Metamodels for Auto-Generative Learning Objects Dedicated to Unix Operating System Disciplines

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

Gaultier Parain
University of Lille
Lille, France
E-mail: gaultier.parain@gmail.com

***Abstract*—The currently developing industry demands more and more IT specialists day by day. Many industrial systems are automated using servers based on Unix and Linux operating systems in industries like: automotive, telecommunications, production etc. Universities and other teaching organizations tend to lack human resources since most of the potential tutors are already employed in the IT industry having significant beneficial packages. In this context cloud enabled auto-generative learning objects can help in the process of training of new IT specialists. They offer the possibility to reuse learning patterns and to fill them with randomly generated but controlled data. Thus, students can learn, exercise and auto-assess their knowledge in Unix commands to be ready for employment in the IT industry.**

***Keywords*—generative learning objects, auto-generative learning objects, Unix / Linux operating systems, cloud applications**

I. Introduction

The current industrial revolution entitled Cyber Physical Systems relies heavily on computing machines. Companies were built and people were hired in this sense, to provide IT services to the growing industry: plants, satellites, communication networks etc. With the widespread and growing bandwidth of the Internet the possibility of providing these IT support services from remote increased. For example, IT specialists from Romania are able to provide IT services to industrial businesses from all over the world through the Internet. The services provided by the IT companies rely on the good knowledge of the Unix and Linux operating systems commands. Because the IT services business is growing fast companies in the field re-target graduates from technical faculties from domains, like constructions - where the employment rate is not very high, towards the IT services industry. Universities and companies from our region lack tutors because most of them went in the industrial IT sector where the beneficial packages and consistent. For example, in some universities the workload of each tutor is doubled in order to be able to teach the high number of incoming students demanded by the IT industry.

Generative learning objects (GLOs) are reusable templates that can be filled with content having a clear learning objective [2]. The content can be written manually or generated through meta-programming. Auto-generative learning objects (AGLOs) are reusable templates that can be instantiated with random data in order to generate learning content [4], [5]. The difference between GLOs and AGLOs is the fact that AGLOs: i) have automatic instantiation in the context of our developed framework; ii) use random values in the instantiation process, the values are generated using random seeds but they are controlled through Mathematical functions.

Students can use AGLOs for learning, training and assessment on a plethora of devices: computers, laptops, tablets or smartphones etc. in any place they have a minimal speed Internet connection and at any time. The use of auto-generative learning objects accelerates the process of learning and training for the candidate employees in the IT services field.

In this paper we present a set of metamodels which follow a reusable pattern in the generation of AGLOs usable in learning operating systems.

The paper is structured as follows. In section II we present recent related works in the field of learning objects and generative learning objects. Section III presents the meta-model of auto-generative learning objects. Section IV presents the potential of the model in writing operating systems learning objects. Section V draws the conclusions and sets up the future work.

II. Related Works

Generative Learning Objects are considered to be second generation learning objects consisting in reusable patterns to be filled with content according to a specific learning objective [2]. The content can be added manually or generate using meta-programming techniques.

In [3] is presented a way of generating GLOs from feature diagrams and meta-programming. The resulted GLOs have as learning objectives several programing language concepts taught using Lego robots.

In [9] are presented GLOs that integrate not only software components but also hardware ones, namely educational robots, in their content. The designed GLOs were used in a project based university course resulting an assimilation model of theoretical knowledge from the robotics project implementation.

The AGLO model was applied in several domains:

978-1-5386-4444-7/18/$31.00 ©2018 IEEE

- algorithm analysis and design, [4] presents ideas about how to create AGLOs for learning Prim and Krushkal graph algorithms;
- data structures and algorithms disciplines [5];
- primary school Arithmetic, [7] presents faction operations for primary school students;
- operating systems, [8] presents the first steps towards the development of AGLOs in the IT services sector.

On the other hand, since our paper is about Unix operating systems we inspired ourselves from several manuals like: [14], [1], [13] of which we have knowledge that are used in local IT services companies for job interviews.

III. Auto-Generative Learning Objects

AGLOs are based on the creation of reusable patterns of static text and figures and dynamic values and figures or animations that can be combined freely in order to reach a planned learning objective.

The structure of the AGLO is presented in Fig. 1:

```
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>" ("integer"|"string"|"fraction")
   "</type>"
08 Expression ::= "<expr>" Function
   "(" ExpressionList ")" "</expr>"
09 Function ::= (element from functions and
   operators of JavaScript using random numbers)
10 ExpressionList ::= Expression (, Expression)*
11 Theory ::= "<theory>" (ID)* "</theory>"
12 Question ::= "<question>"(ID| Value)*
   "</question>"
13 Value ::= "<value>" "<name>" ID "</name>"
   "</value>"
14 Answers ::= "<answer>" (Answer)+ "</answer>"
15 Answer ::= "<answer>" "<id>" INTEGER_LITERAL
   "</id>" (ID|Value)* Correctness "</answer>"
16 Correctness  ::= "<correct>" ("true" |"false")
   "</correct>"
17 Feedbacks ::= "<feedbacks>" (Feedback)
   "</feedbacks>"
18 Feedback ::= "<feedback>" (ID)* "</feedback>"
```

Fig. 1. AGLO Structure

Diving into the AGLO details we designed the following structure.

Line 01 presents the main elements of an AGLO:

- name - the place where AGLO identification data is stored;
- scenario - the place where the symbols are defined;
- theory - the place designed to present theoretical information about the learning objective;
- question - the place designed for the question to be built;
- answers - the place designed for the one or multiple answers the student has to give of to choose from;
- feedback - the place where contextual feedback is built for the student.

Line 02 describes the name element of the AGLO used for identification.

Lines 03-10 describe the scenario section. Line 03 describes the scenario as a repetition of symbols accompanied by comments. The comments are used to express in natural language the role of each symbol and how the entire AGLO works. Line 04 describes the comment as a sequence of identifiers and numbers. Line 05 defines the symbol as having three elements:

- name - an identifier to refer later the symbol;
- type - a keyword defined in line 07, the role of the type is informal, it is used only by the AGLO designer;
- initialization expression - is an expression formed out of JavaScript functions and operators composition (see lines 08-10).

Line 11 describes the theory element which is formed of static text for the student to read before performing the exercise proposed by the current AGLO. Lines 12-13 describe the question element where static texts denoted by IDs and symbol values can be found. Symbol values can have text values or numeric values or they can be other complex structures like: HTML tables or SVG (Scalable Vector Graphics) graphs etc. Combining text and symbol values the AGLO will generate at each runtime different content, namely different questions with corresponding answers.

Lines 14-16 present the structure of answers which can be unique or multiple. When the answer is unique the student will have to fill it in a text field. When the answers are multiple then the student will see a set of radio boxes or a set of check boxes depending on the number of true answers.

Lines 17-18 present the feedback elements. In practice it is very difficult to conceive such feedback models so a general static text conclusion should suffice.

A. AGLO Design

The design of an AGLO implies several actions as presented in Fig. 2.

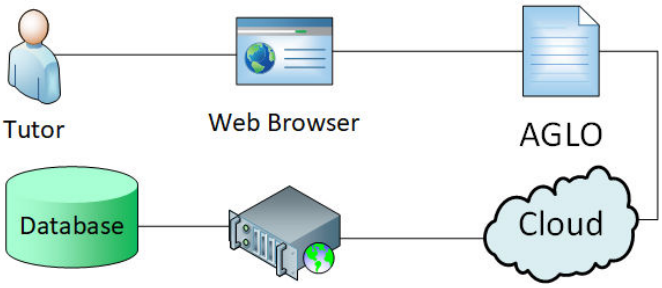

Fig. 2. AGLO Creation

The tutor uses a web browser to edit the AGLO as an XML file mixed with JavaScript expressions including calls to domain specific libraries constructors and methods. The AGLO is stored in a cloud environment which relies on a web server and a database server. The stored AGLOs are then

available to the students organized as a competence tree on multiple competence levels.

Diving into detail, the conceptual design steps of an AGLO are as follows:

- to conceive or reuse a concrete exercise from student assessment tests;
- to write down the solution steps of the exercise;
- to abstract the functional model the exercise, namely formulas of how variables depend functionally on each other;
- to select the function variables or symbols that will be initialized with random values;
- to write the symbols initialization function based on the random number generator, in this point domain specific JavaScript [11] libraries may be used or must be written;
- to write the theory section which is designed to be static text, formulas and images;
- to write the question pattern section where static text is combined with symbols;
- to write the answer pattern, it can be single text field or multiple choice, where correct answer computing symbols must be used;
- to conceive and write the feedback pattern using static text and symbols in order to provide some insight to the student for the current exercise.

B. AGLO Instantiation and Usage

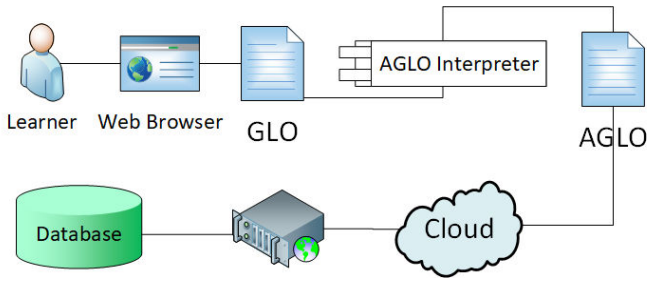

Fig. 3. AGLO Usage

In Fig. 3 is presented the instantiation and usage of an AGLO. The student accesses the AGLO repository through a web browser. Next, the AGLO Interpreter parses the AGLO expressed as XML and generates the corresponding concrete GLO. The AGLO generic pattern is filled with the effective values of the symbols which were initialized with the result of the evaluated JavaScript expressions. In this point the expressions may use domain specific libraries.

For example, if we want to test a Unix command related to files and folders then we might need to generate a random folder tree and pick randomly from it one or two nodes in order to use them in the context of the given command. For the generation of the folder tree a JavaScript class instantiation can be used and to select two nodes from the tree we can use dedicated methods in this sense.

At least one symbol has to compute the correct solution for the given problem in order to enable the automatic verification of the answer. Sometimes symbols must be used to compute distractor variants for the situation in which the multiple answer schema is used.

AGLOs can also be used in contexts where there is no computed solution. Such approaches are recommended for the generation of printed tests that are to be corrected manually by the tutor.

The representation of the tree can be a simple list of indented lines in an HTML `<pre>` or it can be an SVG representation of a diagram like figure using lines for the edges, circles for the nodes and texts for the labels. Such a representation can be used in the text of the question in order to create the particular instance of the problem.

Further on, the student reads the question which states the task that he has to solve. The student has to do some reasoning in order to produce the answer. Depending on the complexity of the task the student might need to use a paper and a pen to complete the necessary calculations. Next, the computed answer is written in a text field or selected from the displayed choices.

In the last step, the AGLO framework compares the two answers the computer computed one with the student computed one. If the multiple answer schema was used the it checks that the correct one was selected. The result is stored using the xAPI format in the cloud infrastructure [10], [6].

After the response fulfillment a feedback is displayed to the student as a conclusion of the exercise. Where possible, the feedback can be used:

- to show the concrete reasoning steps in detail to get the solution;
- to show the general steps with no specific details at all;
- to show the selected distractor details why is not the correct answer;
- to show the selected distractor general principle so the student will understand fully the learning objective and not fall again in the distractors trap.

IV. Unix Commands Auto-Generative Models

The first attempt to imagine and abstract Linux commands exercises is presented in [8]. There were imagined simple uses of several Unix commands. Each AGLO is intended for teaching and training one learning objective, namely one aspect in using a particular command. The commands that we considered in our experiment are all dealing with files and directories or folders. In order to present our generative models we will use the `cat` command which concatenates several files given as arguments to the standard output.

A. Example of Files Concatenation Located in the Current Directory

The first AGLO example is a simple one, it will train the student to use the Unix `cat` command and it serves also for presenting the AGLO infrastructure. In this example we want to test that the student knows how to write the command with two given arguments, representing files in the current working directory.

```
<scenario>
 <text>Generating two file names randomly.</text>
 <symbol name="file1" type="Folder">
  new Folder(0,null);</symbol>
 <symbol name="file2" type="Folder">
  new Folder(1,null);</symbol>
 <symbol name="sfile1" type="string">
  v("file1").toString()+".txt";</symbol>
 <symbol name="sfile2" type="string">
  v("file2").toString()+".txt";</symbol>
 <symbol name="command" type="string">
  "cat "+v("sfile1")+" "+v("sfile2");</symbol>
</scenario>
```

Fig. 4. Example 1 Scenario

In Fig. 4 we present the scenario section of the first AGLO example. As mentioned in section III the scenario consists in a set of symbols to be used in the question and answers sections.

The first two symbols we design, named ”file1” and ”file2”, are two objects of type ”Folder”. This type is a JavaScript class from the operating systems domain library that we created. The ”Folder” class models a folder from the Unix file system and consists in: i) an integer identifier corresponding virtually to an i-node; ii) an integer index referring the parent node; iii) an integer index referring the first child node; iv) an integer index referring the right sibling node; v) a string denoting the name of the folder. In this example we will use only the identifier and the name of the folder. The class is equipped with a ”setRandomName()” method which uses a predefined array of names concatenated with random numbers from 10 to 99 in order to generate random but distinct names.

Next, we create two symbols ”sfile1” and ”sfile2”, denoting string representations of the previously created objects and also concatenating the ”.txt” extension in order to look like regular text files.

Then, we compute the correct command for the student to write using the previously defined symbols, consisting in the name of the command and the two arguments concatenated all together. In order to access the symbols we use a lookup function named v(”name”) which returns the symbol’s value identified by ”name”. This syntactical inconvenience should be eliminated in the next version of the AGLO interpreter using an additional preprocessing phase.

In Fig. 5 we present the next infrastructure of our AGLO. The question is composed out of static text and symbol values referred using the ”value” XML element. Replacing the symbol references with different values we obtain different instances of the same exercise.

In Fig. 6 we see the output of an instantiation: the question and the computed answer expected to be written by the student.

B. Example of Files Concatenation Referring Relative Paths

In this example we exercise the same `cat` command using relative paths.

In Fig. 7 we present a more complicated example where a student must concatenate two files from a directory tree given the fact that his current working directory is in a third location.

```
<question>
 <text>
  Use the cat command to concatenate
  the contents of the file <value name="sfile1"/>
  and the contents of the file
<value name="sfile2"/>. The two files are
located in the working directory.
 </text>
</question>
<answers>
 <answer id="1">
<text>
 <value name="command"/>
</text>
 </answer>
</answers>
```

Fig. 5. Example 1 Question and Answer

```
Use the cat command to concatenate
the contents of the file Alpha07.txt and
the contents of the file Bravo10.txt.
The two files are located in the working
directory.

cat Alpha07.txt Bravo10.txt
```

Fig. 6. Example 1 Output

```
<scenario>
<symbol name="n" type="integer">random(5,8,0);
</symbol>

<symbol name="ft" type="FolderTree">
new FolderTree({"nNoOfNodes" : v("n")});</symbol>
<symbol name="st" type="string">
v("ft").toString();</symbol>

<symbol name="folders" type="Array">
v("ft").selectRandomFolders(3);</symbol>

<symbol name="wd" type="Folder">
v("folders")[0];</symbol>
<symbol name="dir1" type="Folder">
v("folders")[1];</symbol>
<symbol name="dir2" type="Folder">
v("folders")[2];</symbol>

<symbol name="file1" type="Folder">
new Folder(0,null);</symbol>
<symbol name="file2" type="Folder">
new Folder(1,null);</symbol>

<symbol name="pathDir1" type="RelativePath">
v("ft").path(v("wd"),v("dir1"));</symbol>
<symbol name="pathDir2" type="RelativePath">
v("ft").path(v("wd"),v("dir2"));</symbol>

<symbol name="command" type="string">
"cat "+v("pathDir1")+"/"+v("file1").name+" "+
v("pathDir2")+"/"+v("file2");</symbol>
</scenario>
```

Fig. 7. Example 2 Scenario

The student has to determine the relative paths in order to refer the files.

For such a scenario a randomly generated folder tree is needed. In this sense we instantiated the ”FolderTree” class,

in ”ft” symbol, which builds a consistent directory tree having ”n” nodes of random names. The number of nodes for the tree is between 5 and 8 and it is denoted by symbol ”n”.

Next, the ”ft” textual representation is computed into symbol ”st”.

From the computed folder tree we extract randomly 3 folders using the method ”selectRandomFolders(3)”.

The first one will play the role of current working directory denoted by symbol ”wd”.

The other two will play the role of the files to be concatenated denoted by symbols ”dir1” and ”dir2”.

Then, we create two file names ”file1” and respectively ”file2” that will be located in ”dir1” and respectively in ”dir2”.

Next we use the FolderTree method named ”path(cwd,path1)”, which computes the relative path of ”path1” relative to the current working directory ”cwd”. This method is applied for both ”dir1” and ”dir2”.

Finally, the concatenation command is assembled from the computed symbols.

```
Use the cat command to concatenate the
contents of the file Alpha28 and the
contents of the file November40. You
are in the directory Romeo20, the file
Alpha28 is in the directory Mike43 and
the file November40 is in the directory
Delta58.
The directory hierarchy is in the following
figure:
Foxtrot12
*Golf99
*Delta58
*Romeo99
**Sierra40
**Mike43
***Romeo20

cat ../../../Romeo99/Mike43/Alpha28
../../../Delta58/November40
```

Fig. 8. Example 2 Output

In Fig. 8 we present the output of the instantiated AGLO from second example. Since we are located in directory ”Romeo20” we need to descend three levels to ”Foxtrot12” and then to access ”Romeo99” and finally ”Mike43” where file ”Alpha28” is located. For the second file we need to descend the same three levels and to access ”Delta58” where file ”November40” is located

C. Generative Models for Other Unix Commands

The presented ideas can be extended to other Unix commands working with files. Several scenarios are imagined as follows. The change mode `chmod` command may use immediate, absolute and relative files. Additionally, the use of i) permission rights ”r”, ”w”, ”x”; ii) targets ”u”, ”g”, ”o”, ”a”; iii) operators ”+”, ”-”, ”=” can be combined and exercised.

Similarly, copy `cp`, the move `mv`, the link `ln`, remove `rm`, list `ls` commands can be exercised on the following dimensions: i) files and folders; ii) immediate, absolute and relative paths; iii) name patterns.

D. Prototype Implementation

The implementation of the prototype is cloud based. The AGLOs are expressed as XML text files and are stored in the cloud. The AGLO interpreter is written entirely in JavaScript [11] using jQuery library [12] and runs on the client side in common web browsers. On the server side we find PHP code for managing the student responses using xAPI model [10], [6].

V. Conclusions and Perspectives

We conclude that it is possible to design and write AGLO models where two different aspects were composed: i) commands on one hand denoting the action and ii) files and folders on the other hand denoting the target objects.

The complexity of the AGLOs is somehow hidden in the JavaScript domain libraries. Having powerful and well documented domain specific libraries enables the creation of AGLOs.

The AGLO XML format is simple and flexible allowing non-programmers to adjust the presentation, e.g. they can reuse the AGLO by translating the text in a different language.

As future work we intend to develop more complex Unix scenarios in the sector of disks, partitions, volumes and towards batch commands, shell scripts which are highly used in the industry.

Another future work is to create a visual interface for editing static and dynamic content with 2-3 panels where different instantiations are visible.

References

[1] Wander Boessenkool, Bruce Wolfe, Scott McBrien, George Hacker, and Chen Chang. *Red Hat System Administration II Student Workbook*. RedHat, Inc., 2014.

[2] Tom Boyle. The design and development of second generation learning objects. In *World Conference on Educational Multimedia, Hypermedia & Telecommunications*, Orlando, Florida, June 28 2006.

[3] Renata Burbaite, Kristina Bespalova, Robertas Damasevicius, and Vytautas Stuikys. Context-aware generative learning objects for teaching computer science. *International Journal of Engineering Education*, 30(4):929–936, 2014.

[4] Ciprian-Bogdan Chirila. Auto-generative learning objects for it disciplines. In *Proceedings of the International Conference on Virtual Learning 2015*, pages 1–6, Bucharest, Romania, October 2015. University of Bucharest, Romania.

[5] Ciprian-Bogdan Chirila. Auto-generative learning objects in online assessment of data structures disciplines. *BRAIN - Broad Research in Artificial Intelligence and Neuroscience*, 8(1):24–34, April 2017.

[6] Ciprian-Bogdan Chirila. Towards the enhancement of aglos with scorm and xapi. In *Proceedings of the 13-th International Scientific Conference eLearning and Software for Education*, pages 1–7, Bucharest, Romania, April 2017.

[7] Felicia-Mirabela Costea, Ciprian-Bogdan Chirila, and Vladimir Creţu. Auto-generative learning objects for middle school arithmetic. In *Proceedings of the 14-th International Scientific Conference eLearning and Software for Education*, pages 1–8, Bucharest, Romania, April 2018.

[8] Felicia-Mirabela Costea, Ciprian-Bogdan Chirila, and Vladimir Creţu. Towards auto-generative learning objects for industrial it services. In *Proceedings of the IEEE 12-th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 1–5, Timisoara, Romania, May 2018.

[9] Robertas Damasevicius, Lina Narbutaite, Ignas Plauska, and Tomas Blazauskas. Advances in the use of educational robots in project-based teaching. *TEM Journal - Technology Education Management Informatics*, 6(2):342–348, May 2017.

[10] Advanced Distributed Learning Initiative. Experience api (xapi). http://adlnet.gov/adl-research/performance-tracking-analysis/experience-api/, 2017.
[11] ECMA International. Standard ecma-262 ecmascript 2016 language specification. http://www.ecma-international.org/publications/standards/Ecma-262.htm, 2016.
[12] The jQuery Foundation. jquery project. https://www.jquery.com, 2018.
[13] Susan Lauber, Philip Sweany, Rudolf Kastl, and George Hacker. *Red Hat System Administration I Student Workbook*. RedHat, Inc., 2014.
[14] Eric Steven Raymond. *The Art of Unix Programming*. Addison-Wesley, 2003.