>"
07 Type ::=
"<type>"
 ("boolean" | "int" | "float" | "double" | "string" | "array")
"</type>"
08 Expression ::=
"<expr>" Function "(" ExpressionList ")" "</expr>"
09 ExpressionList ::= Expression (, Expression)*
10 Function ::=
(element from functions and operators list of JavaScript
 using random numbers)
```

Figure 2. Autogenerative Learning Object Model Definition (1)

The scenario is the section where the educator expresses the main ideas of the learning object, informally, in natural language, just like programming language comments (see lines 03-04). By ID we denote the identifier token and by CT we denote any constant or literal: integer, float, character or string.

In the scenario section we define symbols that set up the details of the AGLO dynamic context. These symbols act as instantiation parameters for the AGLO in order to be referenced in the next sections. Such a symbol has several properties like: a name, a type and an initialization expression based on random numbers (see lines 05-10). The name is a regular identifier (see line 06), the type is an identifier in a specific non-restrictive list of: boolean, integer, float, double, string and array (see line 07). This meta-information is useful in symbol values further formatting.

Expressions are formed out of several unary, binary operators and functions that will not be detailed in the grammar from figure 2 because of space reasons (see line 10). In order to simplify the implementation and to increase its portability we rely on the JavaScript [10] expressions that can be evaluated with the ”eval” JavaScript function. These expressions are they key to the controlled random number based object instantiation. The expressions may call native JavaScript functions and also user defined functions gathered in domain specific libraries. For example, in the context of data structures we designed a function library capable of generating random trees controlled by several parameters like: the number of nodes, the number of levels, the degree etc. In this AGLO section the educator must define symbols for all the pedagogical logic of all remaining sections like: question, answers and feedbacks.

```
11 Theory ::= "<theory>" (ID)* "</theory>"
12 Question ::= "<question>" (ID | Value)* "</question>"
13 Value ::= "<value>" "<name>" ID "</name>" "</value>"
14 Answers ::= "<answers>" (Answer)+ "</answers>"
15 Answer ::=
"<answer>"
 "<id>" INTEGER_LITERAL "</id>" (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 3. Autogenerative Learning Object Model Definition (2)

Formally, the theory section (see line 11) contains only static data. Here the educator places theoretical information about the learned concept in HTML format.

The question, answers and feedbacks sections (see lines 12-13, 14-17, 18-21) contain a mix of static data and dynamic values referring the previously computed symbol values. The semantic of these sections is to create a dynamic content to be presented to the learner. Formalisms for specific details are also present, like:

i) correctness - because we need to know which is the correct answer if they are multiple;

ii) feedbacks answer identifier list - because feedbacks may be related to certain answers and showed to the learner only in certain conditions.

4. The Competence Tree

In this section we will present the support for our AGLOs. As we designed our AGLOs to refer micro-contexts we need an organizational structure for them. Grouping AGLOs in a simple sequence would result in poor access for both learner and tutor. We inspired ourselves from the curricula of disciplines and from their attached competences. In order to be able to model detailed learning aspects we structured the competencies hierarchically under the form of a five level tree, namely a competence tree.

Domains

We consider that the top level nodes are the domain nodes denoted by DOM. These nodes will correspond to full disciplines from the university curricula. In this article we validated our AGLO model approach on two disciplines, namely Data Structures and Algorithms and Algorithm Design and Analysis from University Politehnica Timişoara, Romania.

General Competences

On the second level of the competence tree we set the general competences denoted by GC. These nodes consist in main lecture topics like: array abstract data type or array linear search. In this particular case of the proposed disciplines we consider that both data structures and algorithms should be treated on the same level from the pedagogical point of view being aware of the fact that algorithms behave as operators that are applied to their corresponding data structure.

Specific Competences

On the third level of the competence tree we set the specific competences denoted by SC. These nodes refer to properties of the previous level nodes. For example, for the array abstract data type GC node we created the following SCs:

i) definitions - informal and mathematical;

ii) operators - mathematical notation, semantics;

iii) implementations - related to primitive language concepts or composed structures.

Another example of SCs for a searching algorithm contains the followings:

i) principles - main ideas of the algorithm;

ii) use cases - practical situations where the algorithm is used;

iii) input data - valid and invalid input data for the algorithm, specific properties;

iv) output data - valid and invalid output data, specific properties;

v) recognition in practice - several forms that the algorithm may have;

vi) algorithm steps - the general phases of the process;

vii) pseudocode - diagram and textual representations;

viii) code and moves - basic execution steps of the code lines;

ix) variables - the role of each variable, specific values;

x) other versions - improved versions, benefits, drawbacks.

Variables

On the fourth level of the competence tree we designed a special kind of nodes called variables denoted by VAR. These nodes represent another level of detail before the final nodes, namely the AGLOs.

Actions

On the fifth level of our competence tree we link the AGLO instances detailed in section 3 with action nodes or ACT.

5. Case Study on Computer Programming Disciplines

In this section we will present a case study applied on computer programming disciplines, namely on data structures and algorithms with emphasis on the generalized tree data structure.

5.1 AGLO Model Example

In this subsection we describe an AGLO model example. Figure 4 presents in details all the XML elements containing AGLO metadata.

```
<action dom="02" gc="11" sc="04" var="01" act="03">
 <name>
  <text>
   Act03 knows how to fill in the parent index array
  </text>
 </name>
 <scenario>
  <text>
   i)to generate a random tree with 8 to 12 nodes having
   character keys;
   ii) to compute the indexes array;
   iii) to compute the parent indexes array;
   iv) to access the names of the first two keys;
  </text>
...
</scenario>
```

Figure 4. AGLO Name and Description

The root XML element is the action element equipped with attributes reflecting its location in the competence tree. Domain 02 refers to the trees and graphs disciplines, general competence 11 refers to general trees, specific competence 04 refers to trees representations, variable 01 refers to parent index array representation while action 03, the analyzed AGLO, deals with filling in the parent index array for a randomly generated tree. The first XML element is reserved for the name of the analyzed AGLO that is displayed to the learner for localization and selection purposes.

The second XML element is the scenario element containing a text description of the AGLO and a set of symbols. The description is expressed in natural language and we can notice that it contains four main steps:

i) random tree generation;

ii) index computation for presentation;

iii) parent index computation for answer validation;

iv) access to the first two keys for particular feedback generation.

In the scenario section depicted in figure 5 several symbols are defined with the following semantics.

```
<scenario>
 ...
 <symbol name="nNoOfNodes" type="integer">
  random(8,12,0);
 </symbol>
 <symbol name="tabKeys" type="array">
  random_array({"nNoOfElements":v("nNoOfNodes"),
   "tabTypes":["character"]});
 </symbol>
 <symbol name="tree" type="tree">
  random_tree({"nNoOfNodes":v("nNoOfNodes"),
   "tabKeys":v("tabKeys")});
 </symbol>
```

```
 <symbol name="treeSVG" type="string">
  v("tree").toSVG();
 </symbol>
 <symbol name="tabIndexes" type="array">
  v("tree").indexes();
 </symbol>
 <symbol name="tabParents" type="array">
  v("tree").parents();
 </symbol>
 <symbol name="node1" type="character">
  v("tabKeys")[0];
 </symbol>
 <symbol name="node2" type="character">
  v("tabKeys")[1];
 </symbol>
</scenario>
```

Figure 5. AGLO Scenario Symbols

The first symbol is an integer and it is defined to store the number of nodes in the tree to be generated. The generation was done using the random() function with three parameters minimum value, maximum value and number of decimals. In this case we considered that the tree should have between 8 to 12 nodes.

The second symbol is an array that was created by random generation for the tree keys. The size of the array has the same value as the number of nodes. For the generation we used the random array() function which accepts as argument a JSON object with several properties like the number of elements and the array of types. For the current AGLO we used the previously generated value and the character type for the keys array. To access the value of a symbol we designed a value access function named v().

The third symbol is of type tree and is generated using the random tree() function accepting as arguments the number of nodes and the types array embedded into a JSON object. There are available also other generation parameters like degree or height for the tree.

The fourth symbol, named treeSVG, of type string, will store the SVG (Scalable Vector Graphic) representation of the generated tree. The SVG representation is obtained by transforming the tree generated data structure into SVG primitives like lines, texts, circles etc. This symbol will be used to display the graphic form of the tree in the browser for the learners eye.

The fifth symbol named tabIndexes of type array holds the array of node indexes, generally integer numbers from 0 to nNoOfNodes − 1. The symbol is initialized with a dedicated function call applied on the tree in object-oriented manner tree.indexes().

The sixth symbol name tabParents of type array holds the index array of parent nodes. The symbol is initialized with the value returned by a tree method tree.parents().

The seventh and eighth symbols named node1 and node2, both of type character, are initialized with the first two values from the keys array. These symbols will be used later in the feedback section. Their initialization is based on accessing the keys using the array indexing operator.

Next, we discuss about the presentation sections of the AGLO depicted in figure 6.

```
<question>
 Given the following tree:<br/> <value name="treeSVG"/>
 The key array is given below:<br/>
 <value name="tabIndexes"/><br/><value name="tabKeys"/><br/>
 Please, fill in the parent index array.
```

```
</question>
<answers>
 <answer id="1">
  <text><value name="tabParents"/></text>
  <correct>true</correct>
 </answer>
</answers>
<feedbacks>
 <feedback>
  For each node we fill in the index of its parent.<br/>
  Node <value name="node1"> has index 0 and has no parent
  so we will use the -1 invalid index.<br/>
  Node <value name="node2"/> has index 1 and his parent
  is <value name="node1"/>
  which has the index 0.<br/>
  etc.
 </feedback>
</feedbacks>
</action>
```

Figure 6. AGLO Presentation Sections

The question section will reference the tree SVG representation, the array of indexes and the array of keys. The referencing is implemented through the use of <value> XML elements qualified with name attributes. The semantic of these elements is that each element will be replaced with their corresponding value, so the e-learning content will be dynamic. Each running session will have a different tree and associated index arrays.

The <answers> XML element is composed out of several individual <answer> XML elements. Each <answer> element will be a mix of static and dynamic values. In the current AGLO example we used only one answer that is the correct one where we referred the array of parent indexes that the learner must fill in. The computed values and the filled in values are compared and a result is issued in the form of a mark to be transmitted to the learner.

The feedbacks section consists in several feedback elements having dynamic content. In our AGLO example we help the learner by providing the first two steps in order to show the logic of the response. In the current example we explain how to choose the parent index for the root node and how to get the parent of the next node. The rest of the exercise is left for the student.

5.2 AGLO Model Instantiation Example

In this subsection we give an example of a model instantiation. We describe the instantiation process for all the symbols in the AGLO model. The nNoOfElements symbol was instantiated with the generated value of 10. The tabKeys symbol was instantiated with the [A,B,C,D,E,F,G,H,I] generated array. The tree symbol was instantiated with a ”Tree” object built with arguments equal with previously instantiated symbols. Class ”Tree” is part of the auxiliary JavaScript library. The treeSVG symbol was assigned to the tree SVG string representation obtained by a method call and is depicted in figure 7.

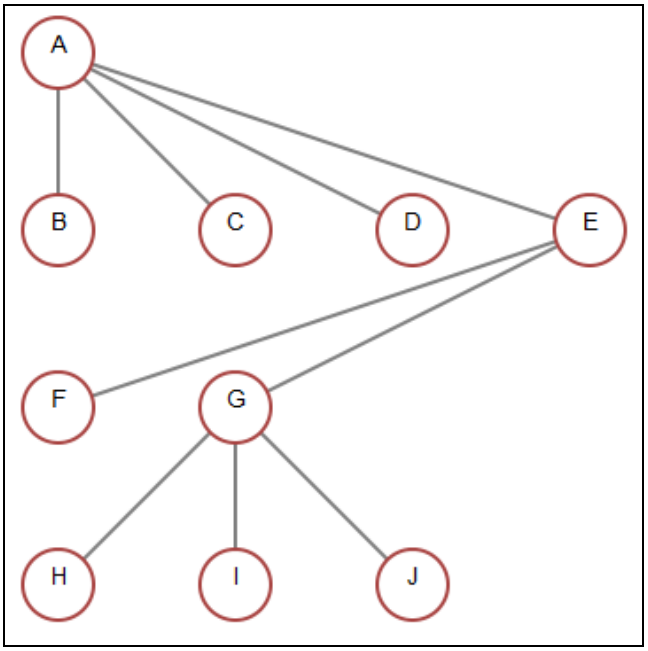

Figure 7. Generated Tree Example

The tabIndexes symbol was instantiated with the [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] computed array.
The tabParents symbol was instantiated with the [−1, 0, 0, 0, 0, 4, 4, 6, 6, 6] computed array. So B, C, D and E will have A as parent, F, G will have E as parent, etc.
The node1 and node2 symbols will get the A and B values from the tabKeys array.
All presentation sections will embed and exhibit the generated and computed values to the AGLO Interpreter component to implement interactivity.

DOM 01 search, sort	Actions no	DOM02 tree, graph	Actions no
GC10 linear search	19	GC11 generalized tree	70
GC11 binary search	53	GC21 binary tree	19
GC12 interp. search	11	GC22 binary ord. tree	5
GC21 insert sort	49	GC71 graph definitions	28
GC22 select sort	13	GC72 graph abs. data type	9
GC23 bubble sort	7	GC73 depth-first search	25
GC31 shell sort	6	GC74 breadth-first search	23
GC32 heap sort	11	GC81 Prim algorithm	10
GC33 quick sort	11	GC83 Krushkal algorithm	25
Total	180	Total	214

Table 1. Number of Actions in the Competence Tree

In table 1 we count the number of actions in each GC for the two implemented domains DOM01 (array searching and sorting) and DOM02 (tree, graph abstract data types).

6. Prototype Implementation

In this section we will present the main aspects about our prototype implementation. The prototype is implemented in JavaScript using the jQuery library and HTML markup language and following the principles of object-oriented programming technology as much as the language allows it. The prototype design consists in several classes created with the help of the “prototype” reflection facility of JavaScript programming language.

We created several modules in order to manage the implemented functionalities.
The user interface module contains the following classes:

i) index.js – represents the JavaScript interpreter entry point, creates and launches the application object;

ii) application.js – instantiates the message table object, sets up the path to the XML data files, initializes the AGLO interpreter and the table of symbols, initializes the breadcrumb object for navigation;

iii) menu.js – models the menus and the corresponding items ;

iv) breadcrumb.js – utility class that stores the current path in the competence tree in any moment of the runtime;

v) messages.js – models the internationalization feature of the prototype for Romanian and English languages;

The competences module contains several classes like:

i) domains.js – helper class allowing to manage domains;

ii) domain.js – models a domain and manages the general competences;

iii) generalcompetence.js – models a general competence and manages the specific competences;

iv) specificcompetence.js – models a specific competence and manages variables;

v) variable.js – models a variable and manages the actions;

vi) action.js – models an action and launches the AGLO interpreter;

The AGLO interpreter module contains the following classes:

i) glointerpreter.js – builds the HTML presentation for the considered action;

ii) tableofsymbols.js – class which manages symbols based on random values;

iii) symbol.js – models the symbol with name, type, initialization expression and value;

iv) theory.js – models the theoretical section;

v) question.js – models the question section;

vi) answers.js – models the answers section;

vii) feedbacks.js – models the feedback section.

For the content specific functionality we designed a random module which contains functionality generating random data structures according to parameters:

i) random.js – models a generator for random numbers like integers, floats, characters, strings;

ii) randomarray.js – models a generator for random arrays according to several parameters;

iii) randomorderedarray.js – models a generator for random ordered arrays according to several parameters;

ii) tree.js – models a random tree generator according to several parameters, expressed as SVG markups;

iii) graph.js – models a random graph generator according to several parameters, expressed as SVG markups;

iv) utility classes like: node.js, circle.js, line.js used to build the tree and graph data structures and to export them as SVG markups;

v) linearsearch.js – models several running states of the linear search algorithm, initiated by random input.

For the front-end we used the Bootstrap library to implement navigation in the competence tree with the help of a breadcrumb class in order to mark the current path in the tree. For the AGLO interpreter implementation we used xml-to-json library, to facilitate the access to XML elements of the AGLO files. We designed a class to model a table of symbols having HTML DOM tree

translation facilities. We modeled classes for each AGLO section: scenario, theory, questions, and feedbacks. We modeled domain specific classes for the learned concepts like trees and graphs having generative methods controllable through parameters.

Figure 8 depicts our prototype front-end.

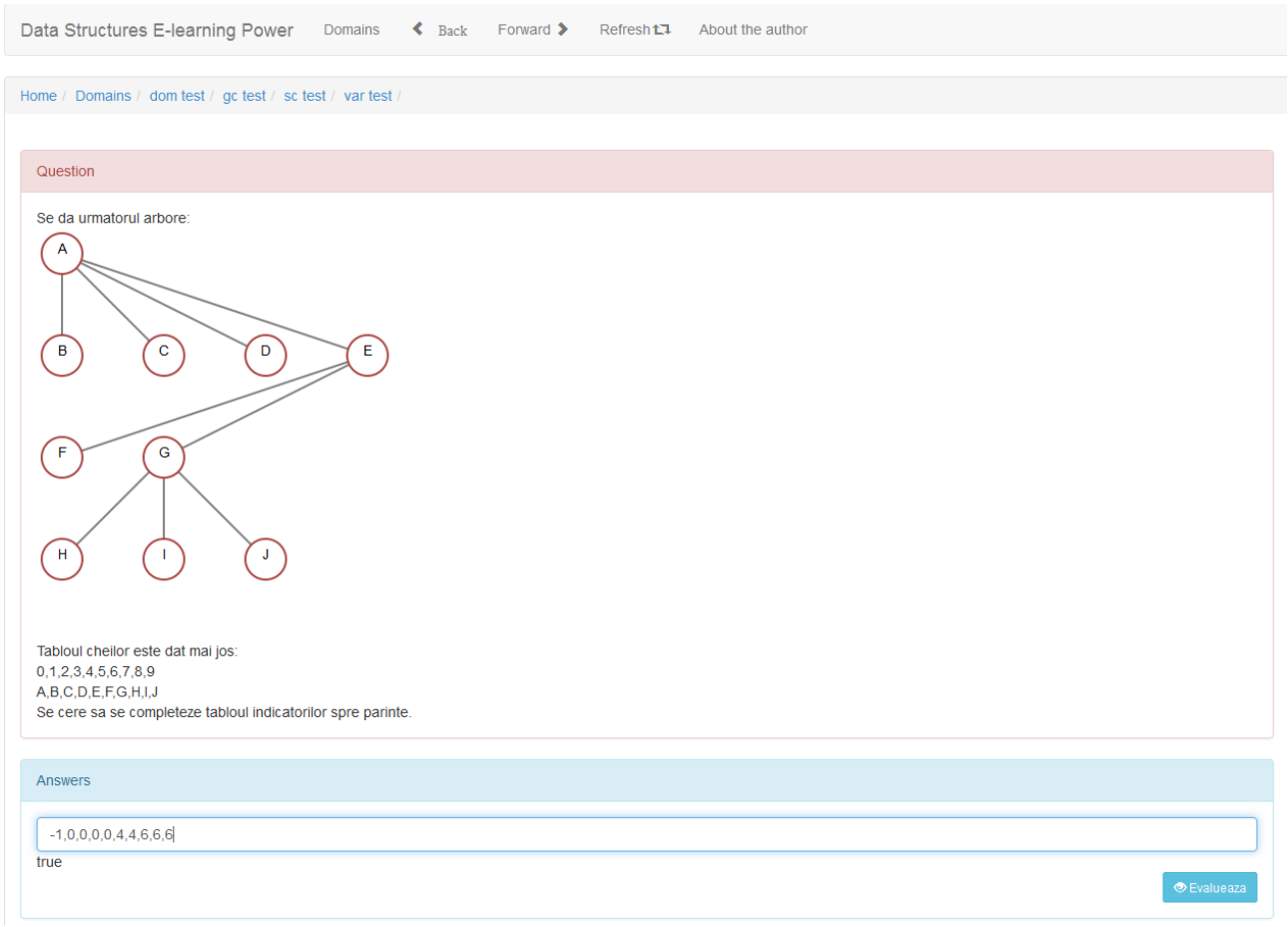

Figure 8. Prototype Front-End

7. Experimental Results

After using AGLOs in knowledge assessment we invited the subject students to fulfill an impression questionnaire on which we computed analytics and drew conclusions.

Most of the subjects work in companies 92.9%. Their age is balanced between 21 and 30 at 50%, between 31 and 40 at 42.9%, while over the age of 41 there are only 7.1%. Among the subjects 78.60% are male, while 21.4% are female. All subjects 100% are from Timisoara. 78.60% study in a private university, while 14.3% study in a state university.

The majority of 92.90% use their laptops in their study process with a maximum intensity of 5 on scale from 1 to 5, while only 7.10% use their laptops with an intensity of 4 on the same scale. The tablet use is quite low 21.40% use the tablet regularly, 7.10% on rare occasions, and 71.40% on very rare occasions. The mobile phones use statistics shows that 42% of the subjects use them very often and 28.60% use them on very rare occasions. Given the results we can estimate that the mobile phone usage will increase. We have to include in the further developments a mobile phone version of the web AGLO application or at least optimizations in this sense.

A considerable percentage of the subjects 78.6% declared that they understood the semantics of AGLOs. 50% of the subjects consider the AGLOs as good and 21.4% as very good for their study process. 21% would like to use them in the near future. 0% considers them boring. The AGLO uses are spread among lectures 42.90%, laboratory works 85.70%, and laboratory tests 42.9%. Regarding

the attractiveness of the AGLO concept we got the following results: i) 35.70% liked it very much; ii) 50% liked it much; iii) 14.30% liked it equally.

The usefulness of the AGLOs was measured at 78.60%. The subjects would use AGLOs mostly in laboratory activities: 71.40% and less in final exams: 28.60%. The importance of the restarting capability of an exercise object with different input data and new contexts was measured at 71.40%. In order to prevent exam frauds we proposed the use of generated subjects with a unique instance per student. The importance of this aspect is measured at 85.7% for the subjects. The availability for AGLOs modification and tuning decreases linearly from 35.70% to 21.40% in the second half of the graphic. 71.5% of the subjects would recommend AGLOs to acquaintances. An overwhelming 78.60% would like to use editing fields for the creation, modification and tuning of AGLOs in the expense of text formats XML and JSON.

8. Conclusions and Future Work

The current paper presents an expressive model of learning object AGLO with variable content based on controlled random number parameters and automatic initialization and answer correctness assessment. We created facilities that enable the development of engaging e-learning materials using formalisms of flexible complexity through the AGLO model and its interpreter. Learners will benefit from the content variability feature of the AGLOs and from the interactivity implemented through the feedback mechanism. Educators have new methods and tools for developing dynamic learning objects with different levels of complexity depending on their imaginative and modeling skills. From the implementation point of view the prototype uses modern Web 2.0 technologies that enable portability on multiple platforms and mobile devices.

The approach has also several drawbacks. One drawback is related to expressing new semantics. The solution adopted in the generative model relies on domain specific libraries when the learning content is complex. Another drawback is related to AGLOs development time which is increased compared to the development time for static materials. In our experiments we learned that educator practice and experience reduce this development time and that is directly related to the complexity of the designed AGLO. A different drawback is related to testing complexity of AGLO which is difficult because of the random numbers based generated expressions. One solution is to use the unit testing technique and to decouple the random numbers generation with extreme condition simulation.

As future work we intend to integrate the AGLO concept in widespread learning management systems. Another direction of research is in the area of AGLO development facilities. For example, a pattern catalog would enable educators to create more easily diverse and complex AGLOs.

9. Acknowledgement

This paper is supported by the Sectorial Operational Programme Human Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the project number POSDRU 159/1.5/S/134378.”.

10. References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Chirila, C.B. (2014), Generative Learning Objects in Biomedical Informatics, 2014 Romanian Medical Informatics Conference (RoMedInf), Bucharest, Romania, June.

[7] Chirila, C.B. (2014), Generative Learning Object Assessment Items for a Set of Computer Science Disciplines, In proceedings of SOFA 2014 6-th International Workshop on Soft Computing Applications - Advances in Intelligent and Soft Computing, Springer Verlag, ISSN 1867-5662, Timisoara, Romania, July.

[8] Damasevicius, R.; Stuikys, V. (2008), On the Technlogical Aspects of Generative Learning Object Development, Third International Conference on Informatics in Secondary Schools - Evolution and Perspectives (ISSEP 2008), pp. 337-348, Torun, Poland, July.

[9] 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.

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

[11] 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.

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

[13] 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.

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

[15] Oldfield, J.D. (2008), An implementation of the generative learning object model in accounting, In proceedings of Asciilite (Australasian Society for Computers in Learning in Tertiary Education), Melbourne, Australia.

[16] Stuikys, V.; Damasevicius, R. (2007), Towards Knowledge-Based Generative Learning Objects, Information Technology and Control, 36(2), ISSN 1392-124X.

[17] Stuikys V.; Damasevicius R. (2008), Development of Generative Learning Objects Using Feature Diagrams and Generative Techniques, Journal of Informatics in Education, volume 7(2), pp. 277-288, Institute of Mathematics and Informatics, Vilnius, Lithuania.

[18] Stuikys, V.; Brauklyte, I. (2009), Aggregating of Learning Object Units Derived from a Generative Learning Object, Journal of Informatics in Education, volume 8(2), pp. 295-314, Institute of Mathematics and Informatics, Vilnius, Lithuania.

[19] 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.

[20] Wiley D. (2000) Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. D. Wiley (Ed.), The instructional use of learning objects: Online version. Retrieved January 05, 2015 from http://reusability.org/read/chapters/wiley.doc.

Ciprian-Bogdan CHIRILA (b. 1977) received his M.Sc. in Computer Science (2002) and PhD in Computer Science (2010) from University Politehnica Timişoara and University of Nice Sophia-Antipolis. Now he is senior lecturer at Computer and Information Technology Department, University Politehnica Timişoara, Romania. His current research interests include different aspects of object-oriented class reuse mechanisms and generative learning objects in the context of e-learning. He has (co-)authored 4 books and more than 35 papers. He is IEEE member since 2012 and treasurer of the IEEE SSIT Romanian section.

Horia CIOCÂRLIE (b. 1953) received his M.Sc. in Computer Science (1976) from University Politehnica Timişoara and PhD in Computer Science (1991) from University Politehnica Timişoara, Romania. His current research area focuses on parallel, concurrent and distributed languages (definition and implementation) and Compiling Techniques. He has (co-)authored 9 books and more than 120 papers.

Lăcrămioara **STOICU-TIVADAR** (b. 1961), received MSc in Automation and Computers in 1985 from Politehnic Institute Traian Vuia Timişoara, PhD degree in 1999 from University Politehnica Timişoara, and is Professor in the same university since 2005. Her research interests are in computer modelling of biomedical systems, data processing, ontology in medicine, eHealth. She has over 150 papers and books, and participated in national and international projects. She is IEEE member since 2007.