Research Article

Peter Sedlacek*, Marek Kvet, and Monika Václavková

Development of FRIMAN – Supporting Tool for Object Oriented Programming Teaching

https://doi.org/10.1515/comp-2020-0117
Received Mar 30, 2020; accepted May 04, 2020

Abstract: The main goal of this contribution is to present the current developmental state of FRIMAN – the graphical development environment designed to support the teaching process of the object-oriented paradigm. FRIMAN project has two main purposes: 1. simplifying the understanding of the basics of the object-oriented programming for JAVA language beginners, 2. teaching students of applied informatics to collaborate in bigger project development. Therefore, an application called FRIMAN has been developed at the Faculty of Management Science and Informatics at the University of Žilina. This project is developed by students of Master degrees under the leaderships of experienced software developers. The suggested system consists of several modules. In this paper, we focus on the description of selected modules and their current functionality as well as description of future plans for this project and brief description of FRIMAN development process. Attention is paid to a module for class management and a graphical code editor, which enables the creation of method bodies using flow diagrams without the necessity of programming language syntax knowledge. Based on good evaluation by the development team preparing changes in the high school education process, the current application is planned to be applied in practice.

Keywords: Informatics teaching process, software tool, object programming, class management, graphical editor

***Corresponding Author: Peter Sedlacek:** University of Zilina, Faculty of Management Science and Informatics, Univerzitna 8215/1, 01026 Zilina, Slovakia; Email: peter.sedlacek@fri.uniza.sk
Marek Kvet: University of Zilina, Faculty of Management Science and Informatics, Univerzitna 8215/1, 01026 Zilina, Slovakia; Email: marek.kvet@fri.uniza.sk
Monika Václavková: University of Zilina, Faculty of Management Science and Informatics, Univerzitna 8215/1, 01026 Zilina, Slovakia; Email: monika.vaclavkova@fri.uniza.sk

1 Introduction

The very rapid development in the field of information technologies has become an important part of human lives. That is why many companies are seeking for young professionals, who are able to adapt to new or various changing trends very quickly. One of the most affected subjects of these increasing changes are also the universities, which provide education of future professionals.

Graduates of any technically oriented university must get professional education in their particular study field. First, they need practical skills supported by excellent theoretical knowledge. The skills facilitate entry into employment while theoretical knowledge makes it easier to develop new unique solutions to problems that can arise in practice. Students of information technologies and management science are not an exception.

Faculty of Management Science and Informatics of the University of Žilina has divided the education process into three study programs each with a focus on different parts of information technologies. One of these study programs is management centered on information technologies. Continuous effort is made to help students of this program to either develop or improve their understanding and knowledge of modern information technologies. The result of this effort is an ability of future managers to use the modern information technologies as a competition tool for creating a positive and efficient organizational change or bringing about new solutions based on information technologies. But despite the basic knowledge of various operating systems performance and common office applications, our students should also learn the basics of advanced information systems engineering [1].

Students of the management science at our faculty have to complete several subjects aimed at software development. Many of these students have either very weak or even no experience with any kind of programming. Therefore, it is very hard for them to understand the principles of different programming strategies like object-oriented programming [2, 3].

Object-oriented programming (OOP) paradigm is one of most commonly used paradigms these days and every

student in our faculty has to learn it regardless of the study program. In OOP, any problem is solved via so-called "objects", *i.e.* logical entities with their own data and functions and via communication between these objects. These functions, called methods are often used to modify the data fields of the object which they are associated with. For many beginners in software engineering, even the basic logic of a simple program based on OOP principles constitutes a challenging task, because many applications are designed by making them out of objects that interact with one another [3–5], which differs from the style of procedural programming [6], which some students are familiar with.

Another big issue may follow from the correct syntax, *i.e.* when, where and why to write a semicolon or any other specific symbol. Such problems often occur when changing current programming language for a different one. Therefore, we have been searching for a suitable way of making the education process of the programming basics easier mainly for the beginners.

The FRIMAN project was created as an answer to the fore-mentioned problems - *i.e.* a tool to simplify process of learning OOP paradigm in Java language at our faculty. This project was originally introduced in [7] and reported in [8, 9] and it also serves another purpose. In order to be an expert in Information Technology, it is not sufficient to have knowledge, it is also necessary to have experiences within this field. Therefore our master degree students have to gain experience in the development of bigger projects and in collaboration with other developers. For this reason, the FRIMAN project is designed and developed by students of master degree within our leadership [8].

The main research topic and the contribution of the paper is to present the current development state of FRIMAN system and its particular components.

The remaining part of this paper is organized in the following way. The next section provides the readers with the basic characteristics of the FRIMAN project and it describes the architecture of the suggested system. The third section is devoted to the explanation of the Class viewer module and the fourth section summarizes the features of the Code editor module. The fifth and sixth sections describe parts of this project that have arisen during the continuous development and the seventh, and penultimate section briefly describes the work management and organization in this project. Finally, the last section contains the obtained results and future development directions.

2 The basic characteristics of FRIMAN

The first aspect needed to be taken into account is the expected target group of users of the developed system and its requested primary functions. At the same time, it is sufficient to analyze existing software solutions in this area.

Currently, there are many environments, which offer the possibility of simple software projects development. The comparison of selected suitable solutions has led to the conclusion that the suggested system FRIMAN should find such ways of interaction, which would be easy to be understood by various groups of users, mainly beginners in object-oriented programming. With this software tool we want to provide the users with a simple and interactive style of communication and control.

An example of a suitable existing system could be the well-known environment BlueJ [10]. Currently, it is being used in our teaching courses for the beginners. In contrast with professional environments like IntelliJ [11] or NetBeans [12], BlueJ offers a simpler interface for classes and their instances management via a graphical tool for object inspecting. This feature enables the users to thoroughly understand the basic principles of the object-oriented programming.

The main difference between the existing solutions and the suggested developed tool consists in usage of graphical components for creating the algorithms without the necessity of the knowledge of the specific programming language syntax. Instead of writing the code, various graphical components should be used by the program developers to create their application. The resulting code will be generated from the diagram itself afterwards. The suggested environment should check the correctness of the diagram and allow the users to debug their resulting algorithms.

Based on the mentioned primary features and other supplementary system requirements, we have suggested the following architecture of the FRIMAN software tool. The suggested system consists of five modules:

- Editor,
- Debugger,
- Builder,
- Class viewer,
- Core.

The first module is the Editor. It was developed for creation of individual methods and constructor bodies making use of common flowcharts, which can be made in an

interactive way. This module will be explained in detail in the next section.

Algorithms made in the Editor module are automatically compiled to the JAVA language using the module Builder.

The next module is a Class viewer. Its main task is to provide creating classes and their management and also making instances of these classes. This module will be also described in a separate section.

The Debugger module checks possible logical errors in the proposed algorithms. It processes all used graphical components in the resulting flowchart and based on the corresponding programming code, it detects source of errors. This module is currently being implemented.

Finally, the Core module connects all satellite modules and provides proper performance of the whole application. Obviously, the FRIMAN software architecture was designed with regard to future system enlargements.

3 The Class Viewer module

The Class Viewer module was suggested for managing classes and their instances. With respect to future users' abilities and experience, we are looking for the simplest and the most understandable way, in which the users could communicate with the application and create classes, their instances and processing various operations with them. We were inspired by the commonly known JAVA development environment BlueJ [10], which allows using the inspector function as a specific tool.

BlueJ is currently used as the main teaching tool for beginners in our faculty. As we have mostly positive experience with its usage, the original idea of FRIMAN project was to extend BlueJ with new features including creating method bodies using Activity Diagrams and generating source code from them. Nevertheless, the research has shown, that thanks to the complexity of other parts of FRIMAN, it would be easier to implement class management similar to the one provided by BlueJ rather than to extend BlueJ with our new features. Although the first effort that was needed seems larger, this decision also brings some advantages, specifically that we were free in choosing technologies and we could adjust the class viewer to be uniform with other modules.

The basic functions, which should be implemented first in our module for effective class management, can be summarized as follows:

- Class management,
- Creating object instances,
- Inspecting inner values of attributes in instances,
- Calling methods and sending messages to particular instances and their ancestors,
- Passing the instances as parameters.

The use-case diagram of required functionalities of the Class Viewer module is depicted in Figure 1.

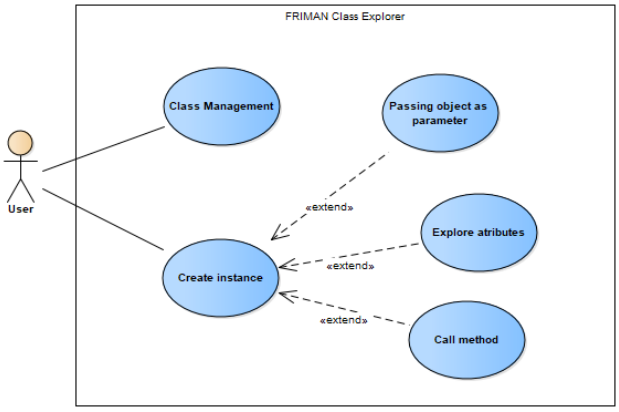

Figure 1: A use case diagram for the Class Viewer module

Based on specified system requirements, the module was developed and implemented in such a way that allows its users to manipulate classes by their own requirements. It provides also functionality to create new instances, modify them and delete the created objects. The instances can be also used as parameters of messages sent to other instances or as other objects constructor parameters.

Since the FRIMAN software development tool has been implemented in JAVA programming language, one of the very important tools used in the Class Viewer module implementation was the Reflection API.

Reflection is a feature in the Java programming language that allows an executing Java application to examine or "introspect" upon itself and manipulate internal properties of the program. It is possible to get the names of all class fields and to display them. The ability to examine and manipulate a Java class from within itself may not sound like very much, but in other programming languages this feature simply does not exist. For example, there is no way in a Pascal or C program to obtain information about the functions defined within that program [13, 14].

Reflection API is symmetric, that means that if we hold an object of a class, we can analyze its inner parts and equally, if we have one of its inner parts, we are able to find out, which class declared this part. This way, we can move in both directions from a class to a method, to a parameter of a class, etc. One of the interesting applications of this

technology is to find out the most of relative dependencies between a given class and the rest of the system [15].

The Class Viewer module also cooperates with a file system. All configuration files for classes created in a concrete project developed by our application are saved in a suggested file system. The module creates class instances and these objects will be saved in operation memory. Instances enable us to call partial operations. The process can be formally described by the following activity diagram, which is depicted in Figure 2.

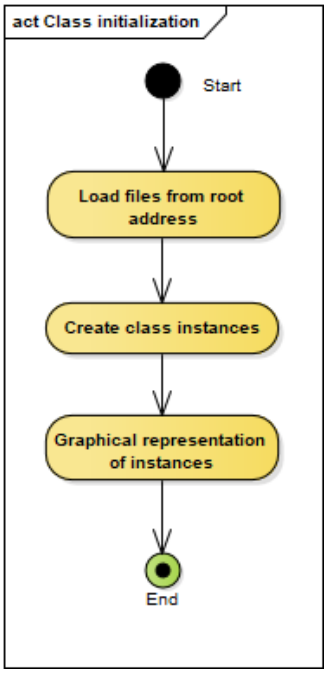

Figure 2: An activity diagram of the class initialization process

The first step of the class initialization process is reading the necessary configuration files from particular folder of the associated file system.

Based on loaded configuration settings, the next step consists of creating the class instances.

Finally, all created objects – class instances are visualized in a specific graphical environment.

Inner object states can be inspected as well as particular object attributes of any instances. Furthermore, the module enables to call any method not only of the associated object itself, but also any inherited method. Specifically, when a method return value is represented by an object, the system is able to create and save an instance of this object. This process is reported in Figure 3.

The first step of the method execution process is calling the method itself. Thus, the code of the method is executed. If the execution of the method finishes by returning

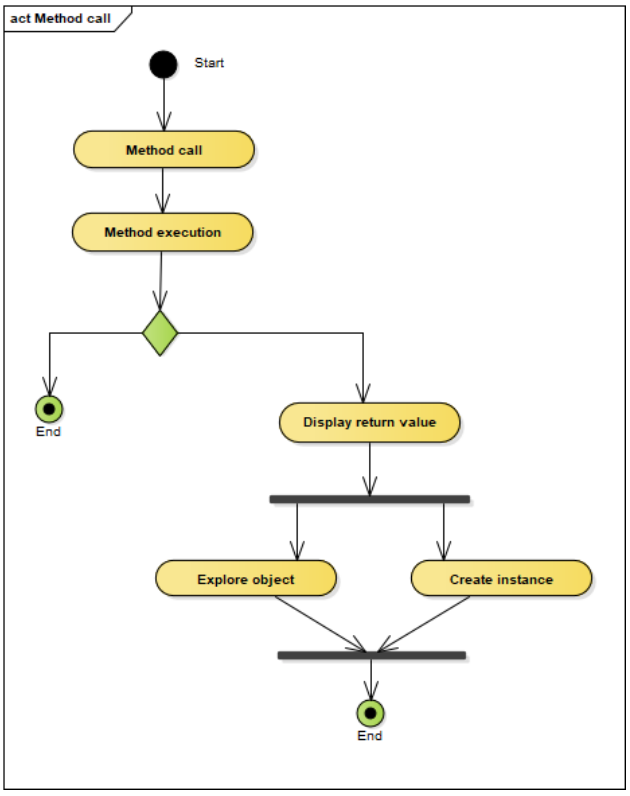

Figure 3: An activity diagram of the method execution process

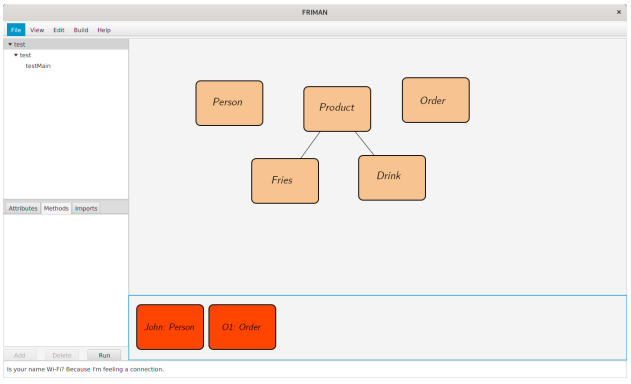

Figure 4: The main form of the FRIMAN system in the Class Viewer mode

any value or an object, then the return value is displayed in a separate graphical element.

If the return value was an object, the module enables to inspect inner parts of this object. This functionality is similar to those, which many BlueJ users are familiar with. Another possibility consists in using the return value as a new object instance to a specific graphical environment for further processing.

Except calling member methods, the user is able to explore selected instance and view values of its fields, as can be seen in Figure 5.

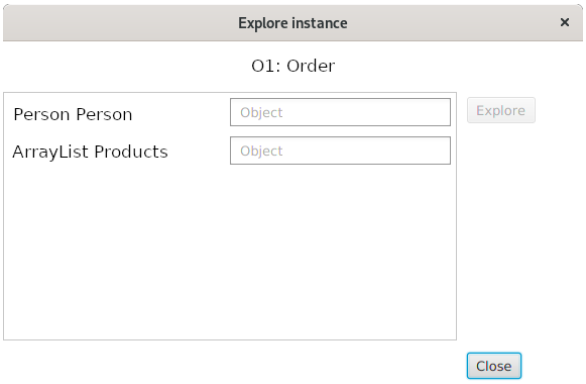

Figure 5: Example of instance exploration

The following Figure 4 depicts the main form of the resulting application in the Class Viewer mode. Obviously, the view mode can be easily changed to the Editor mode, which is reported in Section IV.

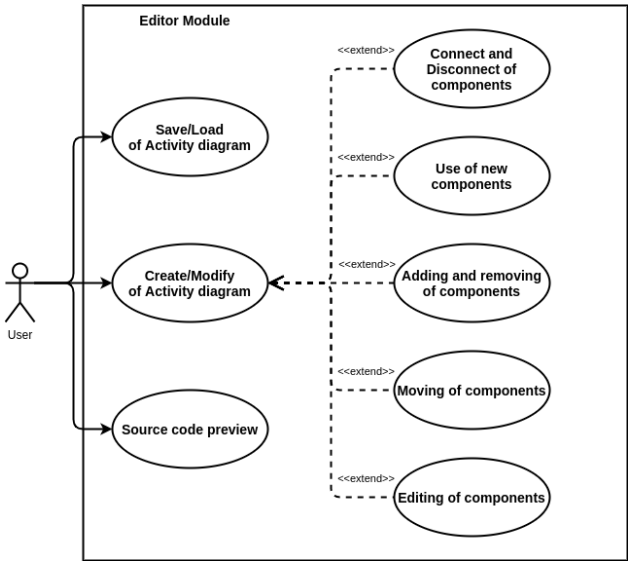

Figure 6: A use case diagram for the Editor module

4 The Editor module

The Editor module was suggested and incorporated into the FRIMAN application because of the necessity of creating and implementing methods and constructor bodies. Our vision was to use activity diagrams for this purpose and to make the process of making the algorithms interactive.

During the development of mentioned module, we have considered future system users, mainly students, beginners. The interactive way of programming is assumed to simplify the learning process of the object programming paradigm and the basics of creating algorithms at the same time. We believe that they will be able to understand the process of creating algorithms inside object methods using this simpler form without the necessity of programming language syntax knowledge.

The main requirements of this module can be seen in Figure 6. These requirements are not specific for OOP paradigm. Therefore the whole Editor module is independent on OOP and can also be easily used in other programming paradigms. In the context of the whole application is the Editor module focused on a specific paradigm. In case the whole application will manage a project with procedural programming paradigm, the editor will be used the same way as with any other paradigm.

Although Editor itself is independent on programming paradigm, each one has specific requirements for editor. In case of OOP, the activity diagram should offer the possibility to use attributes present in current class instance, keyword 'this', call of methods of this object as well as other methods, etc. The current version of the Editor module allows usage of the above mentioned: the source code is generated correctly and also correctly compiled. However it is not currently implemented in the most intuitive way, and users are not lead by Editor module to use these possibilities OOP. This requirement is planned for the next version of FRIMAN and hopefully will be ready when the application will be released.

Based on use cases described in Figure 6, requirements have been divided into the following tasks:

- Loading the library of graphical components, which are necessary for creating activity diagrams.
- Loading and displaying activity diagrams from a file.
- Creating the activity diagram using offered graphical components.
- Possibility of editing existing activity diagrams.
- Modification of text parts and scaling the graphical components.
- Generating the source code from the activity diagram to the JAVA programming language.
- Saving the final version of the activity diagram into a file.

The Editor module has been suggested and then implemented in such a way that it enables intuitive usage by its users. The module allows for the creation of activity diagrams making use of the set of pre-defined graphical components. These graphical elements are modifiable by the needs of users. The provided set of graphical elements offers all basic programming schemes (input, output, assignment, branching, cycles, etc.). Example of this

diagram can be seen in Figure 7. Available graphical elements can be seen in Figure 8.

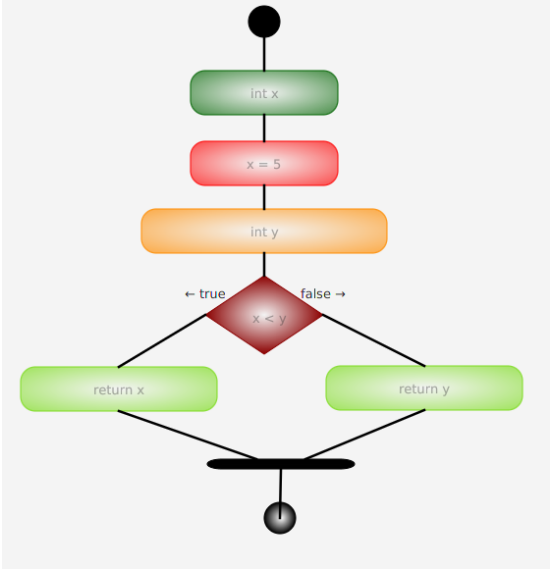

Figure 7: Example of an activity diagram in the Editor module of FRIMAN

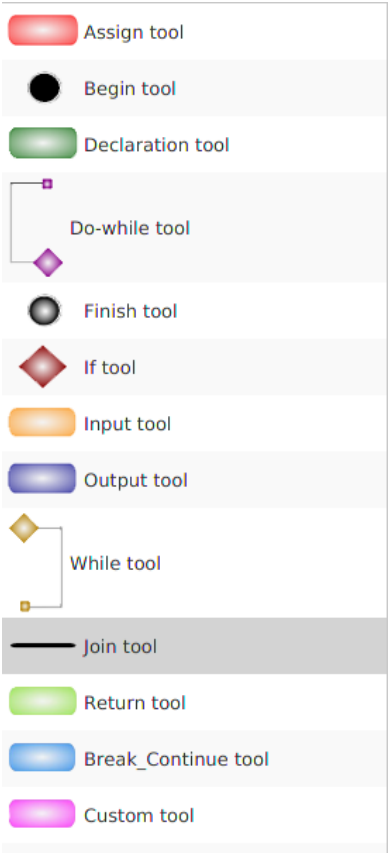

Figure 8: Toolbar for Activity diagram

```
{
    int x;
    x = 5;
    Scanner scanner = new Scanner(System.in);
    int y = scanner.nextInt();
    if(x < y){
        return x;
    }
    else {
        return y;
    }
}
```

Figure 9: Example of a source code generated from the activity diagram

It is also possible to edit the text information inside the graphical components according to the user's concepts and requests.

One activity diagram can consist of many different graphical components, which are joined by connections. The connection provides proper traceability between individual parts of the activity diagram. Any graphical component may be used multiple times according to the created algorithm. Furthermore, they can be hierarchically inserted into each other. If the connection between two parts is no longer needed, the module enables functionality to remove it.

During the process of drawing the diagram, the user is offered to switch to such a mode, in which the source code in JAVA programming language reflecting the activity diagram is being generated (Figure 9). Mainly the advanced FRIMAN system users will appreciate this functionality.

The set of created graphical components is sufficient to create any algorithm. The components have been chosen in such a way that the basics of any programming language are covered by this set. Thus, this functionality enables future extension of the existing system by the possibility of translating the activity diagram to different programming languages. The current version supports only the JAVA language.

When a user switches the code tab, the activity diagram is transformed into the source code in JAVA language. Simultaneously, the generated code is compiled by a JAVA compiler, which is a part of another module – Builder, which is currently being developed.

If there are any mistakes or errors, they are detected. The user should make corrections in an interactive way, e.g. in the graphical mode. In other words, it is not necessary to edit the programming source code in JAVA. When there are any changes performed in the activity diagram, the source code is generated and compiled again. This process can be repeated until obtaining the right result.

The example of a generated source code is depicted in Figure 9.

4.1 Source code generating

The main issue the editor module has to solve is how the activity diagram should be transformed into the source code. There are many ways how this can be handled. In our case we chose the simplest one. Each element that can be used represents a specific statement of code. For example, while element is generated in following way: firstly, keyword while is generated that is followed by logic condition. Next all connected elements are generated to the end point of while element (yellow square in Figure 8). All this is performed with language specific format, *i.e.* where language expects round brackets, round brackets are generated etc. Similarly, each element has its own way how it is to be generated. In other words, each element is responsible for its own generating into source code. Except that, each element has reference to element connected to it.

Thanks to this responsibilities distribution, we can start generating our source code from begin tool. After begin tool is generated, generating for its followers are called. This process continues to the point, where the whole method is generated. Please note, that this transformation is possible only in case, activity diagram does not contain any cyclic connected elements. Therefore some constraints for connection have to be implemented. In our tool we decide, each element can have exactly one parent element, *i.e.* element cannot have more than 1 in-going connection. Begin tool cannot have any in-going connection and finish element cannot have any outgoing connection. Source code is always generated only fro m begin tool. Thanks to these facts, it is not possible to cyclic connect elements and our transformation is working.

The described process is only used for generating source code of method. In order to run compiler and build whole project other parts have to be generated. For these purposes we decide to implement some business objects, that will hold information about various parts of project, the user is working on. Please note, that this is no longer the responsibility of editor module, but an explanation of this is also required to understand the whole process of generating source code. These business objects include package descriptors, class descriptors etc. Package descriptor is generated as folder in file system. Each package contains references to its classes. Class descriptor contains full information about one class, *i.e.* access modifier, implementing interfaces, fields etc. All of this information is also generated into corresponding file with exact syntax of java language. Class descriptor also contains information about its method descriptors. Method descriptor holds information as method access modifier, return type etc. Except that method descriptor also holds its activity diagram. When generating, method descriptor generates firstly its head and then calls generating of source code from activity diagram. This is also included when class is generating itself. One class holds many methods. If one element is changed, the whole class and all its components have to be generated again. This leads to a challenge. How to optimize this process and generate only some parts instead of a whole file.

5 Automatic updates

As the development of FRIMAN application will continue after release. Therefore a new requirement arises. It is a possibility to update an application. This can be handled in different ways:

- allow users to manually download and reinstall whole application,
- allow users to manually download patch and update application using it,
- allow users to update application using auto-update feature, *i.e.* application will handle everything, user only has to agree to update.

From these options we decide to pick the last one as it is most comfortable for users although it is most difficult to implement.

Implementing this requirement consists of several steps. The first one is implementing support for versioning of application as well as individual libraries. This will be useful also in further development processes because it will be possible to track bugs and features to a specific version of application. Versioning application will also allow us to manage changelogs and to plan development process better.

The second step is implementing server, where application will look for updates. Server has to provide information such as version of application and libraries – for FRIMAN application to be able to determine if update is necessary or not, hash of application and libraries - to ensure integrity of downloaded files and server has to also provide files for application to download.

And in the final step, all previously mentioned requirements have to be implemented also in application together the with possibility to update and restart itself if required.

Currently only the first step has been implemented, *i.e.* support for versioning. This information can be obtained from main ribbon → help → about and can be seen in Figure 10.

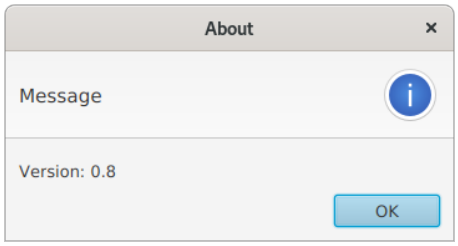

Figure 10: Dialog showing actual version of FRIMAN

6 FRIMAN web

In the previous section the second step to achieve automatic updates was mentioned as implementing server. Howeverc we decide to take it more complex and implement also webserver providing all the information about FRIMAN project. This web will provide the following:

- possibility to download FRIMAN,
- contact to developers (in order to report bugs, ideas for new features etc.),
- tutorials and demos

FRIMAN web is implemented using Laravel framework [16]. In current development phase this web provides possibility to download FRIMAN and contact authors. Currently web is available only in Slovak language. Of course, support for other languages is planned. Other possibilities, such as demos and tutorials will be implemented in next phase. This web will be deployed and available on site "www.friman.fri.uniza.sk" when this phase will be finished. Screenshot of current state can be seen in Figure 11.

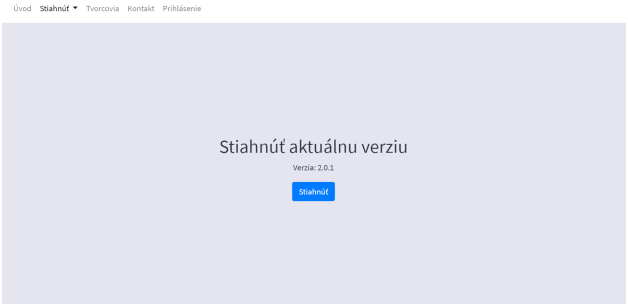

Figure 11: FRIMAN web

7 Management of development process

As was already mentioned before in this paper and also in paper [8], development of FRIMAN project is performed within a project course by a team of students of master degree in our faculty. Working on a project in this way brings several pros – mainly for students as they are forced to understand source code of other programmers, but, for the whole project there are also cons - the whole development process is slower then if the team was permanent. The number of students in one team varies from 2 to 8 people and each of these students is working 3 semesters on this project and during this time he has to:

- Study source code written by other students – for student to be able to implement new features and fix bugs, he have to understand already written source code.
- Collaborate with other developers working on this project – student is not working on project alone, so he have to discuss problems and solution with others, merge code with others etc.
- Work on assigned tasks – each student has assigned several tasks in one moment. He has to understand them and implement requirements without breaking other parts of FRIMAN
- Hand over the project to others – in order for the next team of students to be able to continue working on FRIMAN, it is necessary for the current team of students to explain current state, tasks etc. to the new one.

All the above mentioned will be also useful for students. In order to ensure and to help students to gain these experience, FRIMAN project is led by three experienced developers. They ensure the project is heading the way it should be heading, they are preparing tasks and managing work.

Collaboration on this project is performed in many ways:

- By weekly meetings of the whole team. In these meetings, each student presents what has been done, consults the problems with others and gets new tasks.
- Using GitLab [17] service to manage source code and project lifecycle.
- Using YouTrack [18] issue tracking software to manage tasks, plan work for development cycles etc.

In the future, we would like to extend our team by several testers. Currently, FRIMAN is tested only by the developers and leaders, but it would be helpful to delegate this work to testers and so let the developers focus more on other tasks.

8 Conclusion

This paper was focused on describing the current development state of FRIMAN application. This project originated at the Faculty of Management Science and Informatics at the University of Žilina to simplify learning of programming for non-programmers and for programmers to gain experience in developing a bigger project. The system consists of several cooperating modules. In this paper, the current versions of the Class Viewer and the Editor modules were presented as well as a description of future plans for this project. Class Viewer module allows users to create class instances and call methods using Java reflection API. Editor module provides a possibility of creating method bodies using activity diagrams instead of writing the code.

During the development process, many different challenges for future work on the system have occurred. First, we need to finalize the implementation of the missing modules. Then, the cooperation among modules may need to be resolved. The biggest challenge from our point of view is the Debugger module, because it should be done also in a graphical way. Another research topic that needs to be solved in the future work consists in optimization of the code generating process as mentioned in previous sections.

Currently, several new requirements have arisen during the development process. Therefore, features to automatic updates and a FRIMAN web server were introduced.

This paper also briefly describes management of work on FRIMAN. This project is developed by master degree students within the project course. As students are working on this project only for a limited time and new students come every year, it is necessary to organise work on FRIMAN in such a way, which enables the project to proceed.

Based on reported results we can conclude that the FRIMAN system represents a very useful tool for beginners in any object programming subjects.

References

[1] Donald E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition*. Addison-Wesley Professional, jul 1997.

[2] B. Stroustrup. What is object-oriented programming? *IEEE Software*, 5(3):10–20, May 1988.

[3] Ole-Johan Dahl. The birth of object orientation: the simula languages. In *From Object-Orientation to Formal Methods*, pages 15–25. Springer Berlin Heidelberg, 2004.

[4] Axel Schreiner. *Object-Oriented Programming With ANSI-C*. Axel T. Schreiner / Lulu, sep 2011.

[5] Eugene Kindler and Ivan Krivy. Object-oriented simulation of systems with sophisticated control. *International Journal of General Systems*, 40(3):313–343, April 2011.

[6] Howell Jordan, Goetz Botterweck, John Noll, Andrew Butterfield, and Rem Collier. A feature model of actor, agent, functional, object, and procedural programming languages. *Science of Computer Programming*, 98:120–139, February 2015.

[7] E. Parso et al. Friman. *Central European Research Journal*, 2:70–76, 2016.

[8] Peter Sedlacek and Monika Vaclavkova. Tool for supporting education process in information technology. In *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, November 2018.

[9] Jozef Kostolny and Monika Vaclavkova. Learning system friman. In *ICTERI*, 2017.

[10] Bluej. http://www.bluej.org/. [Accessed 22-Oct-2019].

[11] Intellij idea: The java ide for professional developers by jetbrains. https://www.jetbrains.com/idea/. [Accessed: 22-Oct-2019].

[12] Apache netbeans. https://netbeans.org/. [Accessed: 22-Oct-2019].

[13] Using java reflection. https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html. [Accessed: 22-Oct-2019].

[14] J. Jenkov. Java reflection tutorial. http://tutorials.jenkov.com/java-reflection/index.html. [Accessed: 22-Oct-2019].

[15] C. Mcmanis and C. Mcmanis. Take an in-depth look at the java reflection api. https://www.javaworld.com/article/2077015/take-an-in-depth-look-at-the-java-reflection-api.html. [Accessed: 22-Oct-2019].

[16] The php framework for web artisans. https://laravel.com. [Accessed: 27-Feb-2020].

[17] Gitlab. https://about.gitlab.com. [Accessed: 28-Feb-2020].

[18] Youtrack. https://www.jetbrains.com/youtrack/. [Accessed: 28-Feb-2020].