Research Insights
Изследователски проникновения

VERIFICATION OF ACCUMULATIVE FRAME SYSTEM IN PROGRAMMING TRAINING

Silvia Gaftandzhieva, Rositsa Doneva
University of Plovdiv "Paisii Hilendarski" (Bulgaria)

Abstract. The paper presents a part of a study conducted for the needs of e-learning on the application of frames for representation of knowledge and processes in training on a particular subject domain. As a result of the study the classical notion of the frame models is developed further and the so-called accumulative frame model is proposed. The main rationales for the introduction of the accumulative frame model are: (1) to provide a formalism for conceptual modelling in subject domains and (2) to serve as a basis for automation of e-learning and e-training activities by implementation of intelligent tools for extraction, aggregation and accumulation of data and knowledge for educational needs. A system of 36 accumulative frame models and its verification for fulfilment of the intended purpose in the rationale (1) are presented.

Keywords: conceptual modelling; frame-lased knowledge representation; accumulative frame system; programming training

Introduction

Frames as knowledge representation formalism are successfully applied in training to achieve stable and in-depth learners' knowledge. For the teacher, the use of training methods based on frame models allows him or her to direct analysis of the information received by the students. For students, frames help them to learn the content in a thorough and meaningful way, focusing on the relationship between the basic ideas and the main details and thus serve to present easily abstract ideas.

A set of studies and experiments in the field should be mentioned. With the purpose of knowledge systematization and problem solving a number of frame-based representations are developed in various subject areas such as mathematics, informatics, physics, chemistry, history, languages, etc. (Kulgildinova & Uaissova, 2016; Shivacheva, Totkov & Doneva, 2017; Gurina, et al, 2007). An interesting example is the frame-based programming editor Greenfoot IDE (Thomas et al., 2016) implemented to facilitate training in the field of programming. The IDE allows students to combine and edit formatted programming code presented by frame structures and thus limits syntax complexity and errors. In the field of e-learning, a

research on the applicability of frames for knowledge representation and improvement of students' knowledge and understanding, conducted in LMS Moodle (Fonseca, 2015), shows that students obtain higher results during exams. This is due to the fact that frames force students to group and structure information, to focus on concepts, and to achieve a higher level of concentration.

The paper presents a part of a study conducted for the needs of e-learning on the application of frames for representation of knowledge and processes in training on a particular subject domain. As a result of the study the classical notion of the frame models is developed further and the so-called accumulative frame model (AFM) is proposed (see next section). The main rationales for AFM introduction are: (1) to provide a formalism for conceptual modelling in subject domains and (2) to serve as a basis for automation of e-learning and e-training activities by implementation of intelligent tools for extraction, aggregation and accumulation of data and knowledge for educational needs. A system of 36 AFMs is presented below, designed to foster the studying and reasoning of the C++ computer programming. Before starting the computer implementation, in fulfilment of rationale (2), the AFM system is put to verification that it fulfils the intended purpose laid in rationale (1). It is experimented in traditional learning with full-time students studying C++ programming. The verification relies on the comparison of the assessment results obtained by different student groups who solved tasks on the C++ language syntax chosen from six proposed categories of basic learning tasks in this subject domain. Some of the student groups should solve tasks using the frame system and some – using traditional solving approaches.

Accumulative frame system

The innovative accumulative frame model (Doneva, Gaftandzhieva & Totkov, 2018) has been introduced within the research under discussion (Totkov, Gaftandzhieva & Doneva, 2017) as a solution for the problem for conceptual modelling of e-learning tasks. AFM is a development of the classical understanding of a frame model, and therefore has its typical features. In comparison with the classic frame models, besides procedures (which are performed after filling in different slots in the frame) or demons (to calculate values of slots), so-called "accumulative functions" can be attached to the slots of the proposed AFM. They allow the accumulation of additional data in the e-learning process that can be used to automate different learning tasks.

The set of AFMs representing conceptual knowledge in one and the same subject domain are interrelated by various relation links (successor, predecessor, part of, etc.) and form so- called AFM system – the AFM system of the subject domain. An AFM system includes both local knowledge and global knowledge. The local knowledge concerns mainly the knowledge about the network of the included AFMs and about the types to which the values of their slots belong.

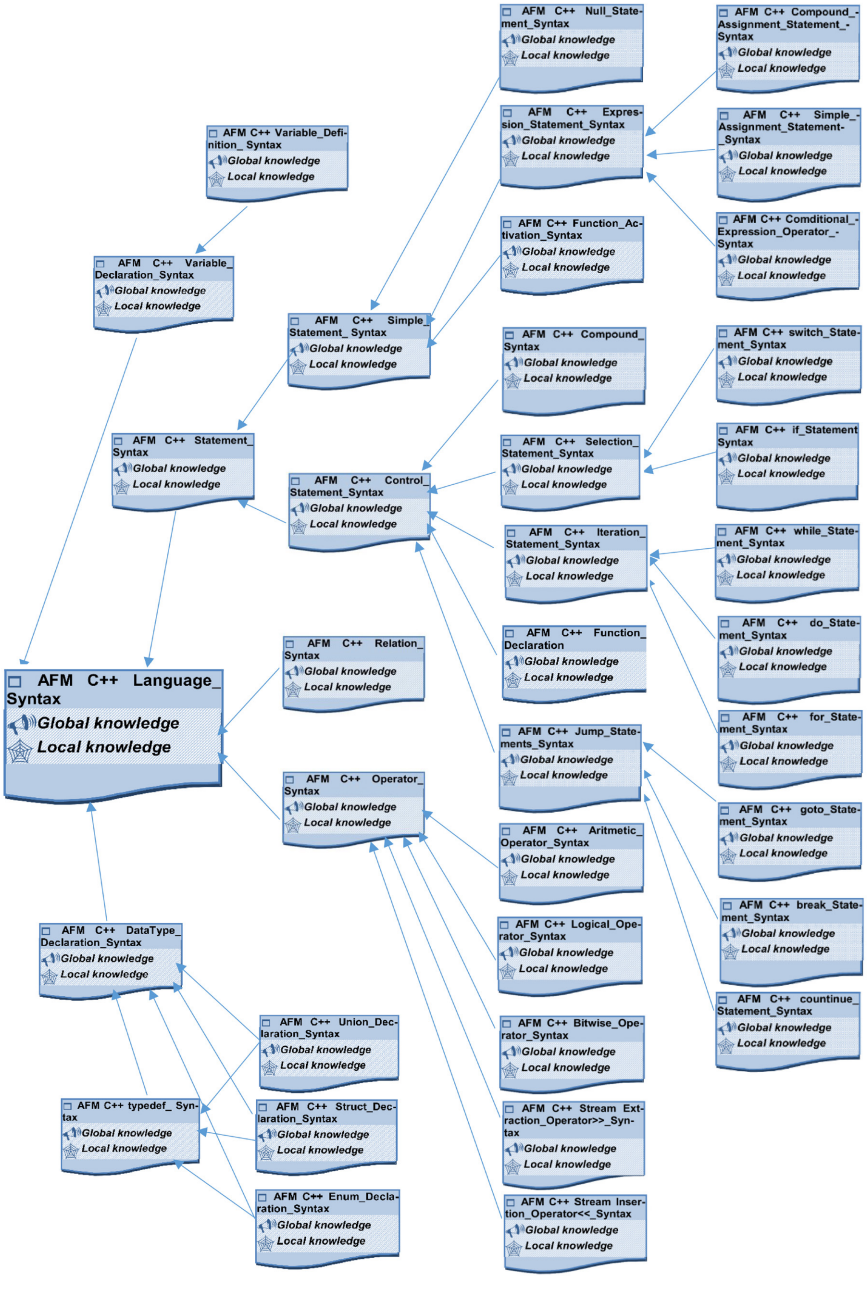

Figure 1. AFM system C++ Language Syntax

The global knowledge includes mechanisms for information extraction over the whole AFM system and evaluation of the individual AFMs slots. These mechanisms (comprising reasoning knowledge) can be bottom-up (data-driven or event-driven) or top-down (driven by expectations). The AFM systems also contain accumulative knowledge, including knowledge about the subject domain, the context and the aspect/purpose of the representation.

Figure 1 presents the AFM system named C++ Language Syntax that is developed on the basis of the proposed models (Doneva, Gaftandzhieva & Totkov, 2018). It offers a conceptual model of knowledge to be studied in the subject domain "Programming" in the context of "Learning programming with C++ programming language" from the aspect of "Learning of the language syntax".

The AFM system consists of 36 AFMs to study the C ++ programming language syntax (Figure 1):

– Declaration and definition of variables;

– Declaration of data types (typedef, struct, union, enum);

– Operators – logical (&&,!, ||), bitwise (&, ^, |, ~, <<, >>), arithmetic (+, -, *, /, %, ++,--);

– Relational and comparison operator (==,! =,>, <, >=, <=);

– Null operator;

– Assignment Statement – Simple Assignment (=) and Compound Assignment (+=, -=, *=, /=, =%);

– Operators for input/output data from/to the standard input/output stream (Stream extraction operator >> and Stream insertion operator <<);

– Conditional operator;

– Jump statements (break, goto and continue);

– Block ({});

– Decision operators – Conditional Statement (if) and Switch Statement;

– Loop operators - for, while and do;

– Grouping (function) operator;

– Calling the function.

Figure 2 presents one of the described AFMs – a frame-prototype for modelling the knowledge for the syntax for declaration of a variable in the C ++ programming language. The frame-instances of the presented AFM must have a special slot called instance_of with value AFM C ++ Vriable_Declaration_Syntax.

From the point of view of the scope, AFM includes both global knowledge and local knowledge about the syntax for declaration of a variable in the C++ programming language.

The global knowledge of AFM includes slots to represent the information about the AFM system to which it belongs. The presented AFM is a part of a subject domain "Programming" (SD slot) from the Computer Science area of study. Correspondingly, the AFM-based representations of knowledge are built in the

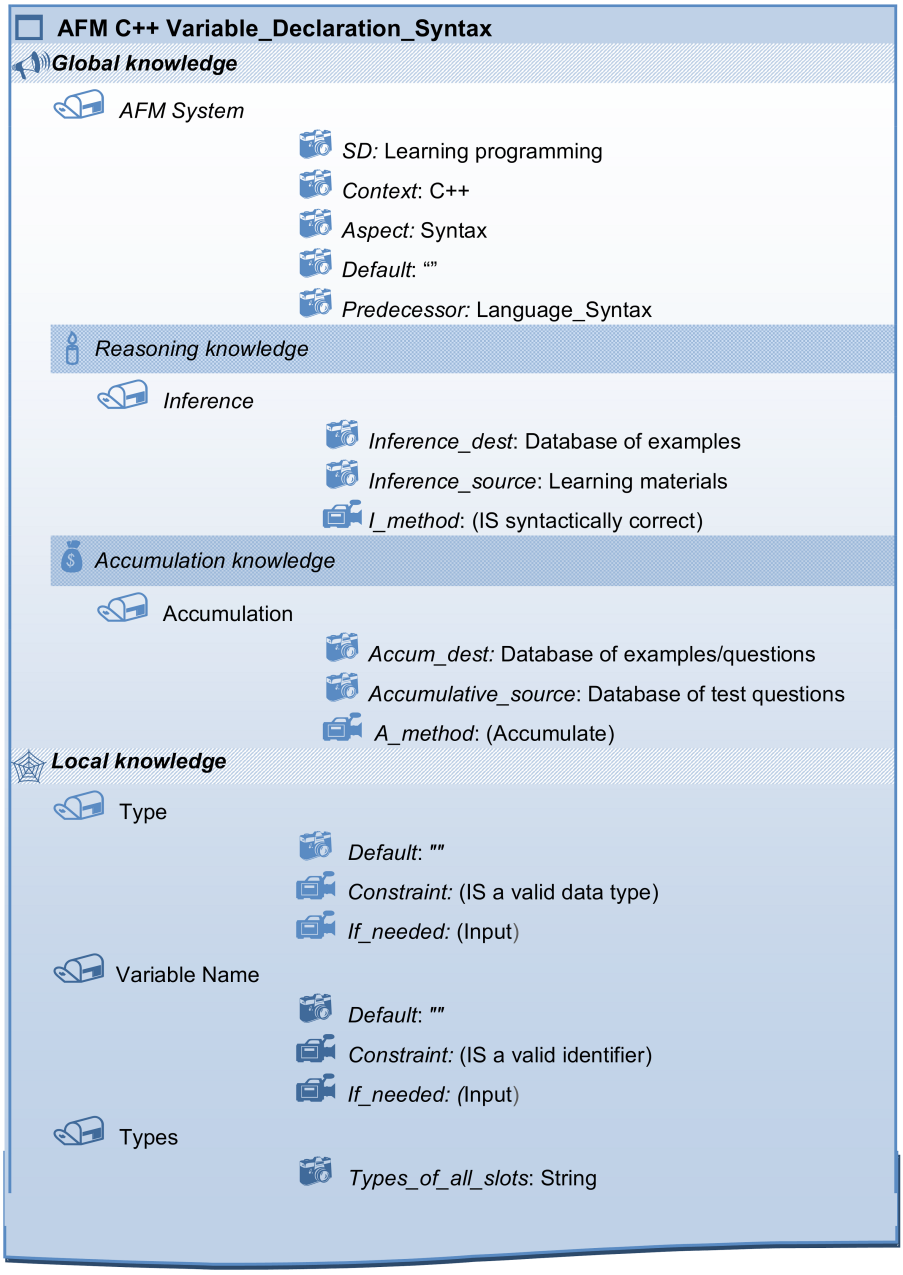

Figure 2. AFM Declaration of a Variable (Syntax)

context of "Learning programming with C++ programming language" from the aspect of "Learning the language syntax" (Aspect slot). The predecessor of the presented AFM is Language_Syntax.

A fundamental part of the global knowledge for each AFM for the representation of knowledge about the syntax for declaration of a variable in the C++ programming language is so-called reasoning knowledge which assists in the extraction/understanding/inference of the information presented through the frame, including special inference methods. They manage a matching process that attempts to assign a value to each AFM slot and that is partially controlled by the relevant slots methods for execution and control of knowledge extraction (filling in the slot, range checking, etc.).

Global knowledge also includes accumulative knowledge and tools for knowledge accumulation – accumulative methods. These methods manage the accumulation process of knowledge related to the C++ programming language syntax in the form of frame-instances, new AFM prototypes and instances, new relations, etc. by using the appropriate interfaces with selected sources of accumulative knowledge. An important part of this accumulative knowledge are the knowledge of the Learning Programming, the context (C++) and the aspect (Syntax) of representation. This knowledge is also part of the global knowledge. Accumulative knowledge is also propagated throughout the local level.

A big part of the knowledge described in the presented AFM (Variable_ Declaration_Syntax) is inherited directly from its predecessor of the FMs system (Language_Syntax) to which it belongs. In particular, it inherits the description about:

– subject domain (SD slot), as well as about the context and the aspect of the representation;

– inference source and destination of data (Inference_source and Inference_dest) and partially about the matching process method (I_method);

– accumulation process, including about the source of accumulative knowledge (Accumulative_source), the destination of the accumulated data (Accum_dest) and the accumulative method (A_method).

The local knowledge describes the AFM for modelling of the knowledge about syntax for declaration of a variable in the C++ programming language from a structural point of view. AFM has two slots. Each slot has a unique name (Type – type name and Name – variable name) and includes one or more facets representing the knowledge for different characteristics of the AFM. Facets serve as a description of both static knowledge (e.g. default value, range) and dynamic knowledge, through the procedures attached to them – so-called "attached procedures" (methods) or "demons". Both Type and Name slots have attached demons that require a value of the slot (If_needed). An important role in the knowledge representation for the syntax for declaration of a variable in the C++ programming language through AFM play the so-called "filing methods" (fillers), which assign values to the slots of the frame (by default value, data retrieval via attached procedures) on the basis of

the current context and slot-specific heuristics. The Constraint procedures start automatically when slot values are inputted/filled in and check whether the inputted/filled in value meets the requirements.

All slots have values from string type. The frame structure is dynamic, i.e. allows the addition of new slots at any time (e.g. adding new slots for declaration of a second variable).

Verification of accumulative frame system

Before starting the computer implementation the AFM system is put to verification in order to be checked whether it fulfil the intended purpose to provide a formalism for conceptual modelling in subject domains. Categories of basic learning tasks on the C++ language syntax are chosen (presented below) for verification of the proposed AFM system. The AFM system is experimented in traditional learning with full-time students studying C++ programming. The verification relies on the comparison of the assessment results obtained by different student groups who solved tasks on the C++ language syntax using the AFM system and traditional solving approaches.

Categories and examples of basic learning tasks

As a first step of the AFM system verification, six categories of basic learning tasks are proposed that are appropriate for studying the C++ programming language syntax – the chosen subject domain:

– ***Category 1***. Development of a syntactically correct programming code on the basis of the created frame-instances;

– ***Category 2***. Development of a syntactically correct programming code after filling in the frame-instances;

– ***Category 3***. Design of task solutions and creation of a syntactically correct programming code;

– ***Category 4***. Detection of syntax errors in the programming code;

– ***Category 5***. Analysis of unfamiliar programming code to detect C++ syntax elements and creation of frame-instances on the basis of frame-prototypes of the syntax elements found in the programming code;

– ***Category 6***. Modification of unfamiliar programming code through using slots' values of frame-instances.

Bellow, each of the six categories is illustrated by an example task and a definition of the students' knowledge it intends to assess.

Example 1 (Task from Category 1): Write a syntactically correct programming code for calculation of the expression value by using the slots values of the frame-instances from Figure 3.

The solution of tasks from Category 1 requires students to create a syntactically correct programming code through the use of given frame-instances (completed by the teacher or extracted from a database with frame-instances).

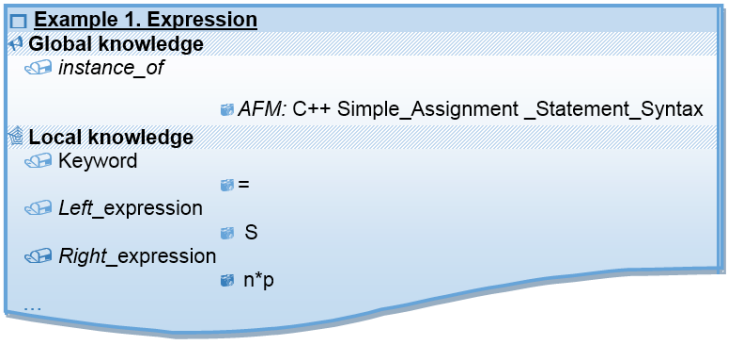

Figure 3. Frame-instance of AFM Simple Assignment Statement (Syntax)

Example 2 (Task from Category 2): Declare a variable to represent the average grade of students from a course. Define and fill out the variable type and its name in the following frame-prototype (Figure 4).

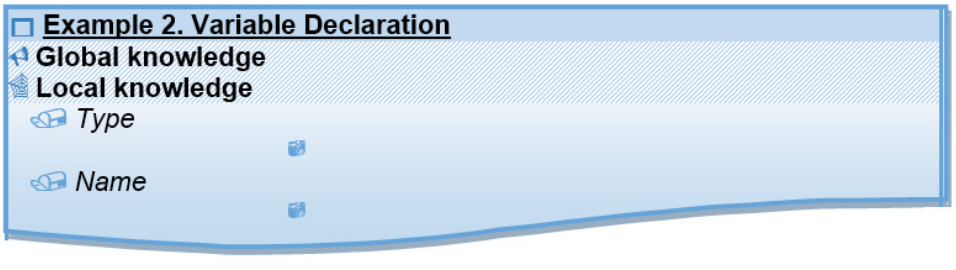

Figure 4. Frame-instance of AFM Declaration of a variable (Syntax)

Unlike the tasks from Category 1, the tasks from Category 2 require students to create frame-instances of the frame-prototypes given by the teacher, and then to write the corresponding programming code.

Example 3 (Task from Category 3): Write a C ++ program that calculates and outputs the total value of the item by its price and count.

The task from Example 3, e.g. requires students to fill out the frame-prototypes Variable_Declaration_Syntax, Stream_Extaction_Operator>>_Syntax, Stream_Insertion_Operator<<_Syntax, Simple_Assignment_Operator_Syntax, Aritmetic_Operator_Syntax, and then to write a C ++ programming code to solve the task.

The tasks included in Category 3 are more difficult and complex than the tasks in Category 1 and Category 2. They check students' knowledge and skills to design the solution of the task independently and create a programming code for its decision. The solution of these tasks requires students to determine what types

of variables, operators, relations, simple and compound statements they have to use to write a programming code. Then students have to create the relevant frame-instances of syntax elements, which they will use to solve the task and finally to write the programming code on the basis of the created frame-instances.

Example 4 (Task from Category 4). Using the created frame-prototype to calculate the expression value (see Figure 3), determine whether the s++n=p programming fragment has syntax errors.

The tasks in Category 4 are intended to check whether students can detect syntax errors in a programming code. Frame-instances (manually filled by the teacher) and a fragment of a programming code in which students have to look for syntax errors are provided to students.

Example 5 (Task from Category 5): Discover the syntax elements in the programming code (Figure 5) and create frame-instances of the relevant frame-prototypes.

```
int main()
{
    double S;
    int n=5;
    double p=5.6;
    S = n+p;
    cout << "S = ";
    cout << S;
    return 0;
}
```

Figure 5. Programming code

Example 6 (Task from Category 6): Modify the programming code (see Figure 5) on the basis of values of the slots of the frame-instance from Figure 3.

The tasks in Category 5 and Category 6 aim to verify whether students can detect syntax elements in unfamiliar programming code and to modify unfamiliar programming code without knowing the task for which the solution it was created.

Verification of the AFM system with full-time students studying C++ programming language

During the assessment, students are randomly divided into three groups. Some of the student groups should solve problems using the frame system and others using

traditional solving approaches. Each group has to solve two tasks that generally have the same requirements:

– Task 1. Declare the appropriate variables for storing data for the count of items, the price of an item, and the total price of the items from the same type.

– Task 2. Declare a C++ function with formal parameters for the count and price of an item which returns as a result of the total price of the items.

But for the first group of students the tasks are formulated so that to meet also the requirements of basic learning tasks from Category 1 – the students have to solve the two tasks by using slots values of the filled frame-instances.

The second group of students has to solve the tasks with the additional requirements, stipulated by Category 2, i.e. by creating frame-instances of frame-prototypes Variable_Declaration_Syntax and Function_Declaration_Syntax.

The third group of students has to write a C++ programming code, as a solution of the tasks, directly without the use of any frames.

The solutions of each task are assessed with grades (using six-point grading scale). Figure 6 presents the grades obtained by each of the three groups of students. The results clearly show that students who have solved the tasks by using the values of the slots of filled frame-instances (Group 1) and creation of frame-instances (Group 2) have higher grades than students in Group 3. Especially large is the difference between "Fail 2" grades and "Excellent 6" grades obtained by the students. The third group of students received the smallest number of excellent grades - 40% of students in Group 1 have excellent grades for the first task and only 20% of students have excellent grades for the second task. In contrast, a large part of students from Group 1 and Group 2 have excellent grades. 80% of students from Group 1 have excellent grades for the first tasks and 60% for the second task. Students from Group 2 have also achieved high results - 57% of them have done well with both tasks. Only students in Group 3 have "Fail 2" grades.

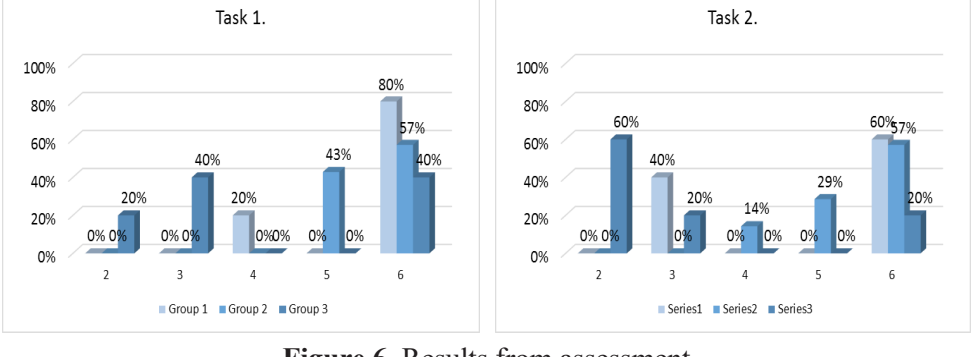

Figure 6. Results from assessment

Conclusion

The paper is part of a study (Totkov, Gaftandzhieva & Doneva, 2017), dedicated to the application of frames for the representation of knowledge and processes in e-learning and their applications for the development of intelligent software solutions in various areas.

The proposed AFM system was experimented with full-time students studying C++ programming language. The results of the verification prove that the proposed AFM system provides an appropriate formalism for conceptual modelling and can be applied in training for improvement of students' knowledge and understanding. The conducted experiments confirm the conclusions of the study (Fonsec, 2015) that the application of the frame-based knowledge representations helps students to achieve higher exam results.

The AFM system can be easily modified and applied in studying other programming languages. In-progress is the design and creation of AFM systems for other subject domains, in the field of Physics, English, Mathematics and Computer Science. The intelligent software tools are in development process. They will allow the automation of a number of training and learning activities, e.g. filling out of the frame-instances by students, accumulation of created frame-instances in a database with good and wrong examples, generation of learning tasks, assessing the students' knowledge, etc. Or, generally speaking, by using frame-based conceptual models of subject domains, the tools will provide acquisition, extraction, inference and accumulation of relevant knowledge.

NOTES

1. Thomas W. et al (2016). Evaluation of a Frame-based Programming Editor. In Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16). ACM, New York, NY, USA, 33-42. DOI: https://doi.org/10.1145/2960310.2960319
2. Gurina R. et al. (2007). Frejmovie opori [Фреймовые опоры. NII Shkolnih tehnologii, Moscow].

REFERENCES

Doneva, R., Gaftandzhieva, S., Totkov, G. (2018). Frame Representations in e-Learning – Applications and Developments. *International Journal on Information Technologies and Security*, 2(10), 23 – 32.

Fonseca, O.H. (2015). Learning styles and knowledge representation systems in Ecaes (quality examinations for higher education) for medical students. *Revista Horizontes Pedagógicos.* 17(1), 42 – 52.

Kulgildinova, T. & Uaissova, A. (2016). Realization of frame-based technologies in the context of education in informatization, *Journal of Theoretic and Applied Information Technology*, Vol.89. No.1.
Shivacheva G., Totkov, G. & Doneva, R. (2017). Reading programming code with understanding. *Technics and Technologies,* Vol.15, 58 – 63.
Totkov, G., Gaftandzhieva, S. & Doneva, R. (2017). Accumulative Frame Models in e-Learning. *Technics and Technologies*, Vol.15, 17 – 20.

✉ **Dr. Silvia Gaftandzhieva, Assist. Prof.**
Department of Computer Science
Faculty of Mathematics and Informatics
University of Plovdiv "Paisii Hilendarski"
Plovdiv, Bulgaria
E-mail: sissiy88@uni-plovdiv.bg

✉ **Prof. Dr. Rositsa Doneva**
ECIT Department
Faculty of Physics and Engineering Technology
University of Plovdiv "Paisii Hilendarski"
Plovdiv, Bulgaria
E-mail: rosi@uni-plovdiv.bg