The 12th International Scientific Conference
eLearning and software for Education
Bucharest, April 21-22, 2016
10.12753/2066-026X-16-000

Reuse Models for Generative e-Learning Content Dedicated to Computer Science Disciplines

Ciprian-Bogdan Chirila
Department of Computer Science and Software Engineering,
University Politehnica Timişoara, V. Pârvan Blv. no. 2, Timişoara, Romania
chirila@cs.upt.ro

***Abstract**: Nowadays the e-learning domain has different development directions. Learning management systems (LMS) tend to integrate standardized content like: Shareable Content Object Reference Model (SCORM), Aviation Industry Computer Based Training Committee (AICC), etc. In products like Storyline 2 and Studio'13 the focus is set on the development of content based on slides. They start from Microsoft PowerPoint slides and enhance it with several facilities like: narrations, annotations, motion paths, screen recordings, videos, iterations, conditional interactions, simulations, language support. Another focus of these products is set on content publishing to various platforms like iPads, Android tablets etc. The Captivate e-learning content authoring tool contains facilities like: to create content for iPads and tablets with responsive design, storyboards based on slides, multiple choice templates, text, image and video galleries, sync with the cloud, e-mailing facilities of the just created story boards, the content is expressed as a Flash clips and HTML5 web pages played on most of the browsers. xAPI is a flexible specification allowing to track informal learning, social learning and real world experiences. The recording format is a very generic one in the form of actor, verb and object memorized in a learning record store (LRS). SCORM (Shareable Content Object Reference Model) is a set of standards for e-learning software in order to increase integration of e-learning content. Generative learning objects (GLO) are reusable pedagogical templates to be filled with content obtained in several ways. One efficient way for e-learning content generation is to use meta-programming on generative models. In this paper we present several generative models to be reused in authoring computer science (CS) e-learning content. The first model we propose is a CS text problem composer embedding features like: composition rules for generating learning objects, linked lists problems generation, modelling problems being built around the composition concept. A second model is a code refactorer based on several refactoring rules like: changing variable names, changing code indentations, changing loop instructions etc. in order to be used by first year students to recognize different algorithms. A third model is a code tamperer based on several code tampering rules used to affect the sensitive sections, operators, variables, etc. of an algorithm where students will have to identify the inserted faults. In this model we include a source code block randomizer component based on abstract syntax tree (AST) subtree swaps and other rules. A source code line randomizer can be included in the same context based on swapping sensitive lines in an algorithm selected manually or automatically. A fourth model is demonstrator based on several concepts like: to give as input an algorithm, to give as output the visualization of the data structure changes during the algorithm run.*

***Keywords:** generative learning objects, meta-programing, computer science*

I. INTRODUCTION

The e-learning domains has several development dimensions. On one hand we have learning management systems (LMS) which integrate and standardize e-learning content using several formats like: SCORM (Shareable Content Object Reference Model), AICC (Aviation Industry Computer Based Training Committee), etc.

Other products developed by strong market player companies tend to enhance the development and production of e-learning content based on slides. Products like Storyline 2 [4] and Studio'13 [5] enhance Microsoft PowerPoint slide documents with advanced facilities like: narrations, annotations, motion paths, screen recordings, videos, iterations, conditional interactions, simulations, language support. After the material preparation, it must be published on various platforms like iPads, Android tablets, etc.

The Captivate product [1] has similar features like:
i) to create responsive design content for iPads and tablets;
ii) to create story boards based on slides;
iii) to offer multiple choice templates;
iv) to offer basic elements like: text, images, videos from galleries;
v) to sync the e-learning content with different cloud storage services;
vi) to enable e-mail facilities for the newly created content;
vii) to export the educational content into Flash clips or HTML5 web pages which offer portability on most of the nowadays web browsers.

xAPI [3] is a flexible specification of learning record storage formats and is meant to track informal learning, social learning and real world experiences. The format is based on the actor, verb and object tuple that is to be stored in a learning record storage (LRS).

SCORM [18] is a standard designed to favor the integration and interoperability of e-learning content in e-learning software.

Generative Learning Objects (GLO) [6,7,8,17] are reusable pedagogical templates that must be filled in with content that can be done using different methods. One common method is to use meta-programming of generative models in order to generate e-learning content.

The current paper presents several generative models that can be reused in authoring computer science (CS) e-learning content. Our approach is based on the integration in the model of parameters based on random numbers that will enable diversity in the universe of the generated CS problems and exercises with automatic or manual correctness assessment.

The paper is structured as follows. In chapter II we discuss general aspects regarding generative learning objects reuse models. In chapter III we present and analyze a composition problems instantiation component. In chapter IV we present and analyze a code refactoring component for generating exercises. In chapter V we present the server pages like based approach for generative learning objects in comparison with the other approaches. Chapter VI presents ideas for a basic algorithm simulator. Chapter VII concludes and sets the perspectives.

II. REUSE MODELS

In this chapter we present AGLO auto-generative learning objects reuse model [10,11,12,13,23]. The approach (see Figure 1) starts with an abstraction of the problem, namely the problem model. These models can be obtained from practice situations through generalization. In this stage we identify parameters, their range and their functional dependencies.

The second stage of the approach is the instantiation based on random numbers. This stage implies computations for parameter values in concordance with their dependencies.

The final stage is the resulting of the concrete problem which can be consumed by the student in the educational process.

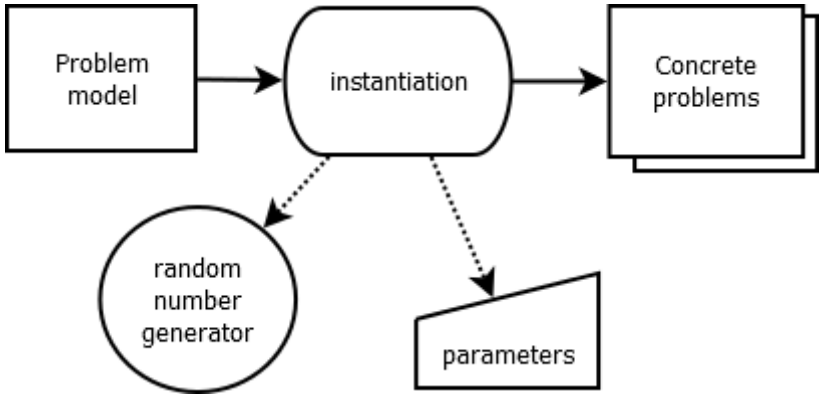

Figure 1. Generic Problem Model

Similar approaches are presented in [9,14,15] and in [19,20,21,22] where GLOs are designed and implemented using a suite of technologies and tools like: meta-programing, feature models, code generators etc.

III. COMPOSITION PROBLEMS INSTANTIATION COMPONENT

In this chapter we describe a set of principles in order to generate student problems based on generic models. We focused on the generation of problems for linked lists organized on two levels. This set of problems cannot be automatically verified but a solution could be generated to the student for feedback purposes. The model of the problem is based on two classes of objects <<Whole>> and <<Part>>.

Figure 2. Composition Problem Model

The class relationship between the concepts is aggregation but the composition relationship is also good. Such a linked lists problem starts with the description of the two concrete classes that will form the concrete problem through instantiation. For example the <<Whole>> class could be instantiated with FootballTeam and the <<Part>> with FootballPlayer.

On the fist level the linked list operators that the student has to write are abstracted in a generic list like:

- initialization();
- add_<<Whole>>(<<param_1>>, <<param_2>>,..., <<param_n>>);
- update_<<Whole>>(<< param_1>>, <<param_2>>,..., <<param_n>>);
- delete_<<Whole>>(<<param_1>>);
- listAll_<<Whole>>().

The operations proposed are a minimal, but if needed the name of the arguments can be involved, for example we can have an operation add_<<Whole>>_orderredBy_<<param_1>> where the list element will be added ordered by <<param_1>>.

For example the generic list in the context of FootballTeam will be instantiated as:

- initialization();
- add_FootballTeam(name, financedSum);
- update_ FootballTeam(name, financedSum);
- delete_ FootballTeam(name);

- listAll_ FootballTeam().

The adding operator could be add_FootballTeam_orderredBy_Name(...).

On the second level of the linked lists we can use the following operators:

- initialization(<<Whole>>_id);
- add_<<Part>>(<<Whole>>_id, <<param_1>>, <<param_2>>,..., <<param_n>>);
- update_<<Part>>(<<Whole>>_id, << param_1>>, <<param_2>>,..., <<param_n>>);
- delete_<<Part>>(<<Whole>>_id, <<param_1>>);
- listAll_<<Part>>(<<Whole>>_id).

The semantics is the same as above but related to a <<Whole>> instance reference. The variability of this model consists in the two instantiable classes, in the list of attributes for each class and in the subset of operations list.

IV. ALGORITHM REFACTORING COMPONENT

In this chapter we present a set of refactoring ideas that will transform algorithm syntax while it will preserve its semantics in order for the student to recognize it. Reasonable ideas in this sense are as follows. One idea is to change variable names between them, this implies a permutation of the identifiers list. Another idea is to change variable names with external names based o categories like: arrays, indexes, temporary variables, etc. Code indentations can be changed like switching between micro-settings of each syntactical unit in the sense of Eclipse code profiles. Another refactoring is to replace loops where possible. Thus, a "for" instruction can be replaced with a "while" instruction and enables student thinking in the recognition of a classic algorithm. Of course that any combination of the presented code refactorings is possible.

For example, an insertion sorting algorithm may have the following variants:

```
// variant v1
void insertsort(int *tab, int n)
{
    int i,j;
    int aux;
    for(i=1;i<n;i++)
    {
        aux=tab[i];
        j=i-1;
        while(tab[j]>aux && j>=0)
        {
            tab[j+1]=tab[j];
            j--;
        }
        tab[j+1]=aux;
    }
}
```

```
// variant v2
void insertsort(int *tab, int n)
{
    int i,j;
    int aux;
    i=1;
    while(i<n)
    {
        aux=tab[i];
        for(j=i-1;tab[j]>aux&&j>=0;j--)
        {
            tab[j+1]=tab[j];
        }
        tab[j+1]=aux;
         i++;
    }
}
```

Figure 3. Variants of insertion sorting algorithm

In Figure 3 we can notice that the outer loop is a "for" in variant v1 and is a "while" in variant v2. The inner loop of the algorithm is vice versa, namely variant v1 uses a "while" instruction and variant v2 uses a "for" instruction. Other variants could use both "while" loops or both "for" loops.

The variability of this component consists in the fact that it can be applied generically to any algorithm subject in CS disciplines and can be configured by enabling the code refactorings presented earlier.

V. ALGORITHM TAMPERER COMPONENT

This component takes as input a correct classic algorithm and outputs a tampered code algorithm embedding smooth mistakes for the student to identify. In this sense we propose a list of basic alterations to be applied on the code like:

- to change the value of constants by increasing or decreasing its value by 1;
- to reverse the sense of operators, e.g. the < operator will be replaced by the > operator;
- to replace similar operators, e.g. the && operator can be replaced by the || operator;
- to change the order of two basic instructions that are related to each other, so the semantics of the code is changed;
- etc.

Figure 4. Error injection component

For example, the sorting by insertion algorithm may be tampered in several ways and in multiple points (see Figure 5).

```
// variant v1
void insertsort(int *tab, int n)
{
    int i,j;
    int aux;
    for(i=1;i<n;i++)
    {
        aux=tab[i];
        j=i-1;
        while(tab[j]>aux && j>=0)
        {
            tab[j+1]=tab[j];
            j--;
        }
        tab[j+1]=aux;
    }
}
```

```
// variant v3
void insertsort(int *tab, int n)
{
    int i,j;
    int aux;
    for(i=1;i>n;i++) // 1 error
    {
        aux=tab[i];
        j=i+1; // 1 error
        while(tab[j]>aux && j>=0)
        {
            tab[j+1]=tab[j];
            j--;
        }
        tab[j+1]=aux;
    }
}
```

Figure 5. Original and tampered insertion sorting algorithm

VI. BASIC ALGORITHM SIMULATOR

This generic component is designed to read an algorithm and to simulate its functionality by depicting each step using colored visualizations of the data structures (see Figure 6). The goal of this component is to make the student understand better the functionality of an algorithm through data modification dynamics.

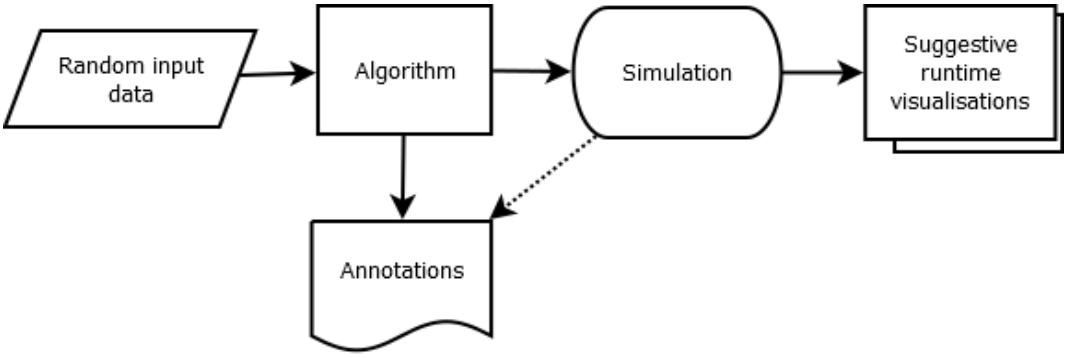

Figure 6. Algorithm simulator component

In order to accomplish our goal we developed several concrete simulators in order to collect their common characteristics trying to abstract a generic model. We implemented concrete simulators for the following basic algorithms:

- Linear search;
- Binary search;
- Sorting by insertion;
- Sorting by selection;
- Sorting by interchanging elements also known as bubble sorting.

For starters we decided that all algorithms should rely on simple data structures: integer values, integer indexes and arrays.

The abstracted principles are:

- variables should have attached a color in order to create a suggestive visualization;
- variables should be categorized automatically into: values, indexes and arrays;
- in the algorithm dynamic the changed values of indexes and their indexed array value should be highlighted with a different color;
- the ranges of elements should be highlighted, but this still remains a challenge.

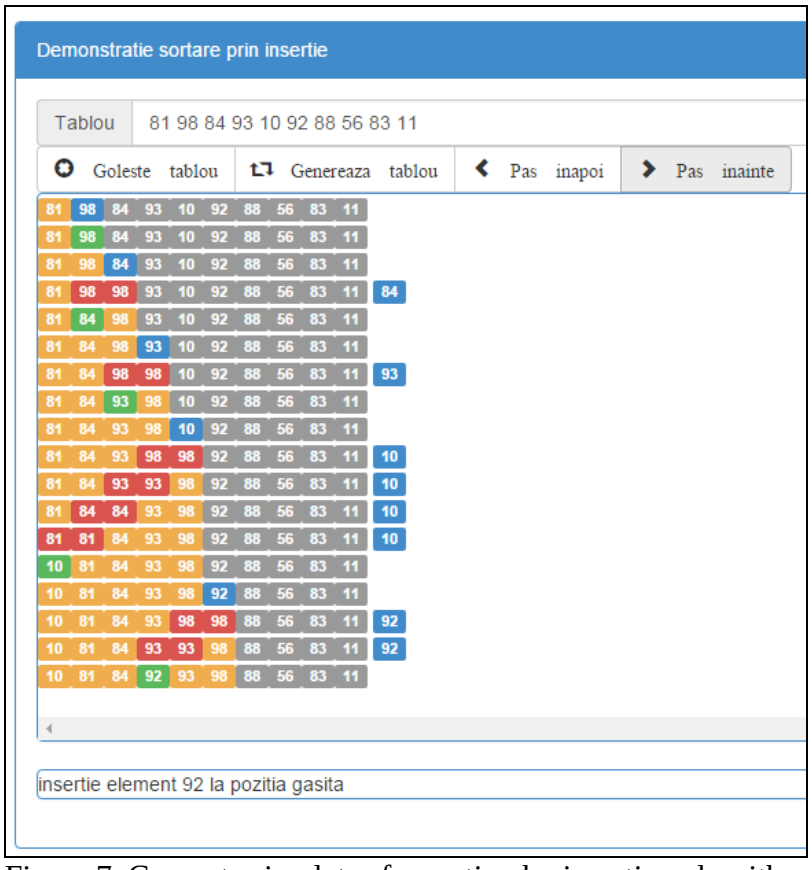

Figure 7. Concrete simulator for sorting by insertion algorithm

In Figure 7 we have an example of the insertion sorting algorithm. The sorted array range is displayed in orange. The extracted element color is blue. The currently compared elements are depicted in red. The inserted element position is green.

Some visualization semantics could be transmitted as inline code comments. The implementation of such a simulator involves AST nodes analysis and an execution environment like a virtual machine.

VII. CONCLUSIONS AND PERSPECTIVES

In this paper we presented generative models and principles to be used in the creation of e-learning content. We showed that several concrete problems can be generalized in order to generate other instances useful for the students in their preparation. We showed for each model which are the reuse dimensions. The use cases selected for these models are not revolutionary, they have to role of enlarging the research in the area of GLOs and CS educational components.

As future work we want to implement fully the described ideas. For the problem instantiation component the implementation challenge stands in integrating various whole-part concepts with concrete parameters in the problem context. The algorithm refactoring component uses AST node transformations. The challenge here is to prove that the obtained algorithm preserves the behavior of the original one, especially in the context of composed transformations. The error injector must manipulate also the AST and produce smooth mistakes which make sense, so the student to detect them.

We want to use these models in practice, during lectures and laboratory works and to assess their impact on students. Another perspective is to integrate these components in web applications on the server or on the client side as suitable.

References

[1] Adobe, 2016. Captivate 9, http://www.adobe.com/products/captivate.html

[2] Advanced Distributed Learning Network, 2016. ADL Network website, http://www.adlnet.gov

[3] Advanced Distributed Learning Network, 2016. Experience API, http://adlnet.gov/adl-research/performance-tracking-analysis/experience-api/

[4] Articulate, 2016. Story Line 2, https://www.articulate.com/products/storyline-why.php

[5] Articulate, 2016. Studio'13, https://www.articulate.com/products/studio.php

[6] Boyle, T., 2003. Design principles for authoring dynamic, reusable learning objects. Australian Journal of Education Technology, vol. 19, no. 1, pp. 46-58

[7] 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

[8] Boyle, T., Bradley, C., 2009. User Guide for the GLO Maker 2 Authoring Tool, http://www.glomaker.org

[9] Burbaite, R., Bespalova, K., Damasevicius, R., Stuikys, V., 2014. Context Aware Generative Learning Objects for Teaching Computer Science, International Journal of Engineering Education, vol. 30, no. 4, pp. 929-936, 2014.

[10] 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

[11] 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

[12] 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

[13] 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

[14] Damasevicius, R., Stuikys, V., 2008. On the Technological 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

[15] Damasevicius, R., Stuikys, V., 2008. Specification and Generation of Learning Object Sequences for e-Learning Using Sequence Feature Diagrams and Metaprogramming Techniques, In proceedings of 2009 9-th Intenrational Conference on Advanced Learning Technologies, 2009.

[16] IEEE Learning Technology Standards Committee, 2016. LOM working draft v4.1 Available: http://ltsc.ieee.org/doc/wg12/LOMv4.1.htm

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

[18] Rustici Software, 2016. SCORM, http://scorm.com/scorm-explained/

[19] Stuikys, V., Brauklyte, I., 2009. Aggregating of Learning Object Units Derived from a Generative Learning Object, Journal of Informatics in Education, vol. 8, no. 2, pp. 295-314, Institute of Mathematics and Informatics, Vilnius.

[20] 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, vol. 12, no. 1, pp. 125-142, Institute of Mathematics and Informatics, Vilnius.
[21] Stuikys, V., Damasevicius, R., 2007. Towards Knowledge-Based Generative Learning Objects,Information Technology and Control, vol. 36, no. 2, ISSN 1392-124X, 2007.
[22] Stuikys, V., Damasevicius, R., 2008. Development of Generative Learning Objects Using Feature Diagrams and Generative Techniques, Journal of Informatics in Education, vol. 7, no. 2, pp. 277-288, Institute of Mathematics and Informatics, Vilnius.
[23] Vlașin, I., 2013. Competența: participarea de calitate la îndemâna oricui, Editura Unirea, Alba Iulia