MDDT: Model Driven Development Tool as Aid for Learning Programming Fundamentals

Dy, Jeric Bryle
College of Computer Studies
De La Salle University
2401 Taft Avenue, Manila,
Metro Manila 1004
(632) 524-4611
jeric.exe@gmail.com

Laureano, Philip
College of Computer Studies
De La Salle University
2401 Taft Avenue, Manila,
Metro Manila 1004
(632) 524-4611
philip.laureano@gmail.com

Liu, Michael David
College of Computer Studies
De La Salle University
2401 Taft Avenue, Manila,
Metro Manila 1004
(632) 524-4611
michael_5569@yahoo.com

Syliongka, Leif Romeritch
College of Computer Studies
De La Salle University
2401 Taft Avenue, Manila,
Metro Manila 1004
(632) 524-4611
lairusi@gmail.com

Roxas, Rachel Edita
College of Computer Studies
De La Salle University
2401 Taft Avenue, Manila,
Metro Manila 1004
(632) 524-4611
rachel_roxas2001@yahoo.com

ABSTRACT
In conventional ways, students, who are learning how to program, struggle with learning the syntax conventions of a particular programming language more rather than learning how to program logically. As a solution, various institutions have incorporated flowcharting as the main tool in teaching fundamentals of programming. Nonetheless, a flowchart is only as good as a design unless implemented to certain a programming language. Hence, in the end, novice programmers still have to struggle with the syntax conventions of a particular programming language.

This paper presents a system in learning how to program easier by allowing the users to implement programs using flowcharts and showing their implementations in various programming language. By doing so, users are able to compare their implementation as to what it would look like in an actual programming language. Based on the survey conducted, the system shows potential in easing the learning curve of learning how to program. Currently, the system is still undergoing further development such as incorporating an interpreter in the system, incorporating an intelligent code generator for non-deterministic flowcharts and so on.

Keywords
language-oriented programming, model-drivel development, basic programming, flowchart, code-generator, translator

1. INTRODUCTION
Flowchart is "a means of visually presenting the flow of data through an information processing systems, the operations performed within the system and the sequence in which they are performed" [6]. It is one of the primary teaching tools used by different institutions in teaching their students the fundamentals of programming. Flowcharting is said to be an effective tool in teaching introductory programming as it visual [3] and majority of the people in the world are said to be visual learners [3].

Several tools have been developed by researchers and institutions over the past years in order to integrate flowcharts into their academe programs. Based on the surveys and tests conducted, integrating flowcharts in teaching programming fundamentals are very effective as students have shown promising results of improvement in formulating their solutions and coding it as well. However, for a flowchart design to be useful, it has to be converted into some target code.

2. Flowchart as Teaching Aid
Currently, various institutions have included flowcharts in teaching novice programmers how to program. Based on the feedbacks and performances of the students, it is said that flowcharting is an effective tool in teaching introductory programming as it visual in nature. This is because students these days are visual learners [3].

By allowing students to model their solutions using flowcharts, it breaks the students away from struggling in syntax conventions of a particular language [3]. In effect, students will have more time to concentrate on learning how to program logically instead of "struggling" how to program in a particular programming language.

3. Language Oriented Programming
Language-Oriented Programming (LOP) or also known as Domain-Specific Language (DSL) is a programming paradigm wherein domain-related functions or solutions are mapped to some general-purpose language (GPL) [4].

Based on the idea that LOP languages are mapped to GPLs, LOP therefore contains abstracted functions and codes of the GPL. By extension, it is also a subset of the GPL. Given the idea that LOPs are abstracted and subsets of GPLs, programming in LOPs therefore makes programming solutions faster as developers have to code less and still be able to produce the solution. Furthermore, by lessening the lines that one has to code for a certain program, developers are hence, bound to commit lesser mistakes in coding their solutions. In addition, since LOPs are "smaller" than GPLs, users will also have to explore less of the programming language and would immediately grasp the idea faster.

Flowcharts in the same manner, is an LOP. In this case, the domain of the flowchart is programming languages. It gives its users an abstracted form of programming languages allowing its users to concentrate more on how to design a program logically without having the deal with the issues on syntax conventions of programming languages.

A flowchart may have various equivalent codes when translated. However, based on the implementations done in RAPTOR, Visual Logic and FLINT, the developers of these systems devised their own way to force the translation to one-to-one. An example of which can be found in Visual Logic, wherein the system included a modified version of a flowchart which now incorporates the for-loop and the while-loop consequently enabling the one-to-one conversion of a flowchart to code.

4. Flowchart Interpreter

Flowchart interpreter is a tool that allows flowcharts to be simulated [1]. It allows its users to model and receive immediate feedback regarding their program without the need to translate their flowcharts to codes.

5. Code Generators

Using the keywords "Code Generators" in Google, there are indeed many code generators present in the Internet alone – CSS generators, JavaScript generator, HTML generator and so on. However, the one we are concern with for this project is the UML to source code generator. Currently, it is one of the most used code generators in the industry to increase productivity. It is also used as one of the teaching aids for students who are starting to learn how the basics of Object-Oriented Programming (OOP) [8].

[5] proposed a UML to source code generator system called Metamodel Engine (ME). As shown in Figure 1, given a development cycle, ME is found between the formulation of the design and the coding phase.

Figure 1: Position ME in the Development Cycle [5]

In converting the UML to source code, ME uses XMI standard format as most of the UML tools support this standard. Given the XMI, ME then imports the XMI file and converts it to the target code based on the rules of conversion incorporated in the system in the form of a DLL [6].

Similarly, another UML to code generator developed by [7] takes the same disposition in the development cycle of a program. However, instead of converting the flowchart to XMI files, before converting it to the target code, the system directly converts the flowchart based on the XML rules included in its repository.

6. System Testing

Based on the testing conducted by [2] for RAPTOR and by [3] for FLINT, the developed system is mandated in the introductory courses of their respective institutions. Then, at the end of their academic term, the effectiveness of the system is evaluation by observing the students' performance based on the examinations conducted and comparing it to the previous academic terms. In addition to observing the performance of the students, RAPTOR was also evaluated by means of letting the students answer some survey questions at the end of the course. Table 1 shows the survey question used for RAPTOR.

Based on the survey questions of RAPTOR, the developers evaluated the usability, usefulness, effectiveness of the system [2]. The survey conducted was only confined to finding out the effectiveness of it only in the modeling or designing level. Since the aim of this research is to help find out how useful it is to be able to let the users see the code version of the flowchart, survey questions in Table 1 are not as helpful for this research; however, some of the questions that will be formulated for the evaluation of the system will be patterned in similar manner.

Table 1: RAPTOR Survey Questions [2]

I had few problems learning how to use RAPTOR to create my programs.
I had few problems getting my programs to run once I had created them.
I found the Help System in RAPTOR to be useful.
I used the Help System in RAPTOR frequently.
RAPTOR helped me to develop and improve my problem solving skills.
RAPTOR helped me to better understand how computer programs operate.
I enjoyed programming in RAPTOR.
Being able to view the contents of variables helped me to test and debug my programs
My teacher gave me enough instruction on using RAPTOR so that I could use it effectively to create programs.

7. MDDT System

7.1 System Overview

The system developed is a tool that aims to aid users how to program from one language to another. Users are to utilize a model-driven development tool, flowchart, wherein they can code programs and allow the system to translate it to different programming languages base on the rules available for the system. Java was the primary tool used in the development of the system.

7.2 System Architecture

Figure 2 illustrates the system architecture of MDDT. As shown, the system is composed of an IDE, Syntax Tree Converter and Code Generator.

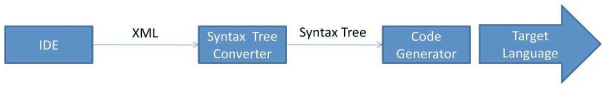

Figure 2: System Architecture

7.2.1 Integrated Development Environment (IDE)

The IDE serves as an environment wherein users will be able to interact with the system such as saving and loading of flowchart, invoking the system to generate code, view the generated code and so on.

In addition, using the IDE component of the system, users will input their implementation in the form of a flowchart through the IDE. Despite the flowchart being envisioned as a drag and drop type of programming, nonetheless, users should follow the flowchart conventions. One may argue that the convention of the flowchart has the potential to limit its users syntactically; however, "the flowchart convention is not as limiting as the convention of a programming language" [6].

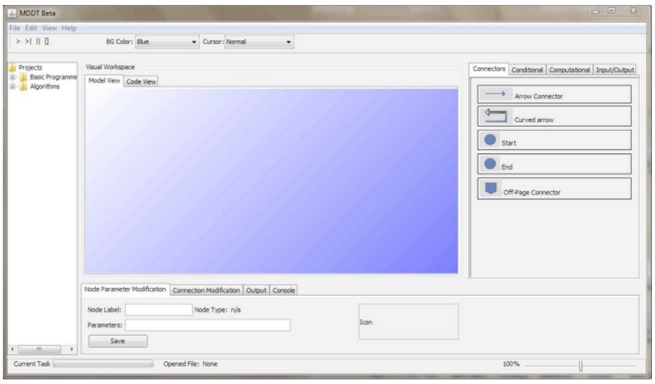

Figure 3: System Integrated Development Environment

7.2.2 Syntax Tree Generator (STG)

Based on flowchart input of the user, the system converts the flowchart to a syntax tree. The generated syntax tree from the flowchart will be used as a basis of conversion of the flowchart to the target programming language by the code generator.

7.2.1 Code Generator (CG)

The code generator uses the syntax tree generated by the STG alongside with a set of formulated rules as its input (Please see Figure 5 for the subset of rules in generating codes in C). The system then produces the target source code of the user's program and returns it to the IDE so that the user can view their implementation on the target language. Figure 4 shows the flowchart input of the user, Figure 5 shows the rules (a subset of the rules) triggered by the sample input in Figure 4. Figure 6 shows the generated code of the flowchart in the target code, C.

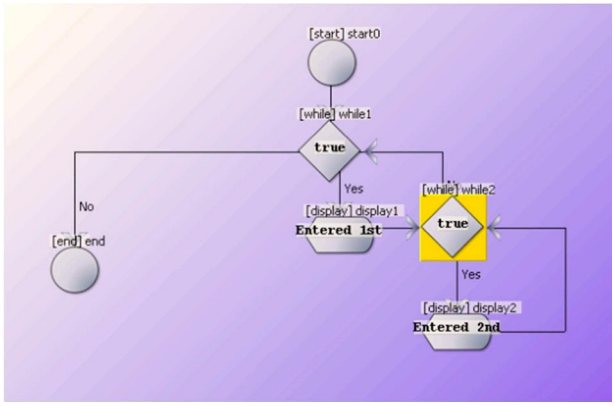

Figure 4: Sample Flowchart

```
<body>
//imports

//methods

#include <stdio.h>
int main()
{
        &generatedCode
        return 0;
}


</body>
<whileLoop>
        while(&condition){
                &true
        }
        &false
</whileLoop>

<printText>
        printf("&text");
</printText>
```

Figure 5: Subset of Rules for Generating Codes in C

```
#include <stdio.h>
int main()
{

        while(true){

                printf("Entered 1st");
                        while(true){

                                printf("Entered 2nd");

                        }



        }


        return 0;
}
```

Figure 6: Generated Code

8. RESULTS AND DISCUSSIONS

8.1 Code Generation

It is important that the system is able to produce codes that are syntactically correct as one of the main objectives of the system is to be able to bridge the gap of its users from flowcharts to implementing it in using a programming language. In order to do so, test cases were used in order to test the system.

Various flowcharts with different structures were used as test cases in testing the system's code generator. Examples of which include nested loops, nested if's, declaration of variables, and so on. After which, the system was asked to the code equivalent of the flowchart in the languages that the system currently supports. Then, codes were then inputted to an IDE where the language of the code generated is supported to determine the syntax errors are present. Base on the test conducted, the system is therefore, able to produce codes that are syntactically correct.

No code quality testing was conducted because the code generated is dependent on the rules incorporated in the system. Hence, if the rules included are subpar, the code generated will also be subpar and vice versa. For this research, the system was only aimed to produce syntactically correct codes.

8.2 Survey Questions

Table 2 shows the survey questions used for the evaluation of the system. As discussed earlier, the survey questions formulated were patterned on RAPTOR's survey question. The questions for the evaluation of the system aims to determine whether the MDDT was able to help the users in gaining new knowledge of programming, potentially helped its users to ease the learning curve of syntax conventions, and determine the user friendliness of the system.

The first question in the survey determines whether the respondent was oriented properly before they were left to use the system. Questions 2 and 5 on the other hand determine the user experience in the system as they were using it and whether or not they will consider using it in the future for learning purposes. Questions 3 and 4 determine whether the user was able to gain knowledge after using the system in terms of creating and designing programs. Lastly, question 6 determines whether the system was able to help the user to ease the learning curve of jumping from designing a program using flowcharts to actually encoding it. For each question, users were asked to rate them from the scale of 1 to 5 where 5 being the highest.

A total of 30 respondents were gathered to evaluate the system. Respondents of the system were non-computer science university students. All of the evaluators of the system knows only a few programming concepts and has some knowledge in flowcharting.

In the first phase of the system evaluation, respondents were first oriented as to what the system does and how to create programs using the available tools in the system. After the orientation, respondents were asked to create simple programs such as testing whether the input was equal to some value, simple guessing game, and so on.

Table 2: Survey Questions and Results

1. I was oriented enough on how to use MDDT so that I could use it to effectively create programs	2.83
2. It was easy to create program designs using MDDT	2.20
3. MDDT was able to help me develop and improve my problem solving skills	2.30
4. MDDT helped me to better understand how computer programs operate	2.83
5. I enjoyed programming in MDDT, and I will consider using it in future.	3.10
6. Being able to see the generated code from my flowchart allowed me to understand how to program using standard codes (particularly in C and/or Java)	3.80

Table 2 shows the results of the survey conducted. The first question shows that the orientation made by the respondents to the system evaluators were not enough before letting them use the system. In addition, the results for question 1 probably yielded average results because it was simply conducted online. In question 2, most of users find it hard to create programs using the tool. The analysis for this is that they have to manually connect the nodes from one to another. It requires respondents to perform many clicks before they are able to do so. In question 3, the tool seems to not have helped the respondents to improve in the problem solving skills. This was because the respondents were not knowledgeable with programming to begin with. The respondents might have not seen that programming can be used to solve problems. In question 4, the tool seem to have helped the evaluators to better understand how to program. This is probably because they were able to experience programming first hand and even design one using visual representations. For question 5, most of the evaluators gave a high score for this. The analysis for this is that, the respondents may have enjoyed using shapes to develop and design a program. For question 6, the respondents believe that being able to relate their created flowchart to the generated code has allowed them to understand programming more. The best analysis for this is that since they have already known the logic behind their program, they no longer have to struggle in understanding the program function, but only has to struggle how to understand the syntax conventions of the programming language.

9. Future Works and Developments

This research could be extended by examining MDDT's real effects in a classroom setting such as in the exam results of the students, programming projects, performance and so on. Should the system reach a satisfactory working level, it could then be mandated into the university's basic programming classes and other institutions as well.

In addition, the system could be extended to incorporate Object-Oriented concepts such as classes, etc. The system could accommodate more functions like source code to flowchart, interpreter, and so on. Additionally, a design analyzer could also be added in order to assist the learner as well. That way, the students could learn how to programming logically without the need of an expert.

A study could be conducted in order to examine the quality of the code that the system is currently providing and how to improve it. Furthermore, since a flowchart could produce one or more codes, intelligence or heuristics could be implemented to determine which will be the best out of the possible codes that can be generated. The system could also be implemented as a module to existing programming tutoring systems that incorporates basic programming tutorials.

10. Conclusions and Recommendations

Based on the experiment conducted by various studies, using of flowcharts for introductory programming courses is beneficial for students. Hence, it is no surprise that many institutions such as those in Taiwan and India incorporate flowcharting as part of their academe.

The system was able to produce codes that are syntactically correct based on the test cases used in evaluating the system. From the flowchart input, the system converts it to syntax tree, and based on the syntax tree, the system then uses the rules of conversion along with the generated syntax tree to produce the target code equivalent of the flowchart. One of the limitations of the system is that not much validation was incorporated in the system. Hence, should the user be inputting an invalid or buggy design, the generated code will also be invalid and buggy.

In addition, rules formulated were only aimed to produce syntactically correct codes. Thus, should the rules be included are subpar, codes that will be generated by the system will also be subpar.

Given an input flowchart, it is possible to generate multiple versions of codes. However, this will not be supported by the system as it will not incorporate artificial intelligence to determine the best code to generate. This idea was adapted from the system developed by [6], Visual Logic. To be able to produce multiple codes given a non-deterministic flowchart may also be incorporated in future development of the system.

Based on the survey conducted, it was found out that allowing novice programmers to relate their formulated design to some code will help them in easing to learn how to program. More so, it was also found out that novice programmers are really visual learners as they enjoy programming using shapes or flowcharts. For future works, developers may add syntax validation for the flowchart and allow the flowchart to be interpreted similar to RAPTOR and FLINT. Moreover, add more capabilities and functions to it. It is also recommend that future developers should include compilers in the system so that user programs may be compiled and run it in real time.

11. REFERENCES

[1] Bowling Green. (2001). *Visual Logic 2.2*. Retrieved November 6, 2009, from Visual Logic: http://www.visuallogic.org/

[2] Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M. (2004). Raptor: Introducing Programming to Non-majors with Flowcharts. *Journal of Computing Sciences in Colleges* , 52 - 60.

[3] Crews, T., & Ziegler, U. (1998). The Flowchart Interpreter for Introductory Programming Courses. *28th Annual Frontiers in Education* (pp. 307 - 312). Tempe: IEEE Computer Society.

[4] Dmitriev, S., & JetBrains. (2004, November). *Language Oriented Programming: The Next Programming Paradigm.* Retrieved September 27, 2009, from OnBoard: http://www.onboard.jetbrains.com/is1/articles/04/10/lop/

[5] Guisset, J., & Mascherpa, L. (2004, September 9). *Metalmodel Engine*. Retrieved November 21, 2009, from Sourceforge: http://m-engine.sourceforge.net/doc/index.xml

[6] National Institute of Open Schooling. (n.d.). *Flowcharting*. Retrieved December 4, 2009, from NOS : Certificate in Computer Applications: http://www.nos.org/htm/basic2.htm

[7] Park, D. H., & Kim, S. D. (2001). XML Rule Based Source Code Generator for UML CASE Tool. *APSEC Proceedings of the Eighth Asia-Pacific on Software Engineering Conference* (p. 53). Washington, DC, USA: IEEE Computer Society.

[8] University of Paderborn. (2009, November 18). *About Fujaba*. Retrieved December 10, 2009, from Fujaba Tool Suite: http://www.fujaba.de/projects/education-with-life3/teaching-object-oriented-concepts-and-design-with-eclipse.html