Towards Auto-Generative Learning Objects for Industrial IT Services

Felicia-Mirabela Costea*
Ciprian-Bogdan Chirila*
Vladimir-Ioan Creţu*
*University Politehnica of Timişoara, Romania
Department of Computers and Information Technology
E-mail: mirabela.costea@cs.upt.ro; chirila@cs.upt.ro; vcretu@cs.upt.ro

***Abstract*—In eastern Europe, in the last few years the IT services industry has significantly grown due to the externalization of IT operations from western Europe and America: production plants, communications, satellites, etc. In this context the pressure on the eastern IT services labor market has increased. IT services companies employed students with no IT background and trained them in their in house academies in order to respond fast to their contractual obligations. The time frame to form IT operations specialists is very narrow so they need to think of new strategies to speedup their training. In this sense auto-generative learning objects as the second generation of learning objects are a potential solution since it allows autonomous learning anytime during the day, when queuing, in the subway or during school or job breaks. Learners are trained with automatically generated dynamic content, get automatic evaluation of their responses and receive dynamic feedback. Thus, they can exercise autonomously different learning situations benefiting from automatic evaluation.**

I. Introduction

In the current economical context the global Industry 4.0 trend is to automate as much as possible all processes, to control production lines from remote using computers and to exchange data in manufacturing technologies.

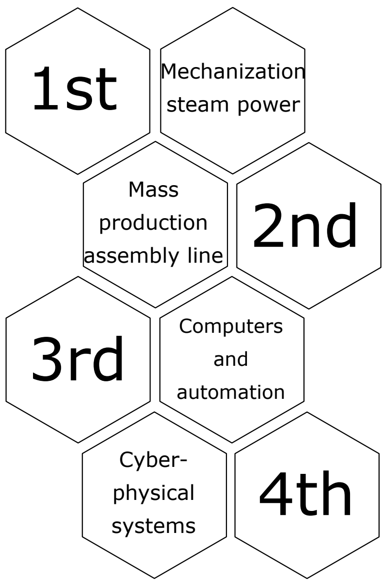


Fig. 1. Industrial Revolutions

In Figure 1 are reviewed briefly the main phases of the industrial revolution. The first industrial revolution happened with the advent of mechanization based on water and steam power. The second industrial revolution happened when the mass production begun using assembly lines and electricity. The third industrial revolution was in the age of computers and automation. The fourth industrial revolution is currently in progress and it is based on cyber-physical systems, Internet of things and cognitive computing. Several businesses outsources their IT services to specialized departments from eastern European countries. Well trained IT specialists are hard to find on the labor market so companies use students and train persons with non IT background just to fill in the qualified human resource crisis.

The industrial equipments rely heavily on computers grouped in clusters e.g. high availability clusters, which are driven by several operating systems. Among these the Unix and Linux based operating systems are very common so we decided to focus on their learning. It is known that Microsoft® family operating systems are frequently used by non-specialists and it is much easier to accommodate with their server versions. Although Unix and Linux platforms graphical user interfaces (GUIs) have been very well developed in the last decade still have few users. An exception in this sense is the use of Android operating system, which is based heavily on Linux, on most of the mobile devices but the access is limited to the GUI.

For example, the `www.w3schools.com` web site counted in the month of December 2017 its browsers operating systems and the results are:

- 76.6% Microsoft® Windows (10, 8, 7, Vista, XP);
- 7.9% Mobile;
- 9.8% Mac;
- 5.5% Linux;
- 0.3% Chrome OS.

It is clear that autodidact learners of that website do not use much Unix / Linux based operating systems in their everyday life.

In this context a training technology for Unix learners based on auto-generative learning objects (AGLO) is suitable because of its features. AGLOs are reusable didactic patterns that can be instantiated and are considered to be the second generation learning objects [2]. Previous uses of AGLOs are in middle school Arithmetic [6] and in data structures disciplines [4], [5].

AGLOs can be instantiated any number of times given always a new exercise of the same type. AGLOs are based on random numbers so the exercise content is diverse. AGLOs have automatic evaluation so they are suitable for autodidact students.

The goal of this paper is to study the potential for AGLOs to be used for learning Unix / Linux based operating systems. For this study we propose the following strategy:

- to research Unix tests and materials that can be found in technical documentations;
- to abstract the idea of each test item or concept and to design an AGLO scenario;
- to implement the required functionality in JavaScript support libraries;
- to generate the consumable learning object using the AGLO online framework.

In academia the Unix / Linux operating system is studied in the context of Operating Systems lectures. Usually such lectures include a commands and shell-scripting section and a systems call section where system calls are exercised in C programs. In this paper we focus on the commands part which is the most used in IT service businesses.

The paper is structured as follows. Section II presents related works in the field of generative e-learning systems and Unix / Linux online tutorials and quizzes. Section III presents briefly the structure of AGLOs. Section IV presents ideas of how AGLOs can be built for the learning objectives of Unix / Linux operating systems. Section V presents a few examples of AGLO implementations in the content of Unix commands. Section VII concludes and sets the future work.

II. Related Works

Generative learning objects (GLOs) [2] are the second generation learning objects consisting in reusable patterns that can be filled with content according to learning objectives. The content can be manually written or generated through meta-programming.

In the vision of Damasevicius and Stuikys [3] GLOs are generated by feature diagrams and meta-programming. They experimented with GLOs on learning programing concepts using Lego robots.

[4], [5], [6] target AGLOs towards different disciplines from academia: data structures and algorithms, fraction arithmetic for primary and secondary school students. They use meta-models for the generation of content based on random numbers and domain specific reusable libraries.

[10] presents in detail key concepts of Unix operating system like: philosophy, history, modularity, textuality, multiprogramming, transparency, configuration, languages and tools. Such textbooks stand as references to industrial training materials in IT services.

[1], [9] are two RedHat® student workbook volumes on system administration that cover topics of community enterprise operating systems which are highly used in industry.

III. AGLOs in a Nutshell

In [5] is presented the AGLO model. One of the advantages of using this model is the variability given by the random instantiation of input data.

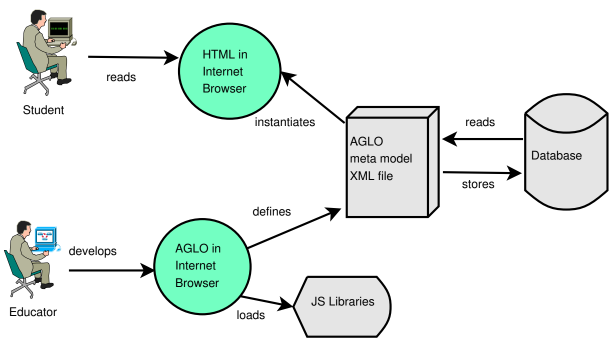


Fig. 2. The AGLO model

In Figure 2 we represented schematically the AGLO workflow. As you can see, the educator using a web browser and accessible environments develops a meta model. This model is saved in an XML file and stored in a database. Students interact with concrete learning object instances through the web browser. When an instance is created, meta-data is loaded from the database and methods from domain specific JS libraries are called.

The AGLO model has the following sections:

- name (line 02);
- scenario (lines 03-10);
- theory (line 11);
- question (line 12);
- answers (lines 14-16);
- feedbacks (lines 17-18).

Figure 3 presents this structure.

The six sections are composed of static text and also of dynamic values.

The first section is the AGLOs name.

The second section is the scenario where the variables are defined through JavaScript compatible expressions. Each variable is defined by a name, a type and its initialization expression. In this section there is also a brief description in natural language of what the exercise does. All the expressions use both predefined JavaScript functions and domain specific user defined functions.

The third section presents the theory which should be applied in the exercise, its content is made of static text only.

The fourth section consists of the question composed for the student. The question is based on the variables defined in the scenario, which makes it dynamic and attractive.

In the fifth section the student introduces the answer that is automatically evaluated.

```
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. 3. AGLO structure

The sixth and last section contains the exercise feedback which is also a combination of dynamic and static values.

We chose to develop this model because it offers students an interactive way of practicing and evaluating knowledge. Exercises are based on random instances that offer variability and diversity in the learning process.

IV. Unix Technical Documentation Analysis

In this section we will synthesize the Unix topics that were found in the learning materials used in IT services local companies. We organized our AGLOs in modules.

A. *AGLOs for Working with Directories*

In the first module we designed three exercises that tests commands for directories. These are useful to test the creation of directories or whole directory hierarchies. To build a hierarchy of directories, we used both the `-p` option and the top-down descending step by step method. Tree geometry and directory names are randomly generated, so for each instantiation the student has to create another hierarchy of folders.

B. *AGLOs for Working with Files*

In the second module we created several exercises to test the creation of a file and the population of an existing hierarchy with files. The directory hierarchy and the file names are randomly generated for the purpose of our exercises. In addition to the file creation commands, we also implemented other file commands. So we designed an AGLO to exercise the permissions changing for accessing a file, and one to exercise the mounting of a file system.

C. *AGLOs for Working with Processes*

In the third module we designed a few exercise ideas dealing with processes. The first exercise tests how a process - which displays data on the screen - is started in the background. In the second exercise we test how current user processes can be listed together with their extended information. In the next exercises the students test how to kill a process, and how to kill a process from the background. In the last exercise we test how to bring a process in the foreground.

D. *AGLOs for Working with Disk Partitions*

In the fourth module we designed exercises that test the creation of Linux ext3, ext4, swap partitions. The partition sizes are randomly generated and the student exercises the `fdisk` utility commands.

E. *AGLOs for Working with Packages*

In the fifth module we have exercises that relate to package management commands. Here the students can test how to install and uninstall program packages using `apt-get` and `yum` commands.

F. *AGLOs for Administrative Commands*

In the sixth module we designed some exercises that tests administrative Linux commands. In this module students can test commands for:

- creating a user;
- deleting a user;
- adding a user to an existing group;
- removing a user from an existing group;
- changing a user's password;
- getting information about a user.

These are simple atomic actions, not complicated to implement and has the main utility of exercising commands which are important in industrial environment. For diversification purposes the user names are generated using random values.

G. *AGLOs for Networking*

In the seventh module we have five AGLOs that exercise basic networking commands. We have created the following exercises:

- to find the user's hardware address;
- to delete a hardware address;
- to configure a new software address with a netmask;
- to manually activate or deactivate a network interface
- add or delete a route connected via a network interface.

We developed exercises that have the purpose to test and practice static commands, but also there are exercises that have random variables. In these exercises the addresses and the network interfaces are randomly generated so that each instance is different.

V. Case Study for Working with Unix Commands

In this section we will present a set of AGLOs dedicated to exercising the `mkdir` Linux command.

The first AGLO for `mkdir` Linux command is trivial. In its scenario the student is asked to write the command that creates a folder with a given name. The student has to type the `mkdir` Linux command followed by the folder name.

The second AGLO for `mkdir` Linux command exercises it's `-p` argument. The exercise implies the creation of full directory chains from the relative root to each terminal node. This exercise will be abstracted and designed as an AGLO while presenting its implementation in detail.

The third AGLO for `mkdir` Linux command is used in combination with the `cd` (change directory) command. The exercise is about creating folders one by one, incrementally, while visiting the directory hierarchy. This exercise has several valid visiting paths depending on the order of the visited nodes, thus making the automatic assessment challenging.

```
<scenario>
 <text>...</text>
 <symbol name="n" type="integer">random(5,8,0);
 </symbol>
 <symbol name="d" type="integer">random(2,3,0);
 </symbol>
 <symbol name="ft" type="FolferTree">
  new FolderTree({"nNoOfNodes" : v("n")});
 </symbol>
 <symbol name="st" type="string">
  v("ft").toString();
 </symbol>
 <symbol name="commmands" type="string">
  v("ft").toPathsString();
 </symbol>
</scenario>
```

Fig. 4. AGLO Scenario Example

In Figure 4 we present the AGLO scenario of our example. The scenario builds randomly a folder tree and contains symbols having the following semantics:

- `n` is the number of nodes or folders and is generated as a random value between 5 and 8;
- `d` is the degree of the tree (the maximum number of children a node may have) and is generated as a random number between 2 and 3;
- `ft` is a reference to the root of the generated tree through `FoldersTree` class instantiation with the previously defined parameters;
- `st` is the string representation of the generated tree (see Figure 5) to be displayed to the student;
- commands is a string containing the correct expected answer (see Figure 6) computed by a library method named `toPathsString()` from the FoldersTree class.

Figure 5 depicts the randomly generated tree. The generation is implemented in JavaScript [7] as a reusable function that create node objects taking random names augmented with two integer digits to avoid conflicts.

Figure 6 depicts the correct answer which is obtained by concatenating the `mkdir` command with every path of the folder tree. In order to obtain all the paths the tree is visited recursively from its root to each terminal node.

```
Create the folder hierarchy from the next
figure using mkdir command and -p option.
Quebec12
*Sierra69
*Charlie81
**November83
***November77
***Lima9
```

Fig. 5. AGLO Generated Question Example

```
mkdir -p Quebec12/Sierra69/
mkdir -p Quebec12/Charlie81/November83/November77/
mkdir -p Quebec12/Charlie81/November83/Lima9/
```

Fig. 6. AGLO Answer Example

VI. Prototype Implementation

The AGLOs are instantiated on the online platform written in JavaScript [7] and assisted by jQuery [8] library. The platform contains several modules:

- user interface module;
- AGLO parser module;
- table of symbols;
- AGLO content generator module;
- domain specific module;
- result persistence module;

The user interface module presents the AGLOs to the user in a competence tree. The AGLO parser module reads the XML elements and creates a set of JavaScript objects in the memory of the browser.

The table of symbols module stores all the symbols from the AGLO scenario section and initializes them with their expressions.

The content generator module generates the combination of dynamic and static content in HTML format to be consumed to the learner.

The domain specific module contains all the classes and methods with specific domain semantic, in our example the `FoldersTree` class with its `toPathsString()` method.

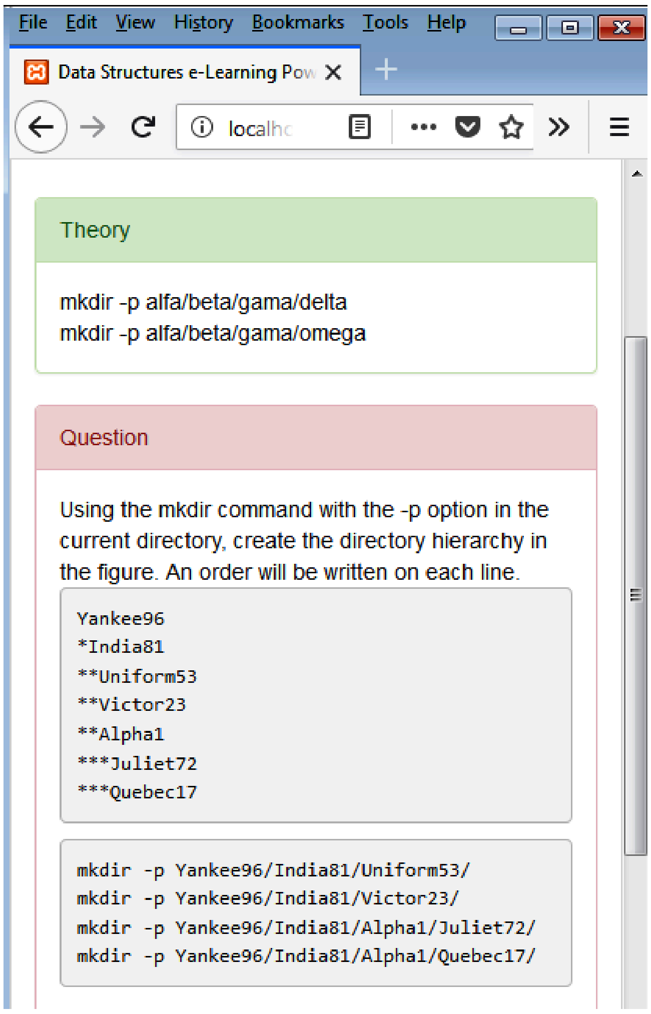
The result persistence module is in charge of evaluating the learner's response and for storing it in the database via AJAX calls.

Figure 7 depicts the view of our prototype when running the previous, thoroughly explained `mkdir -p` example instantiated with different random data.

VII. Conclusions and Future Work

We conclude that we accomplished an abstracting research covering Unix / Linux concepts and topics used in industrial IT services.

One Unix / Linux topic, namely the one about files, was abstracted and compiled into an AGLO model example, which was explained thoroughly. We consider directories to be special files.


File Edit View History Bookmarks Tools Help
Data Structures e-Learning Pow
localhc

Theory

mkdir -p alfa/beta/gama/delta
mkdir -p alfa/beta/gama/omega

Question

Using the mkdir command with the -p option in the current directory, create the directory hierarchy in the figure. An order will be written on each line.

```
Yankee96
*India81
**Uniform53
**Victor23
**Alpha1
***Juliet72
***Quebec17
```

```
mkdir -p Yankee96/India81/Uniform53/
mkdir -p Yankee96/India81/Victor23/
mkdir -p Yankee96/India81/Alpha1/Juliet72/
mkdir -p Yankee96/India81/Alpha1/Quebec17/
```

Fig. 7. Prototype Implementation

Most commands seem to perform simple atomic actions which are easy to implement with AGLOs. The utility of such related AGLOs are the exercising of command line arguments of a very same command which is important in industrial environments.

Other commands like the `fdisk` utility, which is interactive, demands the generation of a complex partitions scenario and can be exercised better separately for each micro commands.

Some commands like `ps`, `ifconfig` do not have any variability symbols for the input, but they output a complex scenario where the random data (processes, network interface configurations) must be carefully generated in order to be realistic.

The Unix / Linux command tests seem to make use of the feedback section from the AGLO model. Thus, the student can see the output of his command.

We can also estimate that there is a large initial effort in creating reusable library components, currently implemented as JavaScript classes and methods, that mimic the Unix / Linux commands features.

From the economical point of view, such an AGLO collection could be interesting for companies and tertiary educational academic or non-academic systems that would like to train technicians able to handle IT service development and maintenance projects.

As future work we plan to extended the topics of AGLOs in the area of cluster configurations, clouds for industrial environments in order to get closer to realistic challenges.

On a long term perspective we intend to abstract domain knowledge, to capture semantics in ontologies and to infer automatically as much as possible corresponding AGLOs from them.

Another perspective is to publish the student AGLO responses in a block chain distributed public ledger that universities and companies may use for human resources skills and knowledge assessment.

REFERENCES

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

[7] ECMA International. Standard ecma-262 ecmascript 2016 language specification. http://www.ecma-international.org/publications/standards/Ecma-262.htm, 2016.

[8] The jQuery Foundation. jquery project. https://www.jquery.com, 2018.

[9] Susan Lauber, Philip Sweany, Rudolf Kastl, and George Hacker. *Red Hat System Administration I Student Workbook*. 2014.

[10] Eric Steven Raymond. *The Art of Unix Programming*. 2003.