http://dx.doi.org/10.14236/ewic/HCI2017.26

Extensible Visual Programming Model for Modular Systems Targeting Novices

Yasura Vithana
University of Moratuwa
Moratuwa, 10400, Sri Lanka
yasura.10@cse.mrt.ac.lk

Hashini Senaratne
University of Moratuwa
Moratuwa, 10400, Sri Lanka
hashini.10@cse.mrt.ac.lk

This paper presents a visual programming model with its entire flow starting from the visual program creation to the execution of the program on the target system. The introduced visual programming model is developed targeting modular hardware systems where the module specific execution of tasks is offloaded to the respective module. This model can be used on systems like robot kits and other module based programmable systems where modules have self-sufficient processing power, specially targeting STEM education. The nature of the target system and the message passing model of task execution have given this model the qualities like simplicity and extensibility. The generated executable consists of instructions that can be executed by the interpreter-like execution engine that resides in the central processing unit of the target system. This also gives the ability to directly execute instructions on the system without going through the traditional program translation process.

Extensible; Modular; SiFEB; STEM; Robot Kits; Visual Programming Language.

1. INTRODUCTION

Computing technologies have become the gateway to the future. It is pointed out that ICT like technological fields have an extraordinary potential to enhance the learning and other development processes of children, by providing them novel opportunities [1]. Therefore, improving teaching and learning in STEM (Science, Technology, Engineering and Mathematics) education has become an economic factor in almost every nation [2]. Building robots has been identified as a supported branch for STEM based activities as it is capable of involving children in problem-based learning while associating with concepts of physics, electronics, mathematics and programming [3]. It has now become a common practice especially in the developed countries to introduce children to this vast field of capabilities at a very young age. There still is a hesitance in many developing nations to get exposed to STEM related activities like computer programming and robotics until higher education. Poor conditions of laboratory facilities and instructional media, lack of research collaboration across STEM fields and lack of support from the school system are some of the root causes responsible for less exposure to STEM based education in these countries [4]. This paper presents part of the work done in developing SiFEB [5]; a low cost robot kit targeting kids which was developed in order to make robotics more appealing to the entrants who do not have significant knowledge on electronics or programming.

Computer programming has come a long way since it was born. Approaches like machine code, assembly code and high level programming languages have been introduced at various levels of complexity and capability. Visual programming emerged as a simpler version of programming opposed to the text based programming approaches. It has been shown that visual programming languages have the ability to improve novice performance in programming activities [6]. With the invention of visual programming languages like Scratch, Alice and Greenfoot and introduction of them to school children, especially a younger crowd joined the computer programming community [7, 8]. The visual programming model introduced in this paper has been developed with the focus on certain aspects that have not been addressed in the other available solutions and specially targeting modular systems like robot kits.

2. RELATED WORK

In programming modular systems like robot kits, different programming approaches like conventional textual programming, visual programming and tangible programming are being used. Arduino development board which is included with a text-based programming language has become a platform used by many educators around the world to cultivate STEM based education in students of different ages [9, 10]. T_ProRob [11] and Tern [12] are some of the tangible programming languages developed targeting robot programming. T_ProRob can be used to program NXT Lego robots while Tern can be used with controlling LEGO MindstormsTM RCX brick. Situated tangible robot programming is a recent project carried out to program a robot by placing specially designed tangible blocks in its workspace [13]. Although tangible programming is identified as an approach which is more enjoyable, on the other hand with increased familiarization with computers, visual programming languages are considered as the easiest approach to use [11]. Another significant benefit of a visual programming language is that unlike in tangible programming frameworks, the number of blocks it can provide to the user is unlimited and therefore it can be used to develop programs of any length and any complexity.

Visual programming languages simplify the programming tasks by replacing the typed syntax based languages by graphical blocks that represent various aspects of the programming domain such as variables, commands, loops and conditional blocks. These graphical blocks can be arranged and connected together in a visual programming editor. Over the past decades many types of visual programming languages have been developed presenting different types of experiences to users. Kodu is a rule based visual programming language which is integrated in a realtime 3D gaming environment with virtual robots and designed specifically for young children to learn through independent exploration [14]. UbiPlay, a technology platform for programmable interactive playgrounds, allows children to create and play games in interactive playground environments using a finite state machine based visual programming language [15]. LabVIEW is a visual controlled dataflow driven language [16] which is also can be used to program Lego Mindstorms Kits. NXT-G is the visual programming language used by the Lego Mindstorms environment [17] where there are other visual programming languages like Open Roberta [18] that have emerged to program Lego Mindstorms. Scratch [19] has become a popular block based visual programming language among children and robot kits like Lego WeDo, Lego Mindstorms NXT and Finch have integrated with Scratch in order to make programming easier for novices. Modkit for Vex robot kit is another block based visual programming language introduced for novice users [20].

The visual programming model that we introduce in this paper has been highly motivated from these existing visual programming solutions, but has been improved in certain areas to specifically suit the target class of systems: modular systems with task offloading ability in a distributed manner.

3. WORK DEVELOPED

The visual programming language in topic is a command based model and the processing of each of the command is expected to be done in a distributed manner. Since this model was developed targeting modular systems like robot kits, the main motive was to coordinate tasks done by the connected modules. These tasks can be standalone tasks which can be simply started and ignored and some other tasks in which the results are passed to decision making. For example, in the context of a robot kit, there can be a task which makes the robot turn right and also a task to measure the distance to an obstacle using a sensor. The visual programming model is developed to program the central controlling entity that can control all the other connected modules by issuing commands and analysing their results. The important point here is that due to this configuration, the output of the visual programming environment can be made extremely simple and uniform as the native executions of the tasks are offloaded to various modules which are built with all the necessary algorithms and other programming components. The visual programming model that we introduce requires the target system to have the above characteristics.

This visual programming environment contains a library of programming blocks and an editor area which is used to place and order the dragged and dropped programming blocks in order to create the intended program. These blocks can be divided into three categories. One category is holders that represent programming structures. These include single task blocks, blocks with a task and terminating condition, loops and conditional blocks. They alone do not result in any useful output but need to be utilized in the process of creating a program combining the other two types of programming blocks. These can be even nested when required. The second category is the programming blocks that represent the execution of a task. Each task that the modules are capable of doing, but do not return a result results in this type of a block. The third category is the blocks that request the modules to return a result with or without doing a specific task. These are the blocks that are used in conditions for control blocks. These blocks were sufficient for the intended task of the original work, but the same concepts can be extended to other blocks as required.

3.1 Code Generation

All the types of blocks are translated to an ordering of four basic instructions which are “execute”, “if”, “jump” and “end”. The “end” instruction is to indicate that the program has reached the end. The “execute” instruction is used to initiate an action on a module. The parameters of this instruction lets the execution engine to know the module to which the instruction is targeted and also lets the module know which action it needs to perform with which settings. The “if” instruction too has the module it is targeting and the input that is required from the module. Additionally it has the necessary parameters for the condition evaluation. If the comparison is valid the execution engine goes on executing the next instruction in the program. If the condition is not valid, the execution engine will ignore the next instruction and will move on to the instruction appearing after that. The “jump” instruction has only one parameter and the instruction forces the execution engine to move its execution to the instruction pointed by this parameter.

All these instructions are represented at the execution engine as an array of bytes which has the common format <instruction length> <instruction_id> <parameters>. The program to be executed is converted to a list of such instructions. The following section explains how the above three commands are used to generate executable instructions for different programming blocks.

Each of the four types of holders found in programming structure category has a code template associated with it. The basic task of code generator is to recursively go through the holders in the program, fill code template for each holder with the information of its content and finally merge all these code snippets together to generate the final code.

3.1.1. Single Task Holder
Single task holder can only has a self-terminating action such as “Turn Right”. In this type of holders, the following code snippet is used as the code template.

Instruction No	Instruction
i	execute(module address, action command, other_params);

3.1.2. Holder for Task with Terminating Condition
This holder can hold a constrained action such as “Go Forward” and a condition such as “Distance is n” that should be satisfied to terminate that action. In this case, the code template is as follows.

Instruction No	Instruction
i	execute(module address, action command, other_params);
i+1	if(module address, condition command, comparison_type, response_size, ref_value, other_params);
i+2	jump(i+4);
i+3	jump(i+1);
i+4	execute(module address, stop action command, other_params);

3.1.3. Repeat (Loop) Holder
Repeat Holder contains a condition block and a set of holders that will be executed repeatedly until the condition becomes true. The following code template is used in such situation.

Instruction No	Instruction
i	if(module address, condition, compare_type, response_size, ref_value, other_params);
i+1	jump(n+1);
i+2 to n-1	//recursively go through the child holders
n	jump(i);
n+1	//next command

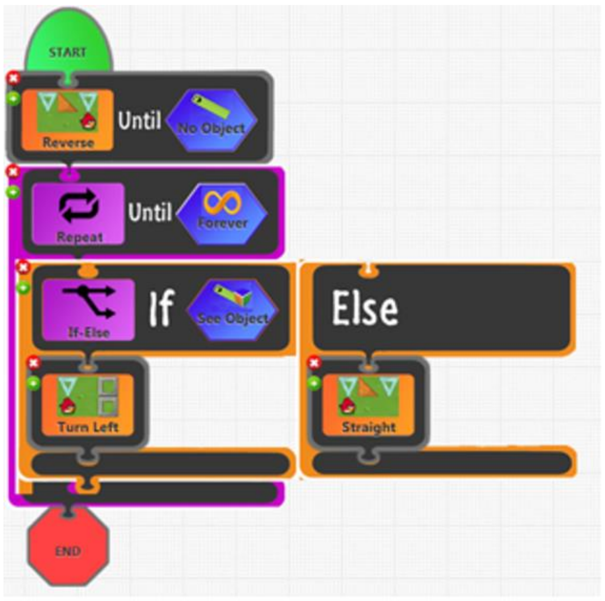
3.1.4. If-Else (Conditional) Holder
If-Else holder can have a condition and two sets of child holders in which the first set will be executed if the condition is true and the second set will be executed otherwise. The following code template is used for the code generation for an If-Else holder.

Instruction No	Instruction
i	if(condition.device.address, condition.command, condition.comparison_type, condition.response_size, condition.ref_value, other_params);
i+1	jump(m+2);
i+2 to m	//recursively go through the set of child holders under “else”
m+1	jump(n+1);
m+2 to n	//recursively go through the set of child holders under “if”
n+1	//next command

Figure 1 shows a sample visual program and the result after converting it to the list of instructions.

3.2 Execution Engine

The execution engine lies in the central controller of the system. This is a software component which is capable of executing the instructions generated by the visual programming environment. The execution engine acts more like an interpreter. This starts from the first instruction of the program and then controls the flow of execution accordingly. When an "execute" instruction is reached, the engine decodes the instruction and passes the necessary parameters to the respective module mentioned in the instruction itself. Then the engine goes on to the next instruction. Similarly when an "if" instruction is reached, the engine passes the necessary parameters to the respective module and evaluates the response according to the parameters of the instruction. If the condition is not valid, the execution engine skips the next instruction. When a "jump" instruction is reached, the execution engine simply goes on to execute the instruction pointed by the parameter of the "jump" instruction. When the "end" instruction is reached, the execution engine stops executing instructions further.



```
 1 execute(a1, com3, other_params);//forward
 2 if(a2, com1, type, size,ref, other_params);
 3 jump(5);
 4 jump(2);
 5 execute(a1, com4, other_params);//terminate
 6 if(a3, com1, type, size,ref, other_params);
 7 jump(14);
 8 if(a2, com1, type, size,ref, other_params);
 9 jump(12);
10 execute(a1, com1, other_params);//straight
11 jump(13);
12 execute(a1, com2, other_params);//turn left
13 jump(6);
14 end
```

Figure 1: *Sample Visual Program and the Corresponding Instruction Translation*

3.3 Other Special Features

Since the task specific knowledge is distributed among the various modules of the system, the programming environment sees all the tasks or commands in a similar format. This omits the requirement of revising the programming environment. In the context of a robot kit, when a module of a new model needs to be used in programming, the user can simply enter the necessary parameters for each capability of the module and the environment may generate a set of new blocks for the new model. Similarly it needs to be mentioned that the existing capabilities too can be customized to cater one's needs by tuning the parameters of each instruction generated representing capability blocks.

The program is not compiled to be run directly on hardware. Instead, there is an interpreter-like execution engine to execute each instruction. Due to this abstraction, this model can even be used to execute instructions at will. This doesn't make sense for the "if", "jump" and "end" instructions, but it may be useful to have this facility for the execution of the "execute" instruction. In the context of robot kits, we can use this to test certain capabilities. Project SiFEB [5] incorporates this feature to help the kids understand what each demonstrable capability of a module is about.

Another important thing is that the programming environment consists of a very lightweight and simple compiler alternative that transforms the visually developed program into a program executable on the target hardware. This allows the application to be less resource consuming thus fitting in more setups.

4. CONCLUSION

The visual programming model introduced in this paper is a programming model especially targeting the modular systems where processing is offloaded among various modules connected to a central processing unit with a master-slave communication capability with the central processing unit as the master. This model is not much adaptive to a single point general purpose processing but has its benefits when implemented targeting a modular systems like robot kits, but the applications of this model are not limited to robot kits. If a system can be looked into as a modular system where each module is self-equipped to accept and execute commands, then this model can be adapted to that system. This model results in a very lightweight compiler alternative which can be extended easily without modifying any of the core components. Project SiFEB [5] turned out to be a successful project which had a great contribution from this visual programming model.

5. REFERENCES

[1] Kalas I. (2013) Integration of ICT in Early Childhood Education. The World Conference on Computers in Education, Poland, 2-5 July 2013, pp. 217–225.

[2] Kennedy T. J. and Odell R. L. (2014) Engaging Students in STEM Education. Science Education International, Vol. 25, No. 3, pp. 246-258.

[3] Vanderborght B., Ciocci C., Vandevelde C. and Saldien J. (2013) Overview of Technologies for Building Robots in the Classroom. The International Conf. on Robotics in Education, Lodz, Poland, 2013, pp.122-130.

[4] Ejiwale J. A. (2013) Barriers to Successful Implementation of STEM Education. Journal of Education and Learning, Vol. 7, No. 2.

[5] Senaratne H., Gunatilaka P., Gunaratna U., Vithana Y., de Silva C. and Fernando P. (2014) SiFEB - A Simple, Interactive and Extensible Robot Playmate for Kids. 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology, Malaysia, 3-5 Dec. 2014, pp. 143-148. IEEE.

[6] Price T. W. and Barnes T. (2015) Comparing Textual and Block Interfaces in a Novice Programming Environment. The eleventh annual International Conference on International Computing Education Research, Nebraska, USA, 9-13 July 2015, pp.91-99. ACM, New York, USA.

[7] Utting I., Cooper S., Kolling M., Maloney J. and Resnick M. (2010) Alice, Greenfoot, and Scratch – A Discussion. ACM Transactions on Computing Education, Vol. 10, No. 4.

[8]Maloney J., Resnick M., Rusk N., Silverman B. and Eastmond E. (2010) The Scratch Programming Language and Environment. ACM Transactions on Computing Education, Vol. 10, No. 4.

[9] Mellodge P. and Russell I. (2013) Using the Arduino Platform to Enhance Student Learning Experiences. The 18th ACM Conference on Innovation and Technology in Computer Science Education, England, UK, 1-3 July 2013, pp.338-338. ACM, New York, USA.

[10] Rubio M. A., Hierro C. M. and Pablo A. P. (2013) Using Arduino to Enhance Computer Programming Courses in Science and Engineering. The EDULEARN13 Conference, Barcelona, Spain, 1-3 July 2013, pp 1527-1533.

[11] Sapounidis T. and Demetriadis S. (2013) Tangible versus graphical user interfaces for robot programming: exploring cross-age children's preferences. Personal and Ubiquitous Computing, vol. 17, no. 8, pp. 1775-1786.

[12] Horn M. S. and Jacob R. (2007) Tangible programming in the classroom with tern. CHI '07 Extended Abstracts on Human Factors in Computing Systems, San Jose, California, USA, 27 April – 3 May 2007, pp. 1965-1970.

[13] Sefidgar Y. S., Agarwal P. and Cakmak M. (2017) Situated Tangible Robot Programming. The 2017 ACM/IEEE International Conference on Human-Robot Interaction, Vienna, Austria, 6-9 March 2017, pp. 473-482. ACM, New York, USA.

[14] MacLaurin M. B. (2011) The design of kodu: a tiny visual programming language for children on the Xbox 360. 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Austin, 26-28 January 2011, pp.241-246. ACM, New York, USA.

[15] Mattila J. and Väätänen A. (2006) UbiPlay: an interactive playground and visual programming tools for children. Conference on Interaction design and children, Tampere, Finland, 7-9 June 2006, pp.129-136.

[16] Cox P. T. and Gauvin S. (2011) Controlled Dataflow Visual Programming Languages. The 2011 Visual Information Communication - International Symposium, Hong Kong, China, 4-5 August 2011. ACM, New York, USA.

[17] Griffin T. (2010) The Art of Lego Mindstorms NXT-G Programming, 1st ed. San Francisco.

[18] Jost B., Ketterl M. and Budde R. and Leimbach T. (2014) Graphical Programming Environments for Educational Robots: Open Roberta - Yet another One?. IEEE International Symposium on Multimedia, 10-12 December 2014, pp. 381-386. IEEE.

[19] Resnick M., Maloney J., Monroyhernández A., Rusk N., Eastmond E., Brennan K., Millner A., Rosenbaum E., Silver J., Silverman B. and Kafai Y. (2009) Scratch: programming for all. Communications of the ACM, Vol. 52, No. 11 pp. 60-67. ACM, New York, USA.

[20] Millner A. and Baafi E. (2011) Modkit: Blending and Extending Approachable Platforms for Creating Computer Programs and Interactive Objects. The 10th International Conf. on Interaction Design and Children, Ann Arbor, USA, 20-23 June 2011, pp. 2–5. ACM, New York, USA.