International Journal of Engineering & Technology, 7 (4.36) (2018) 315-322

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Software for Creating Tutorials and Examinations on Natural Languages

Polina Dolmatova

Institution: Applied Mathematics and Informatics department, American University of Central Asia, 7/6 Aaly Tokombaev Street, 720060, Bishkek, Kyrgyz Republic
Telephone: +996312915000 (+426)
Fax: +996312915028
Hand phone: +996555241450
**Corresponding author E-mail: dolmatova_p@auca.kg*

Abstract

Currently, philologists often use various software in studying and testing foreign languages. This work presents the software for creating tutorials and exams by philologists independently from software developers. To cover various features of natural languages and involve more abilities of users, such tutorials and exams contain randomly generated tasks of different types. Both questions and answers of the tasks can be text, graphic, sound, or combined. This software implements the algorithmic programming language for creating randomly generated (parameterized) tasks. The paper describes the objects, the syntax, and the procedure for generating random tasks in this programming language. The developed software is tested and used in secondary and higher educational institutions.

Keywords: *algorithmic programming language, complex examination, natural language, parameterized question, randomly generated task.*

1. Introduction

Nowadays, computer equipment is widely used in studying foreign languages and testing of language proficiency. There are various software packages for learning languages. However, majority of such approaches use methods of studying the language with the help of an intermediary language (most often, a native language for a person).
Existing software for language proficiency testing (TOEFL, IELTS, recently developed KyrgyzTest) do not completely involve possibilities brought by up-to-date computers: tasks are unvaried; most are textual, based on multiple choice.
We decided to develop software for creating tutorials and exams with following options:

1. Tutorials and exams contain randomly generated tasks of all types (textual, graphic, audio, or combined) with properly generated responds (textual, actions with graphic objects, voice, or their combinations).
2. Philologists can develop their own tutorials and exams independently from software developer.
3. User-learner and user-examinee can give a text respond, a voice respond, and a respond in the form of actions (parallel transfer, rotation, and other transformations) with graphical objects on the screen.

2.Preceding Works

Asher (1966) proposed the method of studying the language Total Physical Response (TPR). The essence of this method is the following. A person who studies the language should understand new words and react to them in a corresponding way without translation into a language known to him. However, Choo (2006) mentioned in his review of Live Action English Interactive – TPR on a computer that this technique did not appear earlier in the study of languages using computer programs.
Kashy et al. (1993) suggested using personalized assignments while conducting learning and testing on computers.
Pankov (1992) proposed to study languages by means of various "actions" with objects on the screen without any other intermediate language. This is as follows: computer demonstrates the word and the corresponding actions to the user; the same word and the corresponding initial situation appear; the user needs to perform the action (with the cursor); the computer checks the correctness of the action performed by the user.
If the set of situations is fixed, then the user can form a connection of a certain concept, not only with the word being studied, but also with other words present in the corresponding commands. Therefore, situations (tasks) should be generated randomly, that is, when the program is restarted, other, but similar, situations should appear (Pankov, 1996).
To improve the efficiency of testing, Pankov and Janalieva (1995) formulated the following principles of complex examination:
- generativity (a complete task must not exist before the testing and must be randomly generated just before it);
- uniqueness (all examinees must obtain different versions of tasks);
- full confidence (if the testing is official and is conducted by a computer program (generating tasks) then nobody must know answers before the end of testing).

Pankov and Alimbay (2005) presented the software for interactive learning of natural language. However, each new concept and every scene had to be programmed anew that made this software dependent on the developer.
Kustova and Paducheva (1994) suggested mathematical definition of the elementary movements using the only example. Using this definition, Pankov and Dolmatova (2008) proposed a draft of

algorithmic language for independent representation of natural languages' notions. Pankov and Dolmatova (2009) presented the first version of algorithmic programming language TaskLang for developing exams on natural languages.

3.Algorithmic Programming Language Tasklang

We developed the software "Language Education System" (LES) for creating tutorials and complex examinations on natural languages. Philologists can use LES to compose tutorials and exams independently from developers of LES. Since not all philologists know the programming languages, a special algorithmic language has been developed; the author of the tutorial or exam can quickly learn it and create tasks using it. We call the developed language TaskLang. Programming in TaskLang consists of (Pankov & Dolmatova, 2009):
- defining primary objects;
- forming extended objects;
- composing random tasks;
- forming of examination.

All this we do in the interpreter of TaskLang – "Language Education System" (LES). User can write a set of instructions in TaskLang (TaskLang program) in special editor. Each program execution generates a new random task.

There are two types of objects: primary and extended (set of homogeneous primary objects). A primary object is a string. This string can identify various types of information: text, graphics, and sound. If an extended object is used, then a primary object is substituted (randomly) from the corresponding set. If we use an extended object in the task several times, it takes the same value. You can query the second value of the same object, and then it takes a different value from the first, but it remains constant. If we run the task for the second time, the extended object values will change according to this rule.

3.1. Objects in TaskLang

The definition of a primary object includes:
- the name of the object (object-name);
- the C# class, capable of responding to external events and drawing instances of this class (object-class). Such classes can be taken from the standard class library, or written in C#;
- the text word or the list of its grammar forms (object-word);
- the attribute values. The standard attribute is object-word.

The definition of an extended object includes:
- the name of the object;
- list of primary objects.

Extended objects are of two types:
- random sequence of primary objects;
- fixed sequence of primary objects.

3.2. Random text task

A random text task is a sequence of:
- the string constants (including words, spaces, and punctuation marks);
- the names of primary objects;
- the names of extended objects (considered as references to object-words) with definitions of necessary grammatical forms of object-words;
- the operators of TaskLang.

3.3. Random graphic task

A random graphic task is a sequence of:
- the arranging of extended objects in the graphical window;
- the assigning number of layers to extended objects;
- the assigning the direction of slow motion to some extended objects;
- the setting the sequence of actions as the correct answer to the task and the set of incorrect actions.

3.4. Response in random task

In the case of a random text task, the response is an array.
In the case of a random graphic task, the textual response is a special function that displays an invitation to enter a text.
User can give the response in form of actions with graphical objects on the screen. To analyze such response, random graphic tasks use the concept of an event. Event is some action performed as a reaction to certain actions of the user. In TaskLang, there are two types of events:
- correct events. Occur when the user has performed some action correctly; such events are performed strictly in turn and allow you to change the state of the scene during the execution of tasks;
- incorrect events. Occur in response to incorrect user actions; the occurrence of an incorrect event stops the task, and it means that the user failed with this task.

3.5. Logic of TaskLang program execution

TaskLang program is executed from the top down. Normal text is output in the same form as it is in the text of the program. Special constructions in TaskLang, enclosed in [] (square brackets) are output in a different form.
The simplest constructions are the branching structure, a comment, a set of primary objects, a set of extended objects, and a reference to an extended object.
Table 1
TaskLang syntax

Table 1 contains the syntax of TaskLang.

TaskLang construction	Syntax
Comment	[<plain-text>]
Branching structure	[<plain-text>|<plain-text>|…]
Declaring primary object	[Declare*<object-name>#<object-class>|word=<object-word>|<attribute-name2>=<attribute-meaning2>|…]
Declaring extended object as random sequence of primary objects	[<extended-object-name>~<primary-object-name1>|<primary-object-name2>|…]
Declaring extended object as fixed sequence of primary objects	[<extended-object-name>=<primary-object-name1>|<primary-object-name2>|…]
Extended object reference	[<extended-object-name>#<number>]
Function that returns the value of the object's property	[GetProperty*[<extended-object-name1>#<number1>]|<attribute-name1>] or [GetProperty*<primary-object-name1>|<attribute-name1>]
Function that sets the value of the object's property	[SetProperty*[<extended-object-name1>#<number1>]|<attribute-name1>=<attribute-meaning1>] or [SetProperty*<primary-object-name1>|<attribute-name1>=<attribute-meaning1>]
Function describing the word-formation algorithm	[<language-code>*<word>|<primary-form-of-affix>] or [Gram=word|grammar-form-1|grammar-form-2|grammar-form-3|…]

Function describing the word-formation algorithm in Kyrgyz language	[KGZALG*<word>|<primary-form-of-affix>]
Function showing the interpreter that the sequence of characters following the prefix should be recognized as an arithmetic expression	[MATH*<numeric-expression>]
Function that displays the object on the screen	[Draw*[<extended-object-name1>#<number1>]] or [Draw*<primary-object-name1>]
Function that sets the response	[Answer*[GetProperty*[<extended-object-name1>#<number1>]|<attribute-name1>]] or [Answer* [GetProperty*<primary-object-name1>|<attribute-name1>]] or [Answer*<plain-text>]
Function that plays the sound	[Playsound*<file-name>]
Function of speech recognition	[Speech*< plain-text >|<probability>]
Correct event	[Event*<Boolean-Condition>]
Incorrect event	[IncorrectEvent*<Boolean-Condition>]
Function that specifies the delay between events	[Delay*<number-of-seconds>]
Mouse grasping event	[Grasp*<primary-object-name1>]
Mouse ungrasping event	[Ungrasp*<primary-object-name1>]
Function of two objects intersection	Intersect(<object-name1>, <object-name2>)
Function of two objects superposition	Subset(<object-name1>, <object-name2>)

4.Procedure for generating an extended task

First, the TaskLang program is loaded from a binary file. Then an instance of the Parser class is created, and the text of the TaskLang program is transferred to it. Next, the TaskLang interpreter parses the program text. In the parser, a TaskLang expression is analyzed (Figure 1). First, we enter the random number table with a seed offset. Then an instance of the ParserEnvironment class is created. It declares lng, answer, and default variables. Then the ParseLanguage subroutine is called, where ParserEnvironment is passed as a parameter. After that, the current position of the cursor in the Listing TaskLang program is checked. If it is less than the length of Listing, an error message is displayed in Listing. If not, then the randomly generated task text, the correct answer, and the default value for the input field are output.

Consider the ParseLanguage subroutine (Figure 1). The value of the Escaped flag is set, indicating whether the next character of the TaskLang text will be screened. Further, in the cycle, the program performs a symbol-by-symbol processing and the ready TaskLang text of the random task is returned.

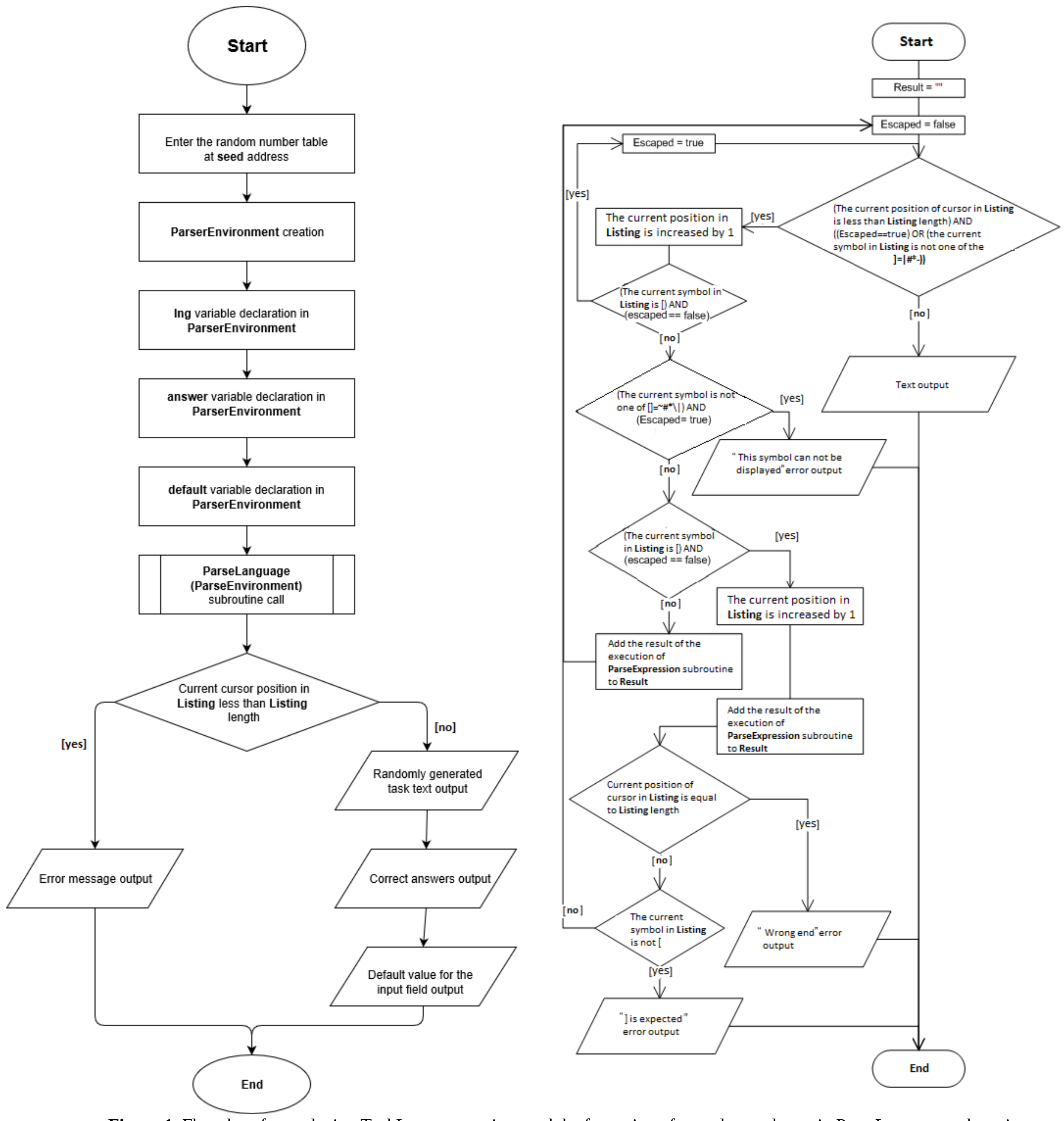
Start
Enter the random number table at seed address
ParserEnvironment creation
lng variable declaration in ParserEnvironment
answer variable declaration in ParserEnvironment
default variable declaration in ParserEnvironment
ParseLanguage (ParseEnvironment) subroutine call
Current cursor position in Listing less than Listing length
[yes]
[no]
Randomly generated task text output
Error message output
Correct answers output
Default value for the input field output
End
Start
Result = ""
Escaped = false
Escaped = true
(The current position of cursor in Listing is less than Listing length) AND ((Escaped==true) OR (the current symbol in Listing is not one of the]=|#*-])
[yes]
The current position in Listing is increased by 1
(The current symbol in Listing is [) AND (escaped == false)
[yes]
[no]
Text output
[no]
(The current symbol is not one of []=~#*\|) AND (Escaped= true)
[yes]
" This symbol can not be displayed" error output
[no]
(The current symbol in Listing is [) AND (escaped == false)
[yes]
The current position in Listing is increased by 1
[no]
Add the result of the execution of ParseExpression subroutine to Result
Add the result of the execution of ParseExpression subroutine to Result
Current position of cursor in Listing is equal to Listing length
[yes]
[no]
" Wrong end" error output
[no]
The current symbol in Listing is not [
[yes]
"] is expected" error output
End

Figure 1. Flowchart for analyzing TaskLang expressions and the formation of a random task text in ParseLanguage subroutine

The subroutine ParseExpression extracts from the TaskLang program expressions containing special constructs and functions in TaskLang, processes them and returns the result in the form of text, which will be further included in the corresponding section of the TaskLang task (Figures 2, 3).

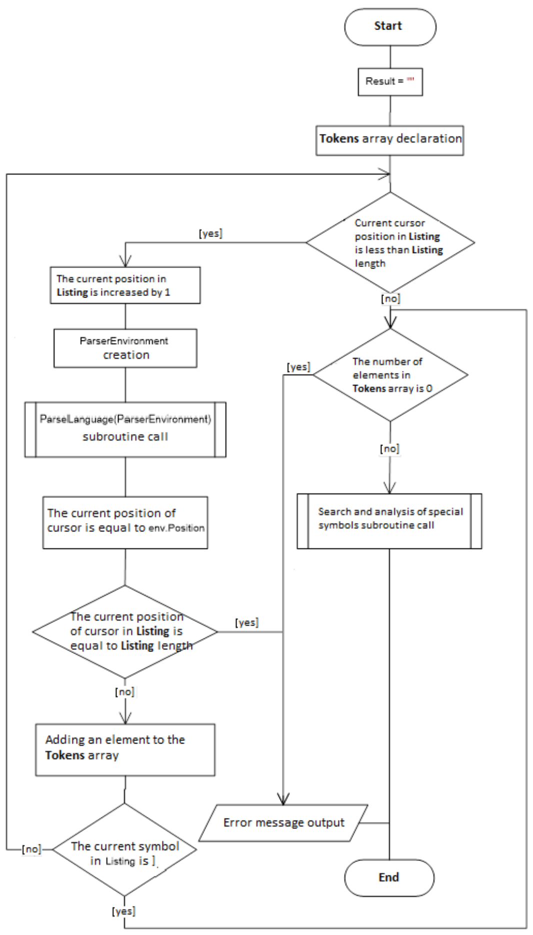
Start
Result = ""
Tokens array declaration
Current cursor position in Listing is less than Listing length
[yes]
[no]
The current position in Listing is increased by 1
ParserEnvironment creation
ParseLanguage(ParserEnvironment) subroutine call
The current position of cursor is equal to env.Position
The current position of cursor in Listing is equal to Listing length
Adding an element to the Tokens array
The current symbol in Listing is]
The number of elements in Tokens array is 0
Search and analysis of special symbols subroutine call
Error message output
End

Figure 2. Flowchart for processing text of TaskLang, performed in the routine ParseExpression

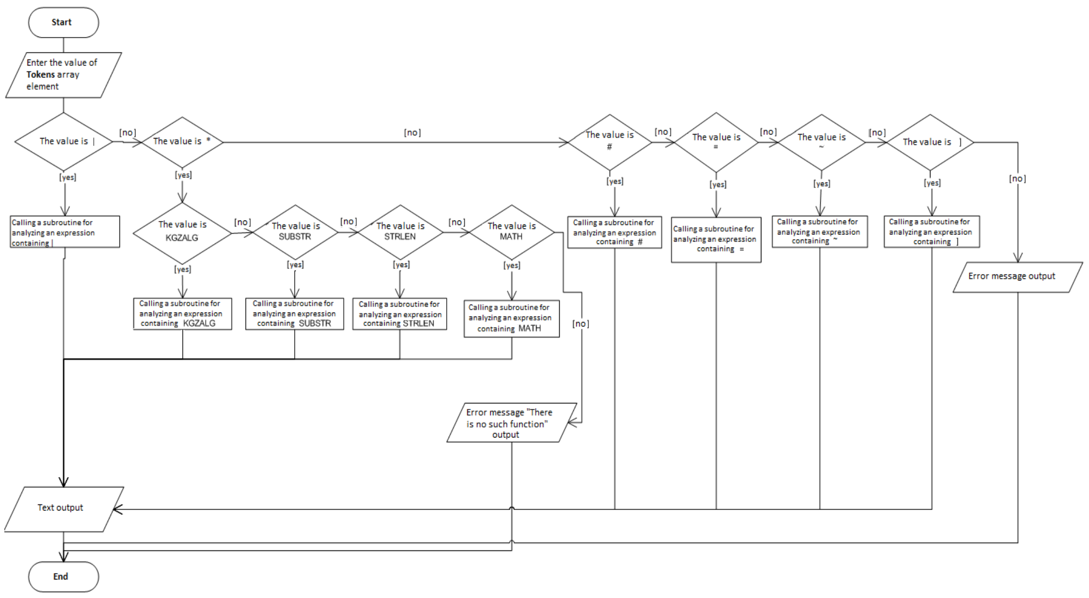
Start
Enter the value of Tokens array element
The value is |
The value is *
The value is #
The value is =
The value is ~
The value is]
The value is KGZALG
The value is SUBSTR
The value is STRLEN
The value is MATH
[yes]
[no]
Calling a subroutine for analyzing an expression containing |
Calling a subroutine for analyzing an expression containing KGZALG
Calling a subroutine for analyzing an expression containing SUBSTR
Calling a subroutine for analyzing an expression containing STRLEN
Calling a subroutine for analyzing an expression containing MATH
Calling a subroutine for analyzing an expression containing #
Calling a subroutine for analyzing an expression containing =
Calling a subroutine for analyzing an expression containing ~
Calling a subroutine for analyzing an expression containing]
Error message output
Error message "There is no such function" output
Text output
End

Figure 3. Flowchart for search and analysis of special TaskLang constructions, performed in the routine ParseExpression

5.Language Education System

Applications in Language Education System have a modular structure. Each module is an isolated part of the set of task types that are required in order to develop a tutorial or complex exam application. The editor, the tutorial, and the tester use a common module factory in their work, which provides access to the modules using the IEditor, ITutorial, and ITester interfaces, respectively (Figure 4). This library uses two main modules - a module of textual tasks (a random text generator) and a module of graphic tasks (a scene generator), accessed through the IBolt interface. The module of textual tasks uses the implementation of the TaskLang interpreter, and the graphic tasks module, in turn, uses the implementation of the TaskLang interpreter with support for graphical functions that are located in the namespaces RandomTextGenerator and SceneGenerator, respectively.

To process collision, we used Box2D engine in order to enhance user experience with scenes. For graphical output, we used OpenGL library.

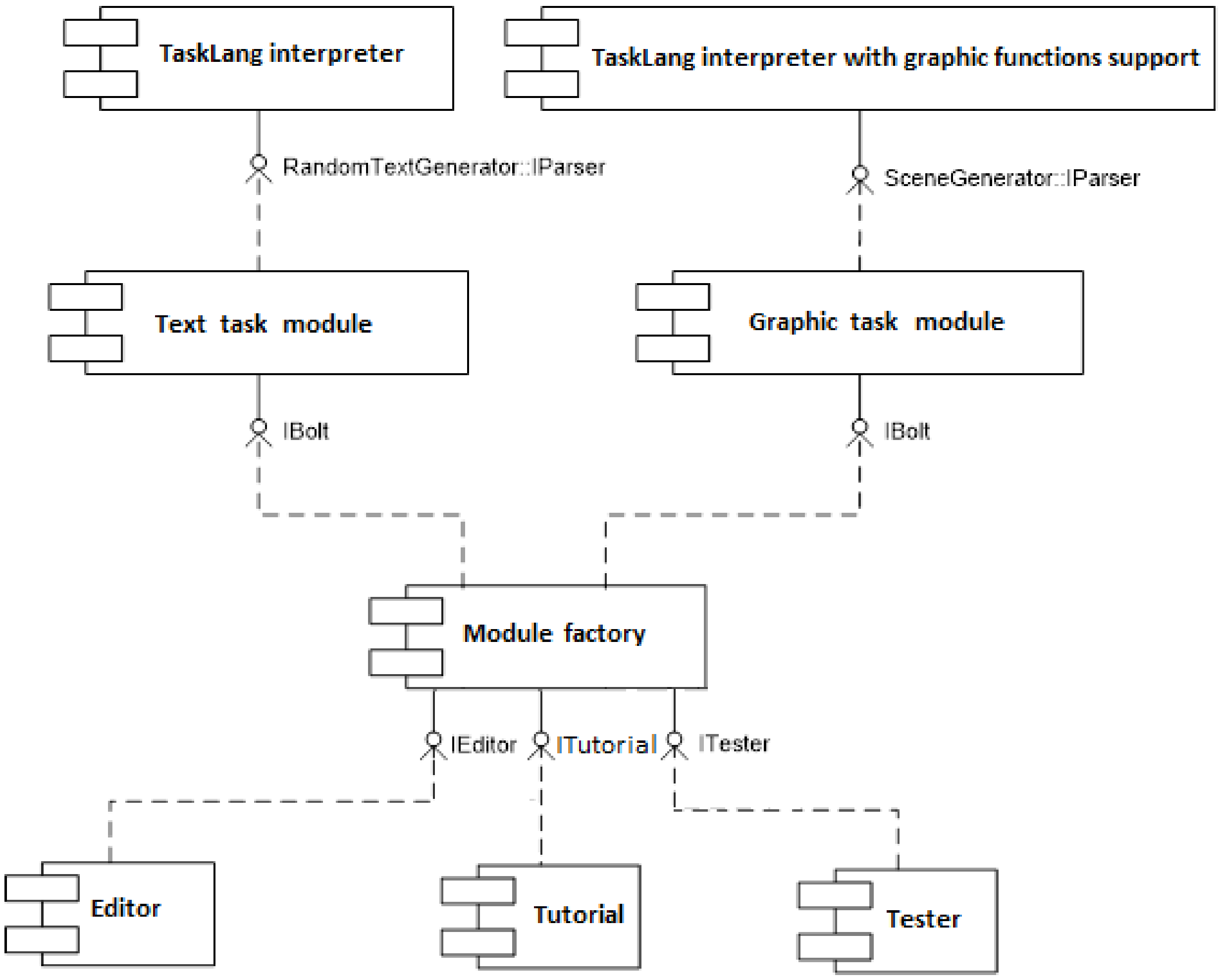

Figure 4. Language Education System components

There are three types of LES users: a philologist (as teacher or examiner), a learner and an examinee.
The philologist has the opportunity to compose his/her own tutorial or exam (Figure 5), that is: to give the name of the tutorial or exam; set parameters such as wrong answer reactions in tutorial and time limits of the exam; create a task of some type; to compile a task tree from these tasks. The philologist also recruits learners or examinees and monitors the results of exam by viewing the report and submitting the final examination score.
The learner can study language by completing the tasks.
The examinee can take the language exam by completing the tasks, and get acquainted with the results of the exam after it is completed.

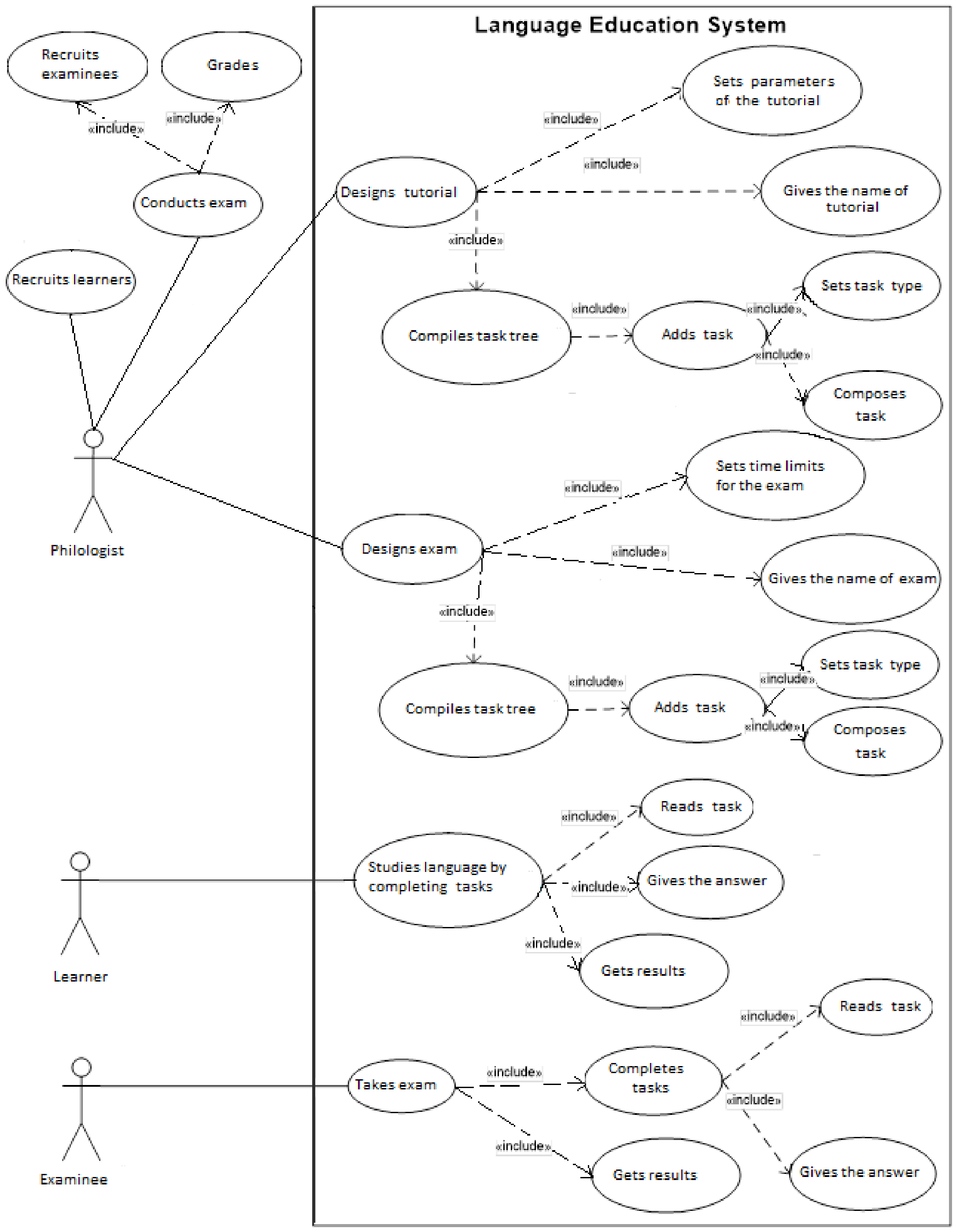

Figure 5. Language Education System users experience

6.Conclusion

The software "Language Education System" combines the following ideas:

- development of randomly generated tasks of all types (textual, graphic, audio) with properly generated responds (textual, actions with graphic objects, voice) in tutorials and exams;
- possibility for using total physical response approach in tutorials;
- easy-to-use for philologists for designing their original tutorials and exams independently from LES developers.

State Language Department of Kyrgyz Republic recommended using LES in educational institutions. LES was used for developing complex exam on Kyrgyz language that was demonstrated in a number of universities, schools and the Institute of Theoretical and Applied Mathematics of the National Academy of Sciences of Kyrgyz Republic. Version 1.1 of LES that was developed within the framework of the projects of the National Academy of Sciences of Kyrgyz Republic is available on http://math.aknet.kg/library/programms.html. Philologists in American University of Central Asia for developing tests and exams on Kyrgyz language use LES.

Special module for Moodle was developed; this module allows creating tasks on TaskLang in Moddle. Tasks in TaskLang in Moodle can be used not only for language courses but also for any other academic course.

References

[1] Asher, J. (1966). The Learning Strategy of the Total Physical Response: A Review. *The Modern Language Journal*, *50*(2), 79-84. doi: 10.1111/j.1540-4781.1966.tb03573.x

[2] Choo, J. (2006). CALICO Journal, 24(1), 209-218. Retrieved from http://www.jstor.org/stable/24156304

[3] Kashy, E., Sherrill, B. M., Tsai, Y., Thaler, D., Weinshank, D., Engelmann, M., & Morissey, D. J. (1993). CAPA, an integrated computer-assisted personalized assignment system. *American Journal of Physics,* 61(12), 1124-1130. doi: 10.1119/1.17307

[4] Kustova, G. I., & Paducheva, E. V. (1994). Semantic Dictionary as a Lexical Database. In W. Martin, W. Meijs, M. Moerland, E. ten Pas, P. van Sterkenburg, & P. Vossen (Eds.), *Proceedings of EURALEX-1994* (pp. 479-485). Amsterdam, the Netherlands.

[5] Pankov, P. S. (1992). *Teaching and Controlling Program on Word-Formation in Kyrgyz Language on PC* (in Russian). Bishkek: Mektep.
Pankov, P. S. (1996). Independent learning for Open society. *Proceedings of the High Education Support Program Seminars* (pp. 27-38). Bishkek, Kyrgyz Republic: Soros-Kyrgyzstan Foundation.

[6] Pankov, P. S., & Alimbay, E. (2005). Virtual Environment for Interactive Learning Languages. In Z. Vetulani (Ed.), *Proceedings of the 2nd Language and Technology Conference* (pp. 357-360). Poznan, Poland.

[7] Pankov, P., & Dolmatova, P. (2008). Algorithmic Language and Classification of Verbs for Computer-Based Presentation. *American University of Central Asia Academic Review, 1*(7), 233-239.

[8] Pankov, P., & Dolmatova, P. (2009). Software for Complex Examination on Natural Languages. In Z. Vetulani (Ed.), *Proceedings of the 4th Language and Technology Conference* (pp. 502-506). Poznan, Poland: Springer.

[9] Pankov, P. S., & Janalieva, J. R. (1995). Experience and prospects of using the complex UNIQTEST of unique test tasks in educational process (in Russian). *Proceedings of the Education and Science in New Geopolitical Space Conference* (p. 217). Bishkek, Kyrgyz Republic: International University of Kyrgyzstan.