Designing for Children
- With focus on ‘Play + Learn’

Visual Programming Application for Children to program Robotic Toys

Nikhil Karwall, Design Consultant, Mumbai, India, nikhil_karwall@yahoo.co.in

Abstract: Robotics toys offer a unique play-based medium for kids to understand different scientific concepts and use their creativity to come with intelligent and fun solutions. Certain Programmable Robotic Kits allow children to give a desired physical structure to the robot and further program it to carry out specific intelligent tasks, fuelling their imagination on multiple levels. The paper discusses the thought process and rationale behind the design of a software-based medium for children to program assembled robots created out of such kits.
The application was targeted to be intuitive and simple for children to program their robots for different kinds of tasks, and again be scalable enough to take care of complicated programming scenarios, while adhering to the technical code-generating backend at all times. Taking special consideration over the dependency on the physical structure of the robot, a unique application was developed that allowed the child to create intuitive and lucid ‘visual algorithms’ with an interface that mandated no prior knowledge of programming.
Unlike design of conventional software applications meant for certain fixed/pre-decided goals, the open-ended tasks of the software necessitated a different kind of ideation process. Testing of the application with children from various programming backgrounds and working alongside the development team while considering the constraints of the used technology also formed as key stages of the design process.

Key words: Visual Programming, Programmable Robotic Kits, Robotic Toys, Programming, Interface, Algorithms.

1. Introduction

Programmable Robotic Kits are robotic toys that possess certain special capabilities such as mobility, motion transfer, hearing (sensing sound), vision (sensing obstacles), computational abilities etc., owing to components such as sensors, processing unit/s, motors and buzzers. The child can further program these kits so as to utilize these capabilities in different ways under different conditions e.g. a child could program a programmable robotic kit in the form of a Car to 'stop' or 'take a left turn' every time there is an obstruction in front of it. Thus, such robotic toys help the child understand and absorb the fundamentals of science and electronics while at play and aid in developing his confidence and general interest towards an intellectual stream such as robotics at a very early stage.

Robots prepared using programmable robotic kits can be instructed to carry out definite actions under different conditions by feeding relevant programs into them. The child needs to write a program on his computer and feed it into the robot. Generally, conventional programming languages such as C are used for the purpose. Being complex and abstract, the learning curve associated with such conventional languages is quite steep, especially for young kids. A Visual Programming Application or Language (VPL) offers a simpler medium of programming. It provides a graphical interface to the user, allowing spatial and visual manipulation of visual elements for creation of programs, instead of writing them textually (Visual Programming Language, Online). In other words, VPLs open up the world of robotics and programming to young children, allowing them to explore the medium and develop interest at an early age.

This paper shall discuss the design process for the visual programming language named *Cimple,* conceptualized for the programmable robotic kit *iPitara*.

2. Programmable Robotic Kits

2.1 General Characteristics

Programmable robotic kits consist of components of two kind, electronic components like motors, processing unit/s, sensors, and buzzers etc., and mechanical components such as wheels, gears, pulleys, connecting linkages and other physical elements. With the processing unit acting as the central intelligence console, certain electronic components act as 'input elements' i.e. provide feedback to the processing unit based on the ambient conditions e.g. sensors, while certain components act as 'output elements' i.e. their actuation is controlled by the processing unit e.g. motors. Thus, by feeding user-defined programs, the capabilities of the processing unit can be used to carry out tasks 'intelligently'. In other words, under different conditions i.e. based on the feedback of input components, the output components can be prompted to carry out pre-determined tasks, such as lifting an object on detection, changing direction of motion on detection of an obstacle etc.

Certain programmable robotic kits have a pre-defined structure, wherein the position and orientation of different components is fixed; while certain kits allow the child to create a robot with a particular shape/structure of choice using certain building-block elements.

Shown in Figure 1 is the programmable robotic kit *Mavin* (ER-6 Programmable Robot Kit, Online), having a fixed structure.

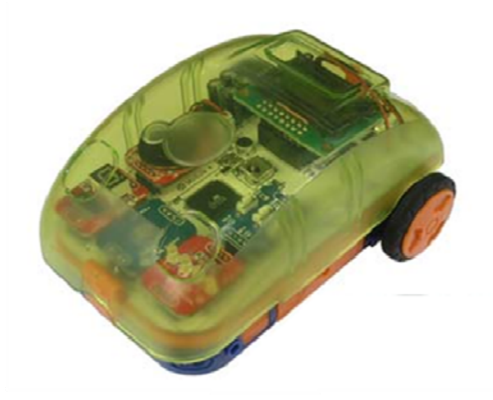


Figure 1. Programmable robotic kit having a fixed structure, *Mavin*.

2.2 Programmable Robotic Kit *iPitara*

iPitara is a modular programmable robotic kit offered by *Thinklabs*. Its key electronic components consist of a processing unit, called 'the brick', inclusive of an LCD display screen and a buzzer; sensors such as touch, obstacle, sound and ultra-sonic sensors; and DC and Servo motors. Key mechanical components consist of wheels, gears and pulleys of different sizes, and a castor. The kit provides *Meccano* (Meccano, Online) elements as building blocks, for creating the structure of the robot. The various electronic and mechanical elements are mounted over these meccano elements. Using specific cables, the different electronic components are connected to the ports available on the brick so as to complete the electronic circuit.

In accordance with the task at hand, the child creates a robot with a suitable structure using the mechanical and electronic components of the iPitara kit. The robot is then fed a program so as to make it act in accordance with the specific task. Shown in figure 2 is the iPitara robotic kit in disassembled state.

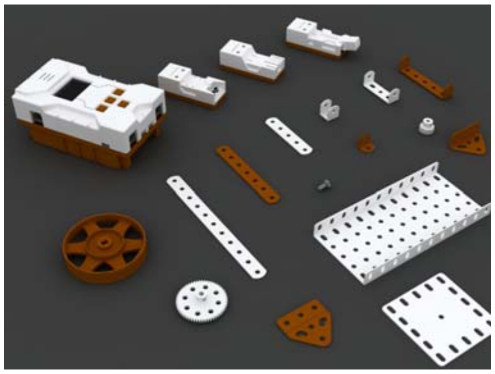
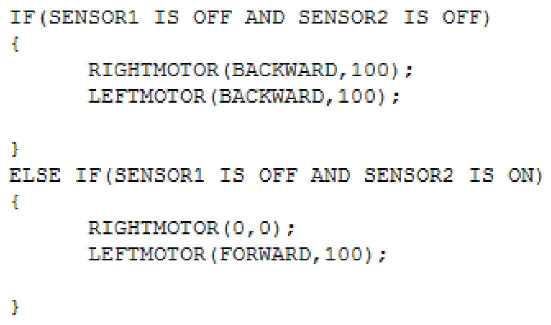


Figure 2. Programmable robotic kit *iPitara*.

2. Programming for Robotic Toys

Programming for robotics generally consists of defining the input-output relationships for various components under different conditions and supplying the necessary information for the tasks to be carried out successfully (Robot Software). Primarily, it consists of preparing an algorithm, in a particular language, with reference to a specific task at hand. For certain complex scenarios, at times it might also necessitate the creation of variables such as integers, arrays and counters, and execution of certain calculations with these variables in reference as part of the program. Shown in figure 3 is a snippet of a robot program written in *Nibble*, a programming language that derives its features primarily out of C language. The code primarily instructs two motors of a robot to regulate their motion based on the conditions sensed by its two sensors.



```
IF(SENSOR1 IS OFF AND SENSOR2 IS OFF)
{
        RIGHTMOTOR(BACKWARD,100);
        LEFTMOTOR(BACKWARD,100);

}
ELSE IF(SENSOR1 IS OFF AND SENSOR2 IS ON)
{
        RIGHTMOTOR(0,0);
        LEFTMOTOR(FORWARD,100);

}
```

Figure 3. Code written in *Nibble* language.

3. Visual Programming Language

A Visual Programming Language is an intuitive and simple means for users to create programs. Rather than typing programs textually under the constraints of certain syntax format, as is necessary in case of conventional programming languages; the visual programming language allows the user to recreate the inherent logic structure using graphical elements. Being designed for easy usage and manipulation, these graphical elements make the task of writing programs quite simple and straightforward. The ‘visual algorithms’ created using visual programming languages are intuitive to comprehend and create, and have no specific syntax requirements such as those associated with conventional programming languages, making them far easier for younger kids to pick up. Shown in Figure 4 is the interface for the *Microsoft Visual Programming Language* (VPL Introduction, Online).

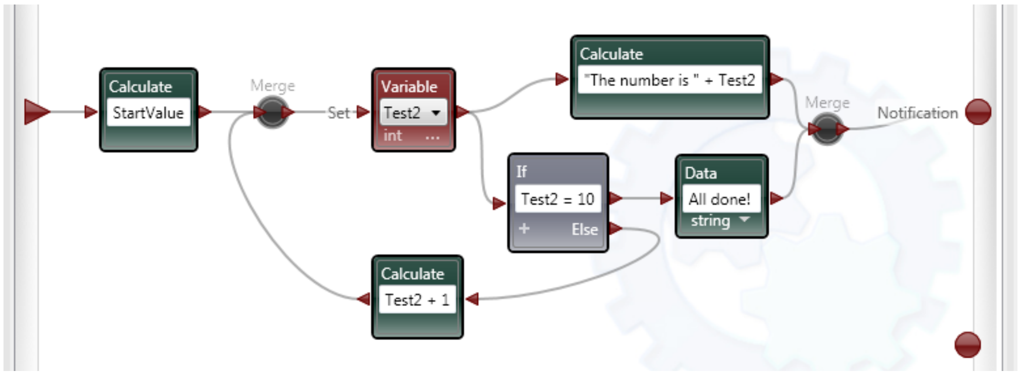


Figure 4. A program written in *Microsoft Visual Programming Language*.

4. Initial Study

4.1 Understanding Conventional Programming

The basic need of creating a Visual Programming Language arises out of the issues associated with conventional programming languages used for creating programs for robots. Being quite abstract, with specific syntax requirements and technical referencing analogies, mastering languages such as C is a complex and time-consuming task, especially for the young minds of children. With a need to come up with an alternate solution for children to program, a very crucial part of the project consisted of initially grasping the underlying logic and complexity of programs created using such conventional programming languages, so as to ensure that the designed VPL would do justice to the basic concept of algorithms and takes care of the associated complex scenarios even while being used by kids.

4.2 User Study

The initial study phase also included observing children at workshops while attempting to create logical algorithms in response to given problems. This was primarily to observe and understand the natural methodology adopted by them while trying to come up with solutions.

4.3 Competitor Visual Programming Languages

The different Visual Programming Languages being offered in the market were studied so as to understand their perspective towards the particular product and also to realize the improvements which could be offered as part of the new VPL with respect to the existing ones. Shown in figure 5 are the snippets of interfaces belonging to different VPLs.

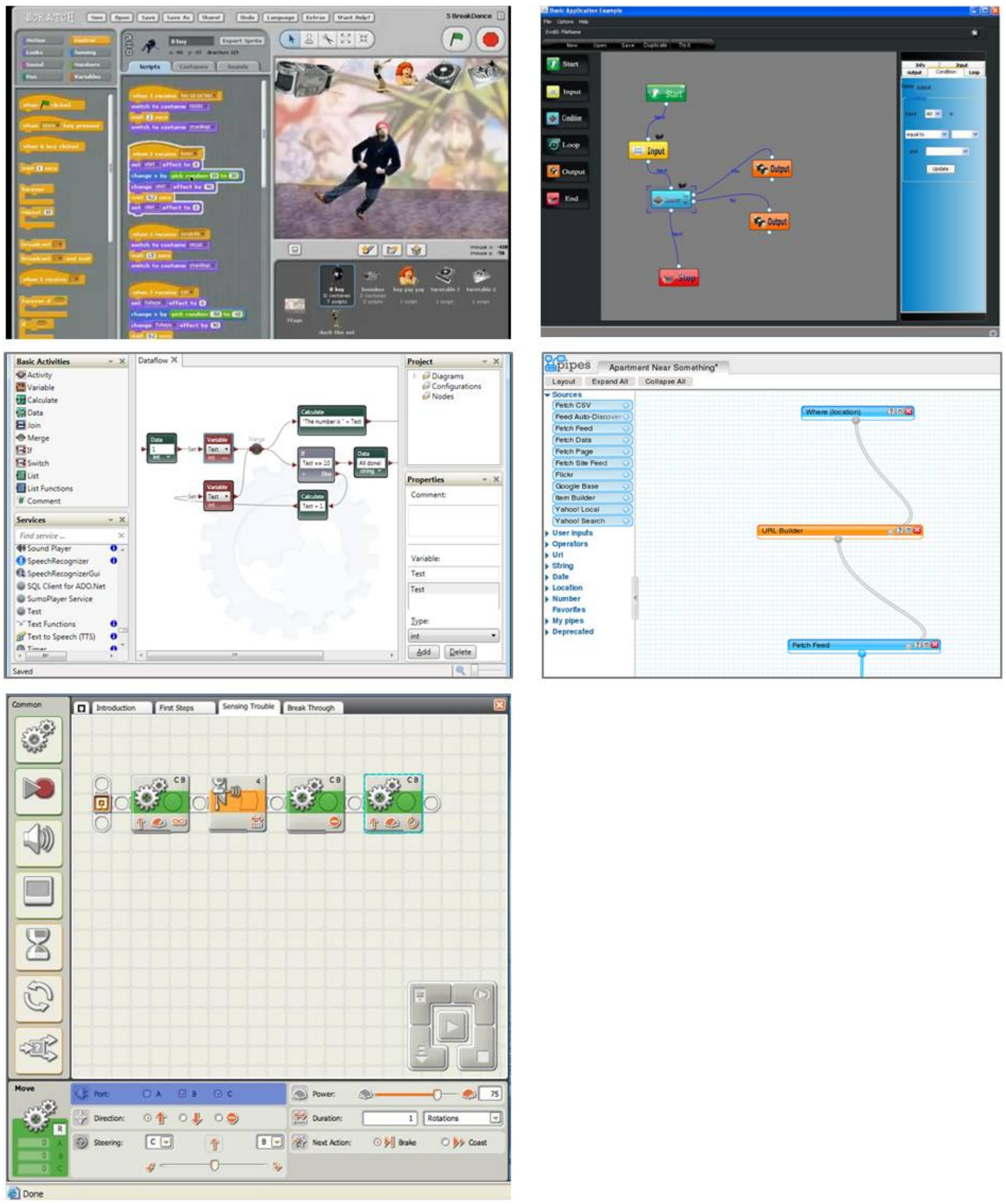


Figure 5. (Clockwise, from Top Left) Interfaces for few VPLs: *Scratch*, *Evobi*, *Yahoo Pipes*, *Lego Mindstorms NXT* and *Microsoft Visual Programming Language*.

Most of these applications, especially *Evobi* (Bi-box, Online), *Yahoo Pipes* (Yahoo Pipes, Online) and *Microsoft Visual Programming Language* (VPL Introduction, Online), allow for a format of programming that is closer to the way algorithmic flowcharts are conceived and written. Few of them being languages developed for certain programmable robotic kits having a fixed structure, the advantage of having a standard structure has been made use of by utilizing references to it while programming. Some, like *Scratch* (Scratch, Online), use visual elements that resemble physical objects which improve their intuitive usage. The software for *Lego Mindstroms NXT* (NXT Software, Online), a product that is closest to *iPitara* in terms of fundamental concept and usage, has an extremely refined interface in terms of usability and visual design, though is still abstract enough to warrant certain amount of learning time and effort.

5. Final Brief

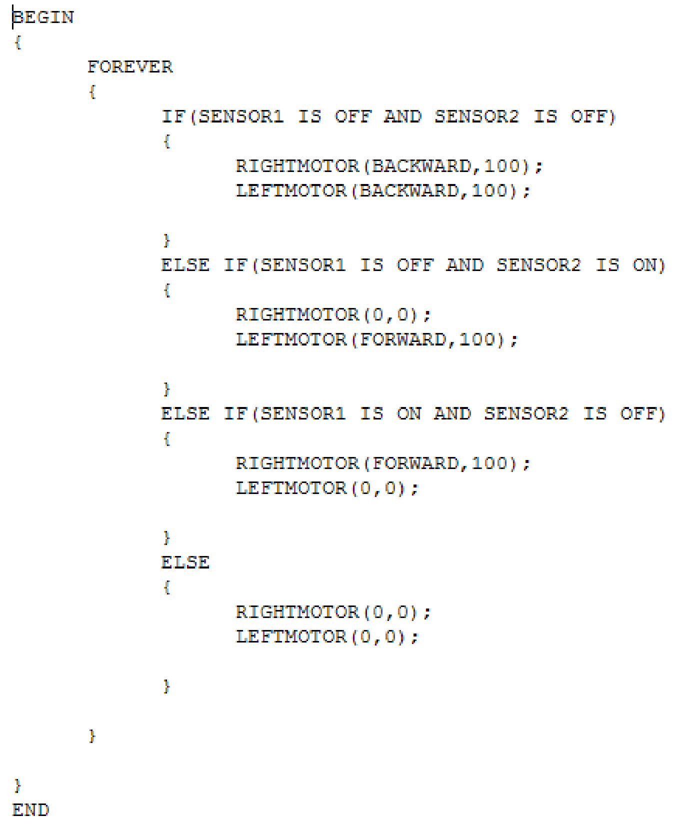
A broad brief for the design of Visual Programming Language was prepared after realizing the key commercial and user-specific needs of the user group primarily gained from the initial study phase. Mentioned below are the key points of the brief:

a. The language needs to be specifically developed for young kids, from 4th to 8th std. in school, who currently either find it very tough or are practically unable to carry out programming using conventional languages.
b. The primary aim is to enable easy and intuitive creation of programs to instruct the assembled robots, without any prior knowledge of any programming platform.
c. The VPL need not provide all the functionalities offered by conventional programming applications, focusing on creating 'simpler' programs only, not beyond a certain level of complexity.
d. The language needs to work in a manner so as to enable a machine-code to be generated at the back-end.
e. The VPL needs be designed in a manner so as to empathize with the fact that the kids eventually need to move to a conventional programming platform after a certain point of time. Thus, it needs to provide a means for the kids to be in touch and gradually develop comfort with a conventional programming language, in this case being *Nibble*.
f. The development platform to be used for creating the application is QT.

6. Ideation

A number of concepts were prepared for the VPL with reference to thought processes in different directions, absorbing the underlying complexities and relationships in the process. The concepts are explained in brief further below.

Though prior to the revelation of concepts, a sample program written in *Nibble* has been shown in Figure 6 as a reference to the format of programming for the *iPitara* kit. *Nibble* was used as the reference conventional programming medium while conceptualizing the VPL. The program shown below is for an 'obstacle avoiding' robot (prepared using the *iPitara* kit) having two touch sensors (for detection of obstacles) and two motors (for providing motion through wheels). The structure of the robot has been shown alongside as well.



```
BEGIN
{
        FOREVER
        {
                IF(SENSOR1 IS OFF AND SENSOR2 IS OFF)
                {
                        RIGHTMOTOR(BACKWARD,100);
                        LEFTMOTOR(BACKWARD,100);

                }
                ELSE IF(SENSOR1 IS OFF AND SENSOR2 IS ON)
                {
                        RIGHTMOTOR(0,0);
                        LEFTMOTOR(FORWARD,100);

                }
                ELSE IF(SENSOR1 IS ON AND SENSOR2 IS OFF)
                {
                        RIGHTMOTOR(FORWARD,100);
                        LEFTMOTOR(0,0);

                }
                ELSE
                {
                        RIGHTMOTOR(0,0);
                        LEFTMOTOR(0,0);

                }

        }

}
END
```

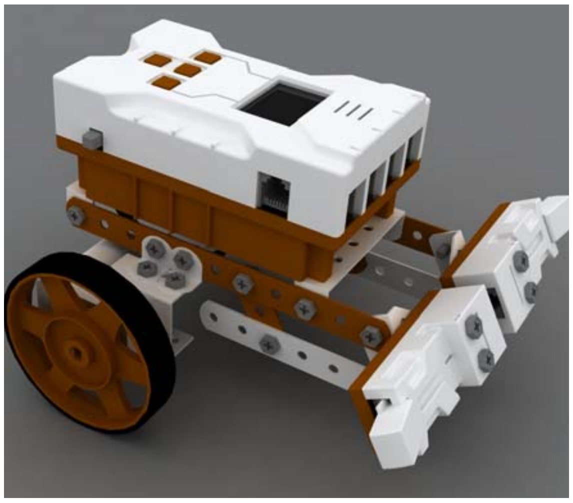


Figure 6. ‘Obstacle Avoider’ program in *Nibble* and image of the associated robot created using the kit.

The programming conventions such as ‘Begin’ and ‘Forever’ as shown in the program above also posed a challenge towards them to be included in the visual programming method since they formed an integral part of the programming requirements.
The ideation concepts have been explained below:

Concept1:
The concept consisted of a number of ‘Condition & Action’ blocks spread across the page. Different components could be inserted into these blocks to define specific conditions and the respective actions. Data associated with the components appeared alongside. Blocks placed at the same level horizontally across the page represented actions being executed in parallel. Shown in figure 7 is the conceptual wireframe interface for the idea.

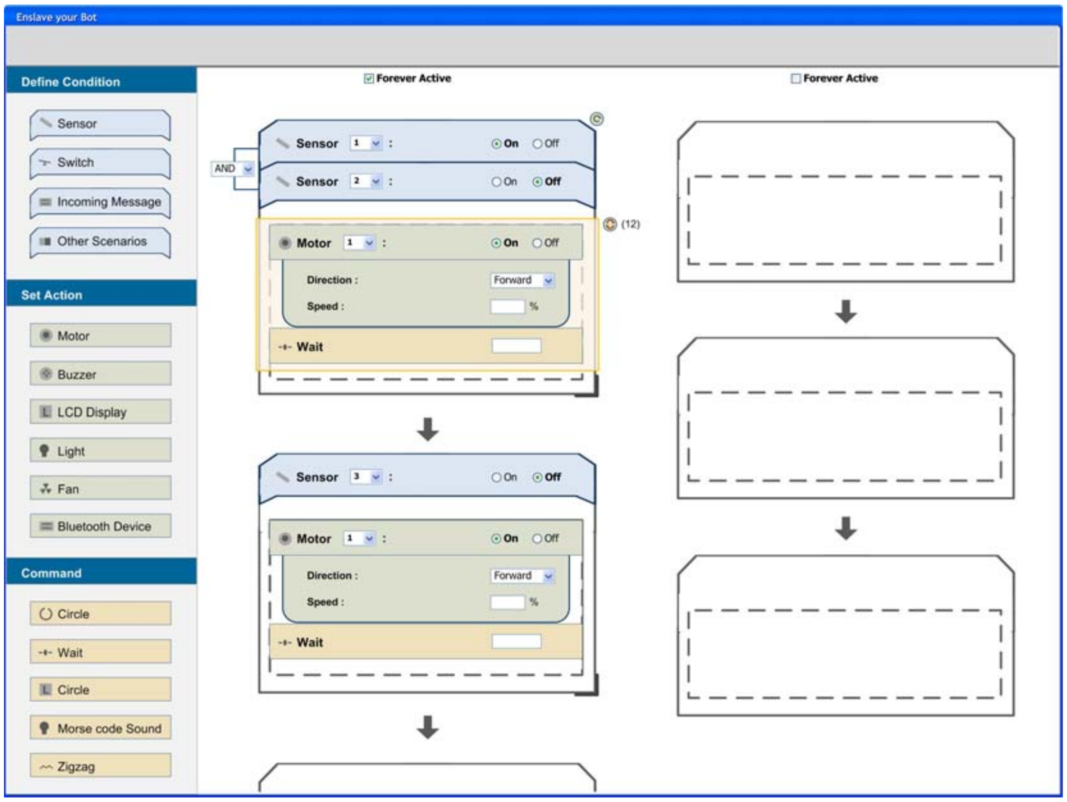


Figure 7. Concept 1.

This concept was rejected primarily for necessarily placing the conditions/actions of objects onto a rigid timeline with respect to each other, which would not always be relevant. It involved a lot of horizontal scrolling as well.

Concept 2:

It consisted of an array of ‘Condition’, ‘Action’ and ‘Otherwise’ titled blocks representing the different conditions and the respective actions when the particular conditions were true and when false. The concept was quite graphical in nature, in terms of defining conditions and providing actions to components. Shown in figure 8 is a wireframe for the particular concept.

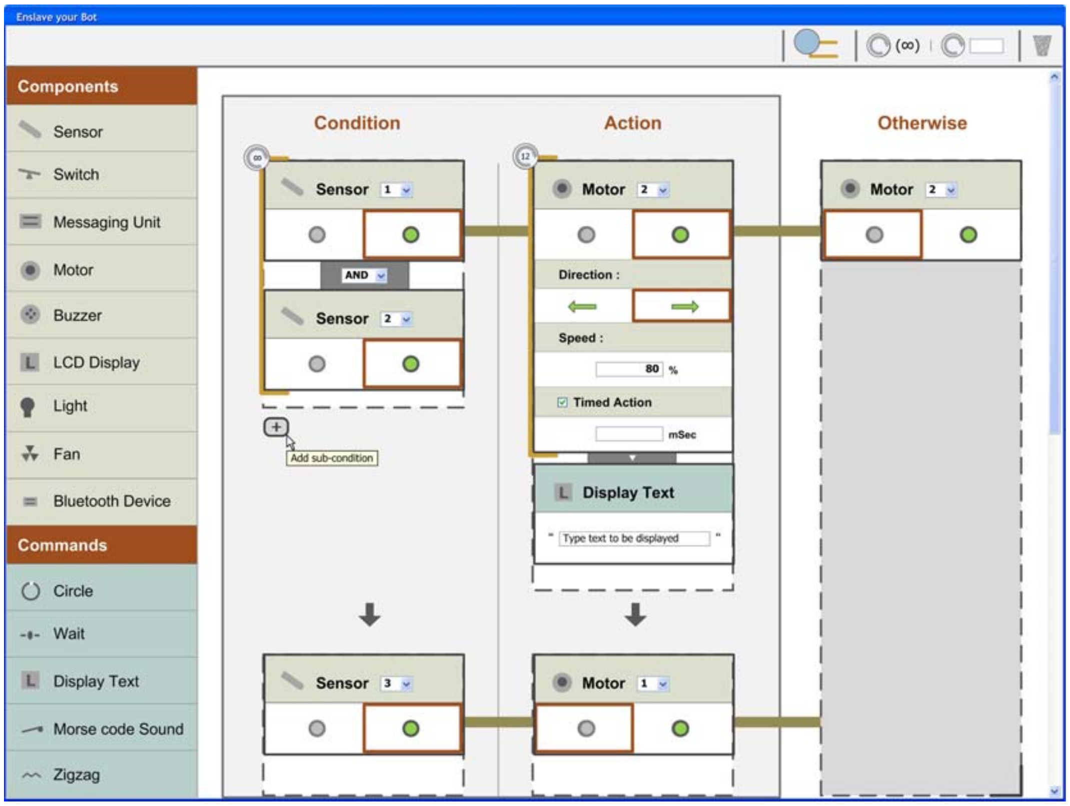
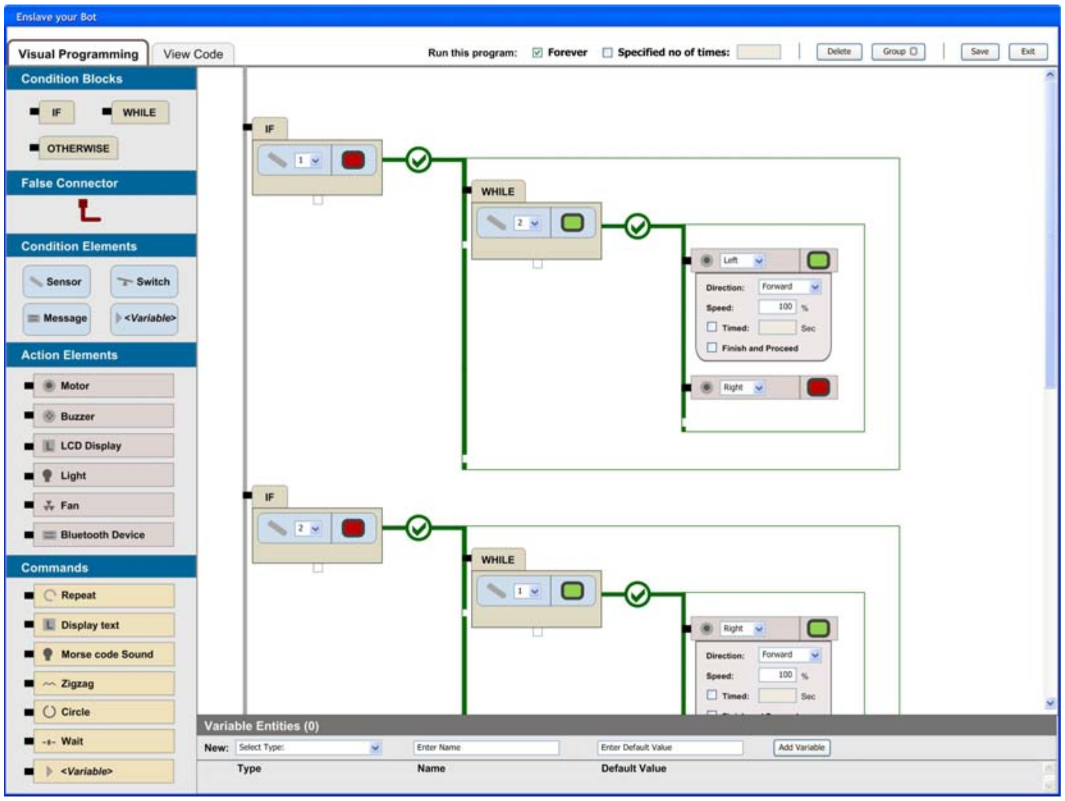


Figure 8. Concept 2.

This concept was particularly rejected because of necessarily asking for an action under every situation, even when a condition was false. This necessarily did not go along the algorithmic way of programming, and required extra thought on part of the person creating the program.

Concept 3:
This concept followed a very similar structure in terms of hierarchy and execution as with *Nibble*, though the process of including conditions and actions was quite ‘visual’ in nature. It was optional for users to provide actions in case of conditions being false. Step-based placement of elements brought out the relationships and hierarchy. Shown in figure 9 is a wireframe for the concept.



Run this program: [✓ Forever] Specified no of times:

Direction: Forward

Speed: 100 %

Timed: Sec

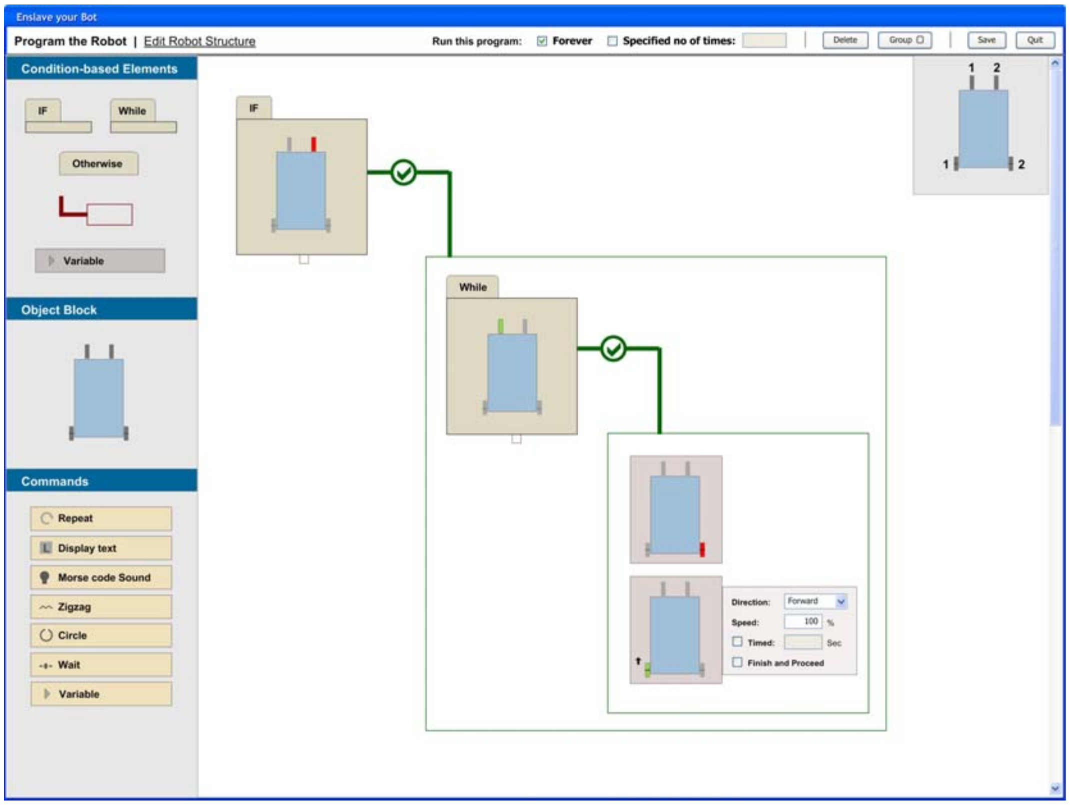
New: Select Type: Enter Name Enter Default Value

Figure 9. Concept 3.

The concept was rejected because of being too close to the conventional programming platform and not adding any critical value over and above.

Concept 4:

This concept was conceived out of the key observation that there was very little direct association between the physical form of the robotic toy with the process of programming. The structure of the robot (sensor positions, motor orientations etc.) as well as the program to be fed into the robot is created in accordance with the particular task at hand; but while using conventional methods of programming, there is no reference to either while doing the other. Trying to associate the two, this particular concept allows the user to replicate the physical state (top view) of the assembled robot and further use it to provide the conditions and respective actions in an intuitive fashion while programming. Shown in figure 10 are the sample initial wireframe for the concept.



Run this program: [✓ Forever] Specified no of times:

Direction: Forward
Speed: 100 %
Timed: Sec
Finish and Proceed

Figure 10. Concept 4.

Being able to remove the abstract nature of programming quite remarkably and bringing it so close to the physical aspects of the robotic toy, this particular concept was taken forward for evolution and refinement. Few positive characteristics of other concepts were also decided to be merged into the final iteration.

7. Concept Evolution and Testing

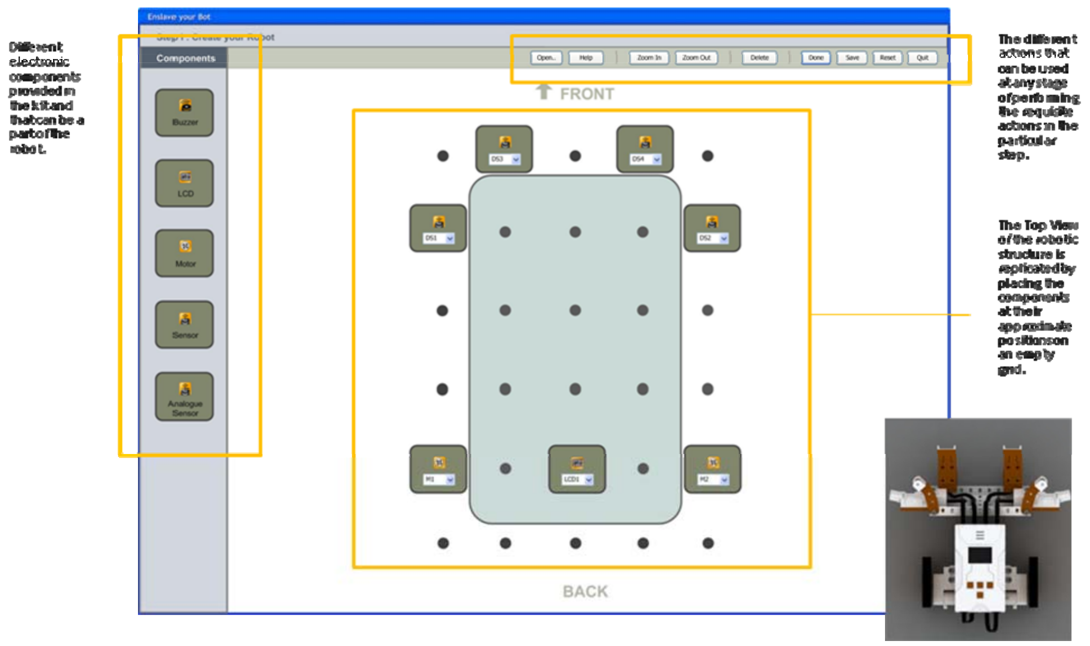
The particular concept was evolved and further evolved into detailed wireframes so as to define the concept more concretely. It was discussed with programming experts and a number of sample programs were created using the particular medium as so ensure its validity under different scenarios.

A basic working prototype of the VPL was created in the QT framework, and further tested with children (Figure 11) so as to ensure it satisfied its primary motives. It also helped grasp the initial acceptability of the application, directions for further refinement and new features to be included. The prototype was tested with children of varying ages and with varying levels of programming expertise, right from first-time programmers to proficient ones, so as to ensure it was readily picked up by novices as well as easily adapted to by experts. This study led to the provision of a number of new features into the VPL and emphasized on certain elements like efficient real-estate utilization and proper alignment and layout.

Figure 11. Initial version of *Cimple* being tested by children.

8. Final Concept

The final concept, *Cimple*, evolved with the refinements driven by user feedback has been shown in figure 12 and 13. There are two key steps towards creating programs using it: recreating the structure of the assembled robot and writing the program using the visual elements, both being fairly simple and straightforward to comprehend and carry out. The application allows the child to drag and drop elements on the interfaces and provides manipulation abilities using intuitive interaction patterns such as clicking directly over the graphical entities. Providing the means to define commands with the physical structure and orientation of the assembled robot, it makes the entire process of programming very intuitive. The design being focused towards young kids, it allows them to create programs up to a certain degree of complexity only, the focus being on able to create simple programs with great amount of ease. The basic features have been indicated alongside the interfaces shown next.

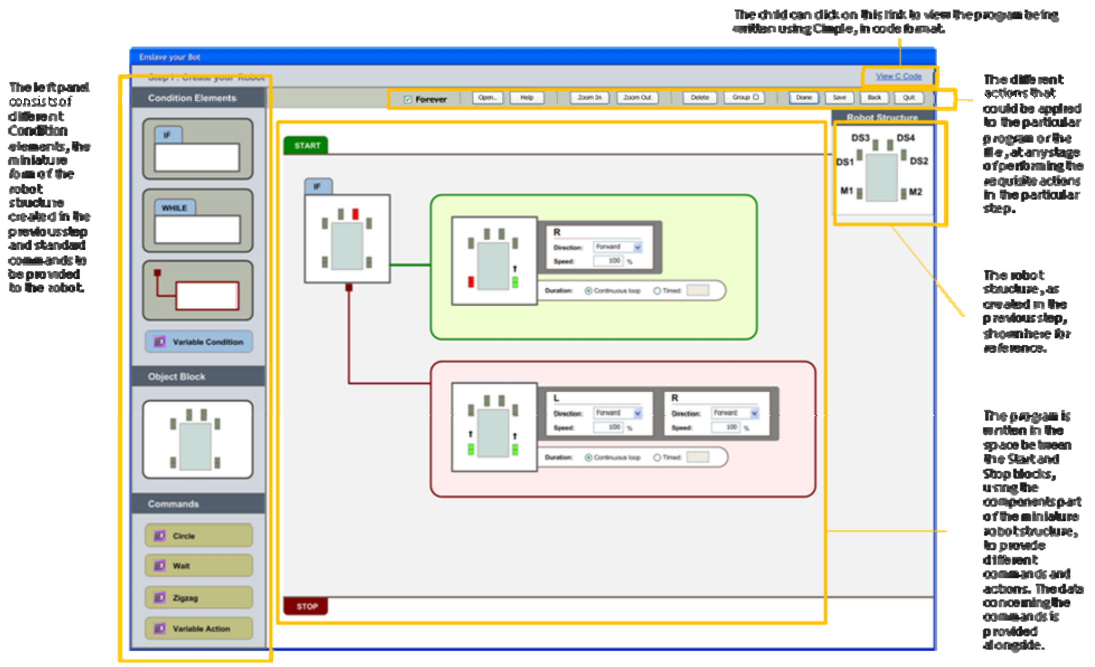


Different electronic components provided in the kit and that can be a part of the robot.

The different actions that can be used at any stage of performing the requisite actions in the particular step.

The Top View of the robotic structure is replicated by placing the components at their approximate positions on an empty grid.

Figure 12. *Cimple* interface I: Re-creating the Robot, and inset, the top view of the Robot.



The child can click on this link to view the program being written using Cimple, in code format.

The left panel consists of different Condition elements, the miniature form of the robot structure created in the previous step and standard commands to be provided to the robot.

The different actions that could be applied to the particular program or the file, at any stage of performing the requisite actions in the particular step.

The robot structure, as created in the previous step, shown here for reference.

The program is written in the space between the Start and Stop blocks, using the components part of the miniature robot structure, to provide different commands and actions. The data concerning the commands is provided alongside.

Figure 13. *Cimple* interface 2: Creating the Program.

It is important to mention here that the concept has been conceived and developed under tight deadlines and hence did not allow much work on visual design other then basic layout, alignment and presentation considerations though it holds a lot of potential for improvement in the particular aspects.

9. Validation

The *Cimple* VPL, after getting developed in the QT platform, was further handed out to kids as part of the robotic kit Workshops conducted by *Thinklabs* at various cities. It led to a lot of positive and interesting revelations, the key ones being:

a. The amount of time taken by children to learn and complete a definite number of tasks reduced from 3 days, while using the conventional programming language *Nibble*, to 1 day, with *Cimple*.
b. The kids were naturally more experimental with the software, trying out more explorations on their own.
c. It led to lesser workload on part of the instructors at the workshops, as they had lesser explanation and teaching to take care of.

Such observations and feedback communicated a successful launch of the product with the verification of the key goals it set out to achieve at the first place. Since then the product has gained fair amount of popularity amongst the target user base of younger kids as well as amongst kids/students of other age groups too.

10. Conclusion

The design of *Cimple* reflects the importance of looking at software products, especially one associated with a product such as a robotic kit which has a physical as well as software interface for the users to interact with, should be designed by paying consideration towards elements of both the mediums. Looking at the software and hardware individually, without consideration towards the other medium would not lead to products beyond a particular level of user-friendliness. The design also somewhere reflects on the fact that software products need to be looked at far more creatively, beyond the standard realm of usability-related services, to improve their utility and intuitiveness.
A clear definition of users and having them at hand to carry out the necessary kind of testing helped in creation of a product with a limited but extremely user-friendly set of features with a great degree of confidence.

References

Visual Programming Language. [Online] Available at <http://en.wikipedia.org/wiki/Visual_programming_language> [Accessed 3 November 2009].

ER-6 Programmable Robot Kit. [Online] Available at <http://www.electronickits.com/robot/ER-6.htm> [Accessed 19 November 2009].

Meccano. [Online] Available at <http://en.wikipedia.org/wiki/Meccano> [Accessed 28 November 2009].

Robot Software. [Online] Available at <http://en.wikipedia.org/wiki/Robot_software> [Accessed 7 November 2009].

VPL Introduction. [Online] Available at <msdn.microsoft.com/en-us/library/bb483088.aspx> [Accessed 22 November 2009].

Bi-box. [Online] Available at <evobi.in/bibox.php> [Accessed 21 October 2009]

Yahoo Pipes. [Online] Available at <http://pipes.yahoo.com> [Accessed 22 November, 2009.].

Scratch. [Online] Available at <http://scratch.mit.edu> [Accessed 21 November 2009].

NXT Software. [Online] Available at <http://mindstorms.lego.com/en-us/whatisnxt/default.aspx> [Accessed 20 November 2009].

Author Information

Nikhil Karwall is an independent design consultant working in the streams of Product Design and Interaction Design since 2007. He holds a Master's in Design from the Industrial Design Centre at IIT Bombay and a Bachelor's in Mechanical Engineering from NIT Jalandhar. He has two publications and a patent to his credit. He received the Businessworld-NID Design Excellence Award for his design work as a student in 2008 and the Best Paper Award for a paper presented at the Conference on Advances in Usability Engineering in 2008.