FlowUmi, a Programming Language that Mitigates the Usage of Hard-Coding

Edwin Puertas, M.Eng, Iván Baños, Ing, and Juan Carlos Martinez-Santos, Ph.D
Universidad Tecnológica de Bolívar, Colombia, epuerta@unitecnologica.edu.co, ivanalejandrobanos@gmail.com, jcmartinezs@unitecnologica.edu.co

Abstract– This paper presents FlowUmi, an activity diagram-based programming language that mitigates the use of hard coding. The main problem that motivates this project is the difficulty of learning best coding techniques. Comprehend and design algorithms are not trivial tasks to everyone. That is because it involves a set of skills that are not well developed in every person. These skills can be classified in two groups: the cognitive ones, which focus on the attitudes of the students, and the procedural ones, which focus on the way of doing. In this work, we focus on the last group by abstracting the code into activity diagrams. By doing activity diagrams instead of hard-coding (writing code), students focus on the design. It also makes the understanding of examples and patterns easier than in the hard-coding way. Traditional programming languages, used in teaching, separate the design and the coding in two different stages. It forces the student to take two courses, one to develop design skills and other to testing the algorithms by writing, compiling, and running code. On the other hand, our approach allows the student to test their designs directly from the graphical design of an activity diagram. There are two main contributions of this project. One is to design the flow chart language and all its components, and the second is to design its compiler. The language provides the basic control structures needed for procedural programming, which is the kind of programming that is commonly taught in freshman's courses. A prototype of FlowUmi, developed in Java, was tested with simple algorithms that were written without a single line of code. Currently, we are working on making FlowUmi available to everyone. The main idea is to put it under test and get feedback about usability, future features, and suggestion to making it better.

Digital Object Identifier
(DOI):http://dx.doi.org/10.18687/LACCEI2016.1.1.215
ISBN: 978-0-9822896-9-3
ISSN: 2414-6390

FlowUmi, a Programming Language that Mitigates the Usage of Hard-Coding

Edwin Puertas, M.Eng, Iván Baños, Ing, and Juan Carlos Martinez-Santos, Ph.D
Universidad Tecnológica de Bolívar, Colombia, epuerta@unitecnologica.edu.co, *ivanalejandrobanos@gmail.com*, jcmartinezs@unitecnologica.edu.co

Abstract*– This paper presents FlowUmi, an activity diagram-based programming language that mitigates the use of hard coding. The main problem that motivates this project is the difficulty of learning best coding techniques. Comprehend and design algorithms are not trivial tasks to everyone. That is because it involves a set of skills that are not well developed in every person. These skills can be classified in two groups: the cognitive ones, which focus on the attitudes of the students, and the procedural ones, which focus on the way of doing. In this work, we focus on the last group by abstracting the code into activity diagrams. By doing activity diagrams instead of hard-coding (writing code), students focus on the design. It also makes the understanding of examples and patterns easier than in the hard-coding way. Traditional programming languages, used in teaching, separate the design and the coding in two different stages. It forces the student to take two courses, one to develop design skills and other to testing the algorithms by writing, compiling, and running code. On the other hand, our approach allows the student to test their designs directly from the graphical design of an activity diagram. There are two main contributions of this project. One is to design the flow chart language and all its components, and the second is to design its compiler. The language provides the basic control structures needed for procedural programming, which is the kind of programming that is commonly taught in freshman's courses. A prototype of FlowUmi, developed in Java, was tested with simple algorithms that were written without a single line of code. Currently, we are working on making FlowUmi available to everyone. The main idea is to put it under test and get feedback about usability, future features, and suggestion to making it better.

I. Introduction

Nowadays, there is a certain apathy from the students to courses related to computer science. The biggest companies in the technology business are worried, and we see that the owners are making big donation and racing campaigns in pro of studding something related to computer science. Students also have problems in understanding concepts like the design an algorithm, subdividing into smaller and simpler pieces of code, hypothetical error situations and so on. Even with concepts like variables, data type and memory addressing, because there is no representation of these topics in the real world [1]. Studies that focus on finding the roots of these problems are divided into two groups: the cognitive problem and the procedural problem. Both are trying to find how to solve the problems in the learning of programming. The cognitive problem focuses on the attitudes of the students, their motivation, beliefs, and ways of studying. The procedural one tries to categorize the topics and focuses on find the way of learning the harder programming topic.

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2016.1.1.215
ISBN: 978-0-9822896-9-3
ISSN: 2414-6390

One of the problems shows up while a person is choosing whether to study or not programming related careers is the bad reputation that programming courses have earned. People think these courses are hard or boring, because of the though that everyone who studies programming is a nerd who does not go out of their homes or because is needed a big knowledge in math or logical reasoning [1]. Besides, the fact that a person spends 10 years becoming an expert in programming can influence a person to not studying programming because it is too much time to spend before having a good position or salary [2].

The purpose of the present study is to design and implement a programming language as show figure 1. Our approach is based on a graphic technique to make easier the comprehension of algorithm concepts eliminating the hard-core and facilitating the process of learning to program. Based on the best practices of software engineering, we develop a graphical interface in a web environment, so that students design their algorithm using activity diagrams. In addition, they are able to validate the solution proposed by the simulation algorithm using test cases. The purpose of this document is located to provide a better way to learn how to program without using code-hard. This programming language has three main elements: interface design and simulation, design language, and compiler design.

This paper is organized as follows. Section II contains the motivation and the problems faced students first year of computer science and engineering. Section III describes details of FlowUmi, and Section IV shows our preliminary results. Section V discusses related work about the problems learning programming language and tools for development application. Finally, Section V presents the conclusions and future work.

II. Motivation

Students of computer science and end careers have trouble with abstract concepts such as design a solution to a problem, subdivide a code into smaller pieces and simpler code, think of error scenarios and test for purposes and find errors.

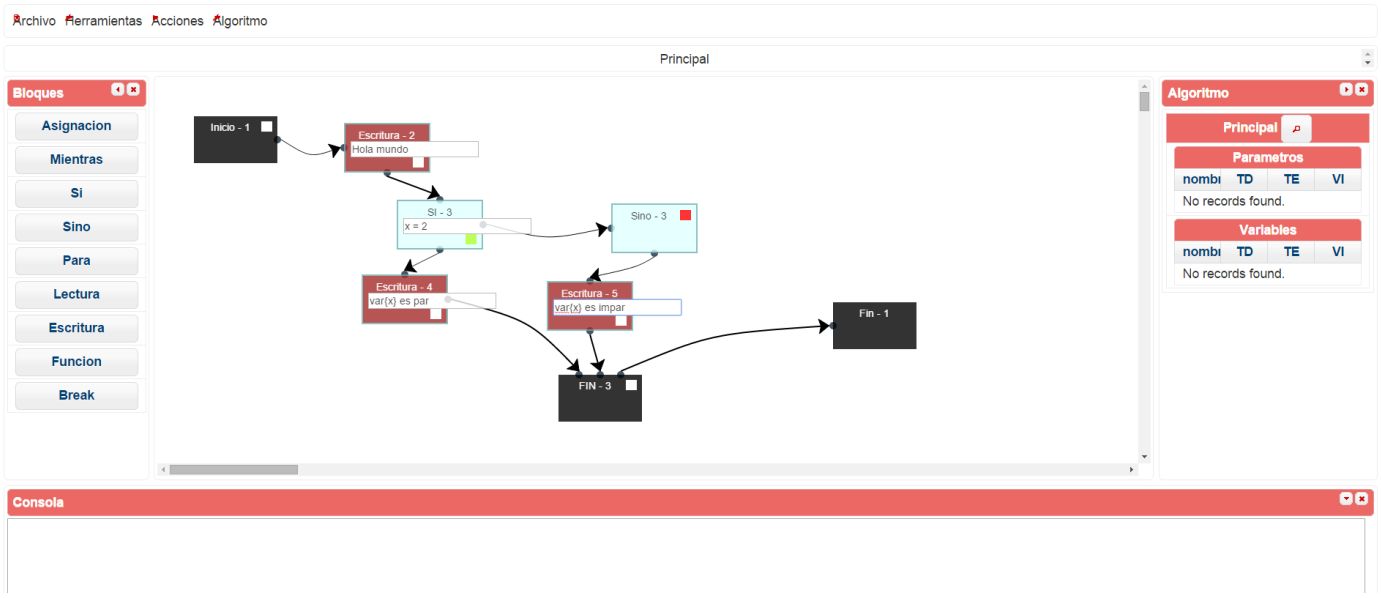

Figure 1. Example of an algorithm using FlowUmi

data types, or memory addressing is hard because they do not have a representation in the real world. Several authors have tried to find the reason why this phenomenon occurs. They also have tried to find the way to attack the problems. There are studies using tools, proposing sample courses for study, or mixtures between them to solve the problem, yielding results. Based on these investigations [1] [3] [4] [7] and [5], we will build a useful application for learning to program in a good way and taking into account best practices.

One of the problems that arise when enrolling to a computer science degree is the bad reputation that has earned programming courses. Students believe that these courses are difficult and boring. They think that all who study computer science is a ”nerd” who does not leave his homes. Moreover, students believe they need extensive knowledge in math and logical reasoning [1]. Another problem is the fact that person becomes an expert programming in 10 years of study [2]. This definitively can influence anyone not to take this career. This will determinate the time needed to earn a well paid or have a good job.

Students starting programming courses, usually try to solve problems without a plan, with limited usage of flow charts or pseudo-code. Because of this, they become frustrated. They do not know what to do when an error occurs. They do not know if the logic implemented in the program is or not appropriate, and they do not know how to identify these problems. They merely have superficial knowledge of the program language used, and they usually create their programs line by line. Sometimes, they are more interested on solving syntactic problems, why the compiler complies, instead of solving the problem itself. Therefore, students do not progress much in an introductory programming course.

Our system consists in web application of courses when instructors can post assignments for the students with pre-established case of study and a set of input and output data to test and give feedback. Every assignment consists in solve a problem with an algorithm. And it also has a free window to program anything desired. In this section, we explain the whole system. First, it is explained the language design, and then the compiler. The last part explains the case study activity requirement.

A. Language Design

The FlowUmi language is structured and support functions. It is based on activity diagrams. However, it has its own extension for different types of sentences. Moreover, because we are eliminating the hard-coding, the way how the sentences are structured in the language has to be as simple as they can be.

The variables are declared outside the flowchart to make it easy to identify. FlowUmi has three different types of variable: normal variables, arrays, and matrices. It also has five primitive data types: float, integer, strings, chart, and logic type. The language is strong typed, so the variables never change its type while running the algorithm. It will help to avoid errors that involve type-checking. It also makes variable declaration easy because they are concentrated in only one spot.

The types of data references in FlowUmi can generate references to variables, in other words it allows the user to refer to imperceptible way references to memory locations. So if the value of that variable value in memory changes also will be affected. During the implementation of language, we had to specify what type of variable could appear when declaring a data type and how they are declared according to their type of

variable that these belong an example of this is the result obtained in Figure. 2.

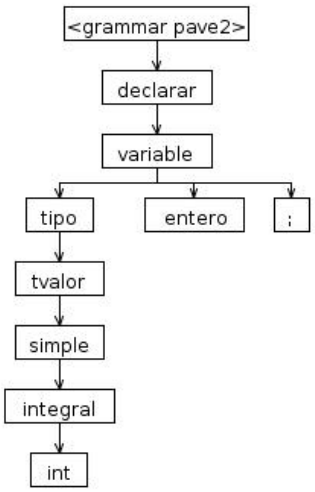

Figure 2. Tipos de datos - FlowUmi grammar

The following illustrates how the rules defined for types in FloUmi.

Grammatical production:

Y = {No terminals, terminals, Productions, Initial}

No terminals:

[*Variable type, tvalor, simple, comprehensive, float, decimal, number, boolean, enum, block*]

Terminals

[*Int, unit, short, ushort, long, ulong, byte, sbyte, float double, long double, decimal, bool, enum, const, class, interface, delegate, dynamic, object, string*]

Productions

```
variable ->
    constant tipo NOMBRE PUNTOYCOMA
    | constant simple NOMBRE IGUAL numero PUNTOYCOMA
    | constant tReference NOMBRE IGUAL (NOMBRE| "NOMBRE")  PUNTOYCOMA
    | tipo NOMBRE bloque PUNTOYCOMA
    | tipo NOMBRE PUNTOYCOMA
    | tipo NOMBRE (',' NOMBRE)* PUNTOYCOMA
    | tipo NOMBRE (',' NOMBRE)* IGUAL NOMBRE (',' NOMBRE)* PUNTOYCOMA
    | tipo NOMBRE (',' NOMBRE)* IGUAL numero (',' numero)* PUNTOYCOMA
    | tipo NOMBRE IGUAL NOMBRE PUNTOYCOMA
    | tipo NOMBRE IGUAL numero PUNTOYCOMA
    | 'char' NOMBRE IGUAL  'CHAR' PUNTOYCOMA
    | 'char' CCI CCD NOMBRE IGUAL 'new' 'char' CCI (numero) CCD PUNTOYCOMA
    | tipo CCI CCD NOMBRE IGUAL 'new' tipo CCI (numero|NOMBRE) CCD PUNTOYCOMA
    | tipo CCI CCD NOMBRE IGUAL bloque PUNTOYCOMA
     tipo -> tvalor
     tvalor -> simple
     simple -> integral|flotante|decimal|booleano|enum
     integral->     'int'|'uint'|'short'|'ushort'|'long'|'ulong'|'byte'|'sbyte'
     flotante -> 'float'|'double'|'long double'
     decimal        -> 'decimal'
     booleano -> 'bool'
     enum -> 'enum'
     bloque->LLI ((NOMBRE (',' NOMBRE)*)| (numero(',' numero)*)) LLD
     numero->INTEGER|FLOAT
     constant->'Const'
     tReference->'class' |'interface' |'delegate' |'dynamic' |'Object' |'string
```

Initial

[variable]

Every control structure has a shape and an identification number. There are two especial shapes "***begin***" and "***end***". Both are represented as circles, one is entire black and the other one has a black circle inside a bigger one. All algorithms start with a "***begin***" shape and end pointing to a "***end***" structure.

The "***assignment***" structure is a squared box where we put an assignment sentence. It has to have an existing variable to be assign and an expression. Because the language is strong typed, the expression is compiler-time evaluated to corroborate that both the expression and the variable match.

The "***if***" structure is the classic rhombus used in activity diagrams to make a decision. It takes only one exit transition for true. It has to follow the paths through the "***if-end***". The "***if***" structure always starts with a rhombus shape that includes the condition, and ends with another rhombus shape (it is the "***if-end***"). They both have the same identification number. The "***if-else***" structure is the identical rhombus for the "if" structure. The difference between the two of then is while the "if" structure has only one exit for true the "***if-else***" structure has two exit transition, for both true and false options.

The "***switch***" structure is also a rhombus which contains an expression like the last two structures. However, it has no limit for exit transitions. Each of these transitions goes to a square which has each case value.

The "***while***" structure has two squares. One is the finish condition, and other one limits the sentences inside the "***while***" structure. The last square has a transition that goes to the first square. This is to emphasize that at the end of the "***while***" structure returns to the beginning of it.

The "***for-each***" structure has a similar structure of the "while" structure. The main difference is that the "***for***" receive as an expression a group of elements in an array. You also have to give a label which every value gains in each iteration.

The language has the sentences "***break***" and "***continue***". Each one is a square box which has those words. The "***break***" sentence will finish the most-internal loop, while the "***continue***" sentence will go to the next iteration of the most internal loop. You also can give them a number, and it will break the loop with that identification.

The "***in***" and "***out***" instructions are instructions that will be used to receive and print data respectively. Both are also square boxes. The "***in***" box will receive the name of a variable, and the "out" box will receive a message. If a user wants to print the value of a variable, he or she will have to specify the variable's name in the message with a (***$***) sign.

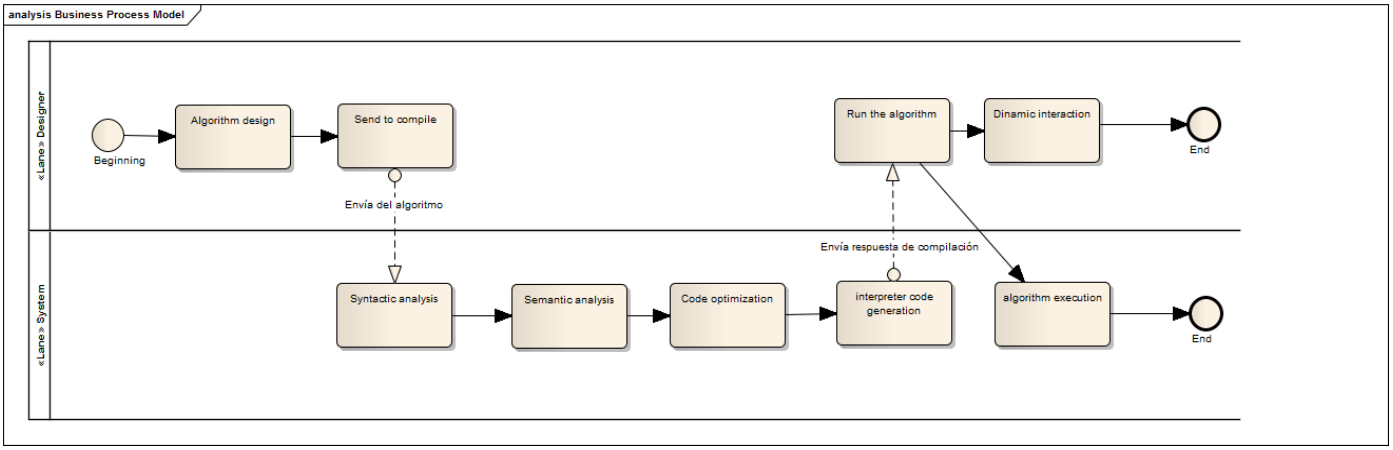

Figure 3. FlowUmi compilation process

The language supports ***functions***. If a user wants to call a function in an expression, he or she will put the variable's name followed by its parameters enclosed with parentheses. Moreover, if the function returns no value, the user can use a function square box. In it, he or she has put the name of the function and its parameters enclosed in parentheses. Expressions allow us to declare function calls, make assignments to values to variables, classes generation among other things. in Figure 3 and 4 you can see two examples of these actions.

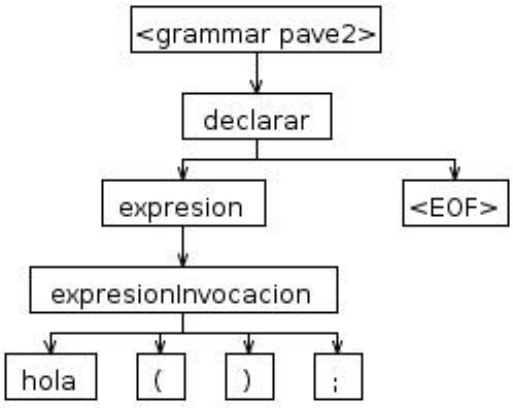

Figure 4. Expression: called - Flowmi grammar

The following illustrates how the rules defined for expression in FloUmi.

Grammatical production:

Y = {No terminals, terminals, Productions, Initial}

No terminals:

[*expresion, expresioncondicional, expresionAditiva, expresionInvocacion, expresionLogica, expresionBooleana, expresionIgualdad, expresionRelacional, expresionAditiva, expressionMultiplicativa, expresionPotencia, expresionCerrada, primaria*]

Terminals

[*Int, unit, short, ushort, long, ulong, byte, sbyte, float double, long double, decimal, bool, enum, const, class, interface, delegate, dynamic, object, string*]

Productions

```
[
expresion-> expresioncondicional
          |   expresionAditiva
          |   expresionInvocacion
expresioncondicional-> expresionLogica
expresionLogica-> expresionBooleana (OR
                  expresionBooleana)*
expresionBooleana-> expresionIgualdad
((EQUALS|NOTEQUALS)expresionIgualdad)*

expresionIgualdad-> expresionRelacional (AND
                    expresionRelacional)*
expresionRelacional-> expresionAditiva
( (LT|LTEQ|GT|GTEQ) expresionAditiva)*

expresionAditiva->
       expressionMultiplicativa( (PLUS|MINUS)
       expressionMultiplicativa)*
expressionMultiplicativa->
       expresionPotencia ( (MULT|DIV|MOD)
       expresionPotencia)*
expresionPotencia-> expresionCerrada  (POW
                    expresionCerrada)*
expresionCerrada-> primaria
    |   NOT primaria
    |   MINUS expresionCerrada
 expresionAditiva->
       expressionMultiplicativa( (PLUS|MINUS)
       expressionMultiplicativa)*
expressionMultiplicativa->
       expresionPotencia ( (MULT|DIV|MOD)
       expresionPotencia)*
expresionPotencia->
       expresionCerrada  (POW expresionCerrada)*
```

```
expresionCerrada->
        primaria
    |   NOT primaria
    |   MINUS expresionCerrada
expresionInvocacion->
          NOMBRE PI (tipo (NOMBRE|valor))* PD
          PUNTOYCOMA
 primaria->valor
 valor->INTEGER   |FLOAT |NOMBRE
]
```

Initial

[*expresion*]

B. Compiler design

The compiler for FlowUmi is developed in Java with ANTLR [18] and it follows the steps represented in *Figure.3*.

Before the algorithm is compiled, it is transformed into a line by line code that represents the same algorithm. This language is the one that the compiler understands, and it does not alter the initial diagram.

The implementation of the compiler is made using ANTLR. ANTLR is a compiler of compilers. It has a structure to create lexers, parsers, and tree walkers. Those tools would be used to check for compilation time errors. The lexer verify the lexemes, and it classifies them and passes them to the parser. The parser checks if they are grammatically correct and create an abstract syntax tree (AST). Afterwards, the system walks through the AST doing the type-checking and generating a simpler code that will look like a machine code.

The lexer uses regular expression that are responsible for lexems recognition. A lexem is a word that belong to the language. The way is used in ANTLR is giving first an expression name, which starts with a capital letter followed by letters or digits, then an '=' symbol and then a regular expression. An example would be like this:

This lexeme represents the integer values:

```
INT : '(-)'?('0'..'9'+);
```

This lexeme represents the identifiers:

```
ID:('a'..'z'|'A'..'Z'|'_')('a'..'z'|'A'..'Z'|'
0'..'9'|'_')*;
```

This lexeme represents the keyword

```
MIENTRAS: MIENTRAS:'MIENTRAS';
```

The parser is a collection of rules that checks that the algorithm follows the language grammar. The parser rules are represented with free-context grammar rules in a extended Backus-Naur form. As in the lexer rules, ANTLR demands a name for every rule. In this case, the rule name starts with a lowercase letter. Inside the rule, we can put other rules as non-terminal symbols and lexemes as terminal symbols. Some examples are listed next.

principal: variablesDec INICIO sentencias* FIN;

This rule called principal begins with another rule that has the grammar for the variable declaration, followed by the keyword ***BEGIN*** (*INICIO*). Then, there is a list of sentences, and ends with the key word ***END*** (*FIN*). It has the basic structure of how is the body of the functions.

```
sentencias : asignacion | si | mientras | para |
lectura |escritura | breaks | llamado;
```

This rule represents all the instructions that the language has. We can see the rule for assignation, the control structures and the others.

The type-checking and the code generation are done by walking the resultant tree of the parsing step. The system walks the tree a first time to do the type-checking, and then it walks another time to generate the code.

We also design an interpreter to the resultant code, and it will be used to interact with the user while they are running an algorithm.

C. Application Design

The system as explained above consist in a course environment where a teacher can post activities, and student can solve them. There are three different roles with specific tasks, as specified in Figure 5.

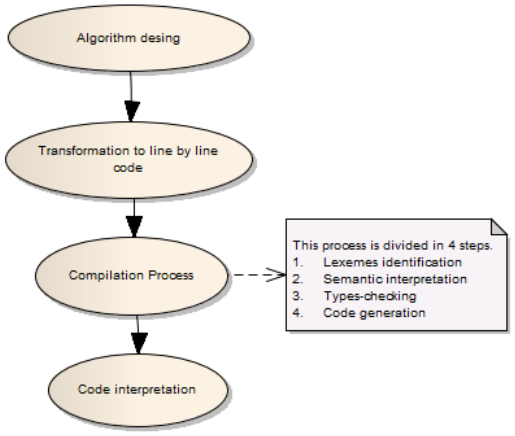

Figure 5. Use case diagram

The administrator is in charge of create, update and delete

courses. Besides, he can sing teacher and students up into the system and subscribe them into the courses. Other duty for the administration is that he has to be aware that the application is running fine by setting up the configuration parameters.

The role of the tteacher in the system is to post assignments. The assignment is an activity to the students. In an assignment the teacher post the title and the statement of the algorithm. The teacher can also give them some part of the code. Finally, the teacher can create some test cases where he puts the in and the out for a specific set of data. To grade the assignments the teacher create test cases and check if the student's algorithm give the same output with the same entries.

The student can only solve assignments leaved by teachers in the courses that he is subscribed. However, they will have a window to program whatever they want.

To sing in into the application the user should fill a form with the credential and depending on the role it would be redirect to a main page.

This framework was built with Java enterprise edition 6 for the business logic and JPA to communicate with the database. It uses a MySQL [19] database to persist the information and ANTLR to the compiler. To the web environment was used bootstrap and JQuery, using a Themeroller [20] theme.

Figure. 3 shows an example of an algorithm using FlowUmi. The programming Graphical user interface (GUI) is divided in 5 blocks. On the top is resembled the name of the actual function that is currently working. On the button there are an output console, when the results of an algorithm execution are showed, it also shows an error if exist. On the left there is a list of the different statements that can be used in algorithms. On the right panel there is the list of variables and functions. In the middle is the activity chart that represents an algorithm.

To add a sentence to the algorithm, the buttons on the left panel are used by clicking on them. This action will add the components of the selected sentence on the chart. A sentence can be erase by clicking on the close mark on the box.

To add a variable, there is a plus button besides the word variable on the top of the right panel. By clicking on it a dialog with a form is displayed. The fields to add a variable are: name of the variable, data type and the initial value, the variable will be added to the current function scope.

To add a function, there is also a plus button besides the word function, as in variables, it changes the fields of the form that are: name of the function, the list of parameters and the return data type.

If there is an error while compiling the algorithm it is showed on the output console. If after compiling there were no errors, a message would be posted on the output console saying that everything went well.

IV. Results

Flowmi allows students to design and simulate sequential programs in a simpler way, using activity diagrams, thus eliminating the use of hard code. This will cause the student to focus more on learning the basics of programming. In addition, you can simulate the behavior of its solution and export the solution in hard code in a high level language, a space also have to save your projects, to download, compile, execute.
The implementation of this programming language was developed by the following phases:

Phase I: a graphical user interface implementation will allow the user to draw his mental abstraction as solving computational problems using flowcharts.

Phase II: compiler implementation will take a graph a flowchart and transform every element in a structure that allows events simulation algorithm under test.

Phase III: implementation will of an interactive system that allows graphically manipulate the states of the algorithm under test.

Phase IV: Evaluate methodologically as is the performance of the student.

This fourth phase has not been carried Just because the language is in testing phase, besides being necessary to the development of this phase right people in the theme of pedagogy.

V. Related work

This section briefly reviews proposals that can be applied to obtain the best learning for students at the first year of computer science and engineering. Although are many tools, the main solutions that occurred in the beginning to this problem are the use of animation to minimize the difficulties of students. Others only show the lively performance of a specific algorithm. However, students cannot make any changes or test different algorithms.

Alice is a development environment for drag and drop, for novice programmers. The primary objective is to prevent students commit syntax errors. Alice was originally designed as a tool to improve the increase in risk-capacity undergraduate computing students to succeed in first-year courses in racing computer science and related fields. It is currently used in hundreds of schools and colleges secondary [8], [9].

Scratch is a learning environment programming language via a simple graphical interface. Scratch Programming is based on a metaphor of building blocks influenced by previous systems as Lego Blocks, which allows beginners to learn to write correctly programs. It is notable for handling and multimedia-based media that are of interest in youth programming, allows the creation of animated stories, games, and interactive presentations [10]–[12].

Greenfoot is a highly specialized for developing interactive graphics applications in educational environment. It is based on the use of the Java programming language. The secondary school students can interactively develop interesting, such as games and simulations. They can implement algorithms as quickly and easily as they learn fundamental programming concepts [11], [13].

One of the principal tools is SICAS. SICAS simulates the algorithm and shows the results using animation. Moreover, the students can analyze how the algorithm works in detail, at their pace identifying and fixing eventual errors. SICAS does not include theoretical contents, instead it consists in an experimentation environment that permits detect errors and correct them, and learning based on activities. There were some tests made to verify that students who used SICAS have built better algorithm than students who did not use SICAS [14].

According to [2], [15], models are the base of the knowledge. If the instructor does not explain well the concept to the students, they can form the wrong model representation out of it. Authors in [2], [16] describe the first year of programming study the students creates the model to solve problems and the programming language is introduced, in particular, cases. One of the principal solutions to the problem was the use of animation to reduce to the minimum the difficulties of the students. The first tools were only made to see an algorithm built and running. By watching the algorithm run the students starts to create the model in their heads [15].

Another issue is when the students try to acknowledge a language syntax. In many cases, the differences with the natural language can generate problems. An example referenced by [6] is the beginners think about the while buckle is evaluated in every expression and no only one time per iteration.

The problems with those tools are the focus for what they were developed. They focus more on teaching the concept by examples or with animations and are good, but for a level of teenagers. Besides, they take a lot of responsibilities while doing algorithm. The idea is to do an application that focuses in create the models by programming but didn't take the responsibility of doing to the students.

CONCLUSIONS

This paper describes a proposal to improve strategies in the learning process of mitigating the hard programming code. The main purpose is to use a programming language supported in an activity diagram to get a problem solved algorithmically. Thus, the learning process aim is to solve a problem and not to know a programming language high level.

Thanks to FlowUmi, the new strategy facilitates student "learns to learn", allowing them to be part of learning, rather than learning only what is told to learn.

A prototype was built in a first stage, with the established functionality. The application allows to create activity diagram based algorithms to represent algorithms.

REFERENCES

[1] Esteves, Micaela, Benjamim Fonseca, Leonel Morgado, and Paulo Martins. "Improving teaching and learning of computer programming through the use of the Second Life virtual world." British Journal of Educational Technology 42, no. 4 (2011): 624-637.

[2] Winslow, Leon E. "Programming pedagogya psychological overview." ACM SIGCSE Bulletin 28, no. 3 (1996): 17-22.

[3] Meerbaum-Salant, Orni, Michal Armoni, and Mordechai Ben-Ari. "Learning computer science concepts with scratch." Computer Science Education 23, no. 3 (2013): 239-264.

[4] Fleury, Ann E. "Encapsualtion and reuse as viewed by java students." In ACM SIGCSE Bulletin, vol. 33, no. 1, pp. 189-193. ACM, 2001.

[5] Lister, Raymond, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney et al. "A multi-national study of reading and tracing skills in novice programmers." In ACM SIGCSE Bulletin, vol. 36, no. 4, pp. 119-150. ACM, 2004.

[6] Ala-Mutka, Kristi. "Problems in learning and teaching programming." Codewitz Needs Analysis (2012).

[7] Eckerdal, Anna. "Novice students' learning of object-oriented programming." (2006).

[8] Resnick, Mitchel, Yasmin Kafai, and John Maeda. "A networked, mediarich programming environment to enhance technological fluency at afterschool centers in economically-disadvantaged communities." (2005).

[9] Cooper, Stephen. "The design of Alice." ACM Transactions on Computing Education (TOCE) 10, no. 4 (2010): 15.

[10] Maloney, John H., Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. "Programming by choice: urban youth learning programming with scratch." ACM SIGCSE Bulletin 40, no. 1 (2008): 367-371.

[11] Fincher, Sally, Stephen Cooper, Michael Klling, and John Maloney. Comparing alice, greenfoot scratch. In Proceedings of the 41st ACM technical symposium on Computer science education, pp. 192-193. ACM, 2010.

[12] Resnick, Mitchel, et al. "Scratch: programming for all." Communications of the ACM 52.11 (2009): 60-67.

[13] Klling, Michael. "The greenfoot programming environment." ACM Transactions on Computing Education (TOCE) 10.4 (2010): 14.

[14] Bravo, Crescencio, Maria Jose Marcelino, Anabela Jesus Gomes, Micaela Esteves, and Antonio Jose Mendes. "Integrating Educational Tools for Collaborative Computer Programming Learning." J. UCS 11, no. 9 (2005): 1505-1517.

[15] Robins, Anthony, Janet Rountree, and Nathan Rountree. "Learning and teaching programming: A review and discussion." Computer Science Education 13, no. 2 (2003): 137-172.

[16] Deek, F. P., McHugh, J. A., & Roxanne Hiltz, S. (2000). Methodology and technology for learning programming. journal of Systems and Information Technology, 4(1), 23-35.

[17] Roberts, Eric, Kim Bruce, James H. Cross II, Robb Cutler, Scott Grissom, Karl Klee, Susan Rodger, Fran Trees, Ian Utting, and Frank Yellin. "The ACM Java task force: Final report." In ACM SIGCSE Bulletin, vol. 38, no. 1, pp. 131-132. ACM, 2006.

[18] Parr, T. (2013). The definitive ANTLR 4 reference. Pragmatic Bookshelf.

[19] MySQL [Online] https://www.mysql.com.

[20] Themeroller [Online] http://jqueryui.com/themeroller