The 11th International Scientific Conference
eLearning and software for Education
Bucharest, April 25-26, 2015
10.12753/2066-026X-15-000

Generative Models for Computer Programming 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 IT industry is more and more present in several areas like: automotive, telecommunications, finance, medicine, pharmaceutics etc. One of the main problems in the development of these domains is the lack of highly qualified IT specialists, thus loosing million euros projects and afferent taxes. Among the IT specialists there is a significant segment allocated to programmers and testers. In order to train more IT specialists universities introduced several programs including distance learning and evening courses. The problem with these students is that they get weak results because of the limited in class face to face learning amount of time. The use learning management systems based on learning objects (LO) can ameliorate this problem. Generative learning objects (GLO) are instantiable generic templates that generate concrete learning objects. They are considered to be the second generation of learning objects (LO). They can be used in several learning areas like: economy - accounting, medical science - nursing, computer science - understanding programming with the help of robots. In this paper we will present a set of generative models designed for the e-learning of computer programming disciplines. In order to enable accessibility, enhanced understanding, creativity stimulation of computer programming we designed a set of generative models which are based on the GLO paradigm obtaining automatically concrete LOs. Our focus is set on providing these models to GLO authors with less programming skills together with the feature of automatic instantiation based on random number generators. Our models were designed for data structure disciplines in the context of developing programming competences, namely how to program using data structures like: arrays, lists, trees and graphs.*

***Keywords:** generative learning objects, generative models, computer science disciplines*

I. INTRODUCTION

The IT (Information Technology) industry produces technologies for almost all other industries in the global economy. These technologies are present in areas like: automotive where cars have a high degree of automation based on software, telecommunications where software is used for control and management, finance where software is needed for banks, stock exchanges, medicine where software is used for managing health systems etc.

The IT industry in the Eastern European countries is under a constant pressure because of the increasing number of incoming IT projects and the migration of the human resources. Human resources usually migrate to companies located in the Western Europe. Each year software companies must reject million euros projects because of lacking human resources. In order to overcome this problem universities must train software engineering students better and faster and deliver them to the industry to increase company capability of accepting and developing more and larger software projects. Companies employ students early in the third year of study in part time of full time programs.

Students tend to be more and more digitally minded, they use more often in their daily life mobile phones, tablets, laptops for information and entertainment (social networks) and also for learning using e-mails, blogging, microblogging, wikis, polls, surveys, collaborative writing tools etc.

The IT specialists face several problems like: limited face to face study time since they are employed very early, limited time to study because they already work in software testing projects with low demands, waiting time not used properly in their personal lives, lack of interactive learning infrastructures. One solution is the use of advanced e-learning techniques in the context of MOOCs [14,15] in this sense which are decoupled in time and space.

The currently used LMSs (Learning Management System) in universities offer support for structuring learning materials and for human interaction through several resources like: polls, wikis, surveys etc. They allow but not necessarily favour the creation of learning objects for content reuse and repurposing.

The current work is in the frame of e-learning and specifically in the area of learning objects. A learning object (LO) [24] is considered to be a deliverable digital resource having a single learning objective designed to reuse its content. In this sense there are several standards like IEEE-LOM [17], SCORM [1,19], Dublin Core [11] in order to offer support at the syntactical level and some semantic support like content composition, content sequencing, learning data result storage.

The limitations of the current approaches are multiple. General learning objects have static contents, indications, feedbacks while the generative ones "imagine" new scenarios at each use case. GLOs even considered to be the second generation learning objects they are not mature enough [23] and are not used on a large scale in practice. This is due to the fact that there is no standard, a formal description of generative learning objects and there are no software tools to implement those standards and to favor the development for this type of LOs.

In this paper we present several generative models for the learning, training and assessment of computer programming disciplines, namely for algorithms. Our generative models are based on a server pages like approach and have been applied successfully to learning mathematics [7] with dialog games and for the assessment of several computer science disciplines [8]. In this paper we explore new models of generative learning objects applied to data structures and algorithms. Our ideas intend to increase the reuse, accessibility and expressivity of learning objects.

The paper is structured as follows. In chapter II we present general concepts regarding generative learning objects. In chapter III we present the model of our data structure disciplines curricula. In chapter IV we present our generative models for data structure disciplines. Chapter V presents related works, while chapter VI concludes and sets the perspectives.

II. RELATED WORKS

In this chapter we present several approaches of LOs closely related to our work. A special category of LOs is the category of generative learning objects (GLO) which are considered to be the second generation of learning objects [4]. The direction in the research of generative learning objects was launched a few years ago by a group of pioneers lead by Professor Tom Boyle and are based on software engineering principles like: modularity, cohesion, decoupling and composition [3].

There are two research groups with different approaches relative to the generative part of these objects: i) Tom Boyle, Ray Jones [16] on one side and ii) Vyutas Stuikys [20,21,22,23], Robertas Damasevicius [9,10,6] on the other having different areas of application. There are also other implementations with similar ideas in [18,13].

A GLO is defined as a LO where the reuse element is the pedagogical template like design patterns [2,12] and not the learning content. The learning content is added to the template or it can be generated by using generative techniques like metaprogramming. The added content can belong to different learning domains thus the model is reusable. The content generation is an instantiation process similar to the one used in object-oriented programming languages.

In the approach of Boyle a learning object targeting a looping instruction is composed out of several frames (learning steps): i) introduction of the concept; ii) comparison of the concept with a familiar one; iii) the presentation of the concept; iv) the detailed presentation of the concept; v) a scaffolded exercise. In order to create such LO one can use the GLO Maker [5] tool.

In the approach of Stuikys and Damasevicius the GLO is expressed as a script in Open PROMOL meta-programming language, configurable by parameters, which is generated into HTML or text consumable LOs at instantiation time. The scripting meta-language is based on text manipulation and on the functional paradigm. Some of the created GLOs are used in combination with LEGO robots and Arduino systems in order to teach basic programming concepts.

In our GLO approach we consider that simple mathematical expressions allow the generation of quite complex learning objects. Text and numerical values can be manipulated in order to express the business logic of the learning unit.

III. DATA STRUCTURE DISCIPLINES COMPETENCE TREE

In this chapter we present our approach regarding the competence tree for the data structures and algorithm discipline.

Firstly, each discipline is modeled as a domain in the tree. In our university curricula the data structures discipline is taught in two lectures: i) the first for simple structures like arrays, strings, lists, hash tables and ii) the second for trees and graphs. Our competence tree refers to several data structures like: arrays, trees and graphs. All algorithm competences are modeled on the data structures: e.g. searching and sorting on the arrays, building and searching on trees, etc.

Secondly, under the domain nodes we modeled general competences which refer to one basic concept set as learning objective. It can be either a data structure or an algorithm. The algorithm will always follow in order the data structure it runs on.

On the next level we designed specific competences which refer to main topics related to a concept modeled at the higher level. For example, the array data structure general competence has several specific competences like: i) to know the definition of the array; ii) to know the operators of the array. In case of an algorithm e.g. like those for searching and sorting the specific competences may refer to the following aspects: i) to know in which situations the algorithm must be used; ii) to know the properties of the input data of the algorithm; iii) to know the output of the algorithm and its properties; iv) to identify in practice the algorithm; v) to know the steps of the algorithm; vi) to know the block diagram of the algorithm; vii) to link the data structure moves with the steps of the algorithm; viii) to know the variables of the algorithm; ix) to know about the enhanced version of the algorithm if any. These steps are just a proposition of a refined structure of competences that will hold the generative actions. For example, the specific competences of a data structure may refer to aspects like: i) definition; ii) operators; iii) basic implementations; iv) any other implementations if it is the case.

Thirdly, the variables are at the next level of refinement in the competence tree. They detail even more several aspects of specific competences. For example in the specific competence of an array definition we will have the following variables: i) what is the textual definition; ii) what is the base type; iii) what is the index.

Fourthly, the actions model the basic generative learning units which are presented in details in the next chapter. We consider that the hierarchical structure of the competence tree allows a good classification of the generative actions and thus enable the development of actions to be focused on detailed aspects of algorithms. The nodes of the competence tree favor the creation of a dependency relationship between concepts at several levels of detail. In our approach GLOs may be considered the subtrees rooted by general competences since they model a whole concept if the concept is considered a learning objective. GLOs can be set also on lower level nodes depending on the learning objective of each LO designer. It depends what the LO designer considers to be a learning objective in his current use case. For example, a software project manager must know only the basic aspects of an algorithm and not all the technical details of its implementations, while a software developer and a software tester must know all aspects.

The presence of the tree nodes offers the flexibility of enabling and disabling them for different learning purposes. The competence tree in the current prototype is represented in the XML markup language but it can be easily imported in a relational database using a representation based on a reference to parent. They can be stored also in XML format in a larger text field but it need to be parsed each time it is used. Each competence tree node has name and description properties expressed also as XML nodes. The competence tree nodes are a good support for attaching learning records in order to compute assessment results. Variable nodes have links towards actions which are represented in their own XML file for easier adaptability and interoperability. Thus, the delivery of a GLO in our approach is possible in the form of: i) XML files representing the competence tree structure; ii) XML files for actions as generative learning units; iii) a container for all the XML representations in the form of folder or of a ZIP file.

IV. GENERATIVE MODELS FOR DATA STRUCTURE DISCIPLINES

In this chapter we present the structure of actions which embeds generative models. The structure of an action consists in: i) scenario section where we define all the symbols used in the models with their types, initialization formulas, functions etc.; ii) theory section that contains demonstrations, comparisons, explanations, details about the learning objective; iii) question section that contains the description if an exercise; iv) answer section that contains the answering possibilities of the learner like: single, choice, multiple choice, short answer, multiple answers etc. but not it is not limitative; v) feedback section that contains either corrective explanations or encouraging and motivating messages for the learner.

It is not necessary to present each section on one frame to the learner in the learning environment. There can be added or removed sections as it is suitable to the learning objective, except for the first one which has the role of instantiating the symbols or parameters that express the variability of the LO content and also of its behavior.

The main features of the generative models of our approach are: i) all sections may use generative models if the GLO designer considers so; ii) the instantiation is based on random numbers; iii) the instantiation is fully automatic; iv) the assessment of the response correctness is fully automatic. In the next chapter we will explain on a concrete example how we obtained these GLO features and what are the challenges and limitations.

Particularly, for data structures disciplines and namely for the learning of algorithms we consider that adding extra expressivity based on XPath and XQuery XML technologies is a must in order to build complex GLOs that are able to focus on intimate details of algorithm running steps that we consider important for programming.

The implementation of such a GLO is based on the current web technologies like JavaScript, JSON, XML. JavaScript functions applied on symbols referring simple variables and arrays offer many possibilities to express variability of GLOs. The functional and dynamic nature of the JavaScript programming language helps with expressivity. Last, but not least the spreading of JavaScript in all popular Internet browsers makes the GLOs portable and deliverable on all platforms. We used XML markup language as support for the generative model because we consider that its syntax is more simple and intuitive than the syntax of JSON representation.

V. USE CASE EXAMPLE

In this chapter we present a use case example GLO. We will focus on its semantics and behavior. For the example we selected a GLO for the learning of the insertion sorting algorithm.

```
<scenario>
        <text>We temper with the comparison signs located in for and while.</text>
        <symbol name="bCorrect1" tip="int">random(0,1)==1 ? true : false;</symbol>
        <symbol name="bCorrect2" tip="int">random(0,1)==1 ? true : false;</symbol>
        <symbol name="bCorrect3" tip="int">random(0,1)==1 ? true : false;</symbol>
        <symbol name="bCorrect" tip="int">
                v("bCorrect1") && v("bCorrect2") && v("bCorrect3");</symbol>
        <symbol name="bIncorrect" tip="int">!v("bCorrect")</symbol>
        <symbol name="nIndex1" tip="int">random(0,2);</symbol>
        <symbol name="nIndex2" tip="int">random(0,2);</symbol>
        <symbol name="nIndex3" tip="int">random(0,2);</symbol>
        <symbol name="tab1" tip="array">["&lt;=","&gt;","&gt;="]</symbol>
        <symbol name="tab2" tip="array">["&lt;","&lt;=","&gt;="]</symbol>
        <symbol name="tab3" tip="array">["&lt;","&lt;=","&gt;"]</symbol>
        <symbol name="op1g" tip="string">v("tab1",v("nIndex1"));</symbol>
        <symbol name="op2g" tip="string">v("tab2",v("nIndex2"));</symbol>
        <symbol name="op3g" tip="string">v("tab3",v("nIndex3"));</symbol>
        <symbol name="op1" tip="string">v("bCorrect1") ? "&lt;" : v("op1g");</symbol>
        <symbol name="op2" tip="string">v("bCorrect2") ? "&gt;" : v("op2g");</symbol>
        <symbol name="op3" tip="string">v("bCorrect3") ? "&gt;=" : v("op3g");</symbol>
</scenario>
```

Figure 1. Scenario example

In Figure 1 we present the scenario section which is the most complex part of this example GLO. The first element is a text description of the scenario. The first three defined variables are of type Boolean and each is made for one comparison operator from the algorithm that we want to temper with. The bCorrect variable denotes the state where the original algorithm is left intact with no operators tempered. The bIncorrect variable denotes the tempered state of the algorithm.

The next tree variables represent indexes for tree element arrays where the wrong comparison operators are stored for each location of the algorithm. The next tree variables denote arrays with the wrong comparison operators, while the next three variables denote random elements from these arrays.

Finally, the operator values are computed taking into account the values of the bCorrect1, bCorrect2, bCorrect3 symbols, the correct operators and the wrong operators.

```
<sentence>
        <text>
        The following algorithm is given:
        int i,j;
        int x;
        for(i=1;i<value name="op1"/>n;i++)
        {
                x=tab[i];
                j=i-1;

                while(tab[j]<value name="op2"/>x &amp;&amp; j<value name="op3"/>0)
                {
                        tab[j+1]=tab[j];
                        j=j-1;
                }

                tab[j+1]=x;
        }
        Identify whether the algorithm is or not a sorting by insertion one.
        </text>
</sentence>
```

Figure 2. Sentence example

In Figure 2 we present the template of the algorithm embedded in the sentence section of the GLO. Each operator may be replaced the right one or with the wrong one depending on the randomly generated values in the scenario section.

```
<answers>
        <answer id="1">
                <text>
                        Yes, it is!
                </text>
                <correct><value name="bCorrect"/></correct>
        </answer>
        <answer id="2">
                <text>
                        No, it is not!
                </text>
                <correct><value name="bIncorrect"/></correct>
        </answer>
</answers>
```

Figure 3. Answers example

In Figure 3 we present the answers example from which the learner has to choose or fulfill. The bCorrect and bIncorrect variables are used in order to help the generated LO interpreter to assess automatically the learners answer.

```
<feedbacks>
        <feedback id="1" answerid="[1]">
                <text>
```

```
                        Yes, the given algorithm is a in insertion sorting one.
                        All comparison operators are in place.
                </text>
        </feedback>

        <feedback id="2" answerid="[2]">
                <text>
                        Not, it is not because:
                        <text visible="!v(bCorrect1)">
                                <![CDATA[- The < operator located in the for cycle
                                was replaced with the ]]> <value name="op1g"/> operator.
                        </text>
                        <text visible="!v(bCorrect2)">
                                <![CDATA[- The > operator from the while cycle
                                namely from condition tab[j]>x was replaced with the ]]>
                                <value name="op2g"/> operator.
                        </text>
                        <text visible="!v(bCorrect3)">
                                <![CDATA[- The >= operator from the while cycle
                                namely from condition j>=0 a was replaced with the ]]>
                                <value name="op3g"/> operator.
                        </text>
                </text>
        </feedback>
</feedbacks>
```
Figure 4. Feedbacks example

In Figure 4 we present the feedbacks section. Each feedback is activated by the learner selected answer. The link is set through the answered attribute which contains an array of answer indexes. In our example we user only one index. The first feedback has a static text message, while the second one has a dynamic one depending on the generated values. For each tempered location in the sorting algorithm we will show a message explaining the comparison operator switch that we did, which is the wrong value and why the algorithm will not work.

VI. CONCLUSIONS AND PERSPECTIVES

In this paper we discussed about finding expressive learning models that: i) are able to fine grain model the subject domain, namely computer programming; ii) may offer divers reuse facilities based on helping software tools; iii) are simple enough to be easily modified with less programming knowledge; iv) its concrete instances generated from the model will increase the performance of the learning process.

The approached issue is important from the scientific point of view because its solving helps in increasing tutor expressivity in developing e-learning materials. It is also important from the economical point of view because the exercise made for solving it drives to the development of ITC specialists thus increasing the companies' projects acceptance rate in the field. The approached issue of GLOs is important from the social point of view because it is created a new advanced learning tool which can help qualification of unemployed, the specialization of those in the working field. The proposed step is aligned to other international projects or programs like Hour of Code.

As perspective we intend to investigate the use of XML technologies to extend our generative models with specific capabilities to be able to build fine grained models to target different learning aspects of algorithms. Another perspective for GLOs is that they need to be integrated in LMSs together with their business logic for a faster widespread of the concept.

Acknowledgements

This paper is supported by the Sectorial Operational Program 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.".

References

[1] Advanced Distributed Learning Network - Sharable Content Object Reference Model, http://www.adlnet.gov, 2015.

[2] C. Alexander, S. Ishikawa, M. Silverstein - A pattern language: Towns, buildings, construction. Oxford: Oxford University Press, 1977.

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

[4] Tom Boyle. The design and development of second generation learning objects. Invited talk at Ed Media 2006, World Conference on Educational Multimedia, Hypermedia and Telecomunications, Orlando, Florida, June 28, 2006.

[5] Tom Boyle, Claire Bradley - User Guide for the GLO Maker 2 Authoring Tool http://www.glomaker.org, 2009.

[6] Renata Burbaite, Kristina Bespalova, Robertas Damasevicius, Vyutatas Stuikys - Context Aware Generative Learning Objects for Teaching Computer Science, International Journal of Engineering Education, vol. 30, no. 4, pp. 929-936, 2014.

[7] Ciprian-Bogdan Chirila - 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, 2013.

[8] Ciprian-Bogdan Chirila - 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, 2014.

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

[10] Robertas Damasevicius, Vyutatas Stuikys - 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.

[11] Dublin Core Metadata Initiative. The Dublin Core metadata element set, ISO 15836:2009 / Cor 1:2009, 2009.

[12] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Design Patterns: Elements of Reusable Object-Oriented Software, Pearson Education, 2004.

[13] Peng Han, Bernd J. Kraemer - Generating Interactive Learning Objects from Configurable Samples. In proceedings of International Conference on Mobile, Hybrid and On-line Learning, 2009.

[14] Catalin Iapa - Outstanding research in MOOC and future development. In proceedings of eLSE 2014 International Scientific Conference eLearning and Software Education, Bucharest, Romania, April, 2014.

[15] Augustin-Catalin Iapa - MOOC and Learning Analytics: interaction and evolution, SMART 2014 - Social Media in Academia: Research and Teaching, Medimond Publishing House, Bologna, Italy, ISBN 978-88-7587-712-5, Timisoara, Romania, September, 2014.

[16] Ray Jones, Tom Boyle - Learning Object Patterns for Programming. Interdisciplinary Journal of Knowledge and Learning Objects, vol. 3, 2007.

[17] IEEE Learning Technology Standards Committee - LOM working draft v4.1 Available: http://ltsc.ieee.org/doc/wg12/LOMv4.1.htm, 2000.

[18] James D. Oldfield - An implementation of the generative learning object model in accounting. In proceedings of Asciilite, Melbourne, 2008.

[19] Rustici Software - Tin Can API, www.tincanapi.com, 2015.

[20] Vyutatas Stuikys, Robertas Damasevicius - Towards Knowledge-Based Generative Learning Objects, Information Technology and Control, vol. 36, no. 2, ISSN 1392-124X, 2007.

[21] Vyutatas Stuikys, Robertas Damasevicius - 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 Matematics and Informatics, Vilnius, 2008.

[22] Vyutatas Stuikys, Ilona Brauklyte - 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, 2009.

[23] Vyutatas Stuikys, Renata Burbaite, Robertas Damasevicius - 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, 2013.

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