CS-308-2014 Final Report

Visual Programming Interface for Firebird V

emBed’em

TH-2

Aashay Harlalka, 100050035  
Nishanth Dikkala, 100050052  
Nivvedan S, 100050084  
Vipul Singh, 100050057

**Table of Contents**

1. Introduction................................................................................................................................... 3

2. Problem Statement........................................................................................................................ 3

3. Requirements................................................................................................................................. 3

3.1 Functional Requirements............................................................................................................ 3

3.2 Non-Functional Requirements.................................................................................................... 4

3.3 Hardware Requirements.............................................................................................................. 4

3.4 Software Requirements............................................................................................................... 4

4. System Design............................................................................................................................... 4

5. Working of the System and Test results....................................................................................... 7

6. Discussion of System..................................................................................................................... 8

7. Future Work.................................................................................................................................... 8

8. Conclusions..................................................................................................................................... 9

9. References....................................................................................................................................... 9

## 1. Introduction

Embedded systems and robotics generate a lot of enthusiasm among a huge section of the population, notably school-going kids. Programming on these systems usually involves C, knowledge of which cannot be expected from such people. To enable them to convert their zeal into useful projects, we wish to remove this programming road-block and provide them a user-friendly visual interface with a high level of abstraction. This is similar to the MIT-Scratch API. Using our interface, one can drag and drop blocks, enter few values and thus, create programs based purely on intuition and run them on the bots.  
Our project aims at developing a rich visual programming environment for Firebird V. This project would be building upon the work by Akshar and Shweta for a course project CS 684, on similar lines and is highly inspired from MIT’s scratch. This rest of this document describes the project in detail.

## 2. Problem Statement

We build a Visual Programming Interface for the ATMEGA-2560 based Firebird-V bot. This allows users to work with high-level abstractions and build programs. Our work follows on another project and we take up the following tasks:

1. Addition of support for the XBee communication module
2. Inclusion of procedures, variables and assignment statements
3. Provision of case studies which show that programming using the interface is easier than writing C code explicitly.
4. Ensure correctness and completeness of earlier version, e.g.: if-then was provided on interface but not implemented; some sensors were left out
5. Add support for different types of data values such as strings and characters apart from the existing support for numbers and boolean values.
6. Automation of the generation of the hex file which can be burnt onto the bot without going through the AVR Studio interface.

**3. Requirements**

### 3.1 Functional Requirements

1. A visual interface with drop-down lists from where user can pick blocks, combine them and create a schematic diagram for some program that he wishes to implement.
2. Ability to generate corresponding C-code from schematic via button-click.
3. Enable user to implement laptop-to-bot communication using the interface.
4. Add support for as many sensors as possible, assignment statements and procedures to the existing version.
5. Add support for different kidns of values not just numbers and boolean. (for eg, strings, characters etc)

### 3.2 Non-Functional Requirements

1. Firebird V ATMEGA2560 architecture only is supported.
2. Application will run on JRE 1.6 and above.
3. Well documented, object-oriented code that will enable future developer to add features with ease.

### 3.3 Hardware Requirements

This is a software-intensive project, the primary requirement being *Visual Programming for Firebird V* (Akshar’s Project) which we build upon.

Following were the hardware requirements:

(i) ATMEGA-2560 based Firebird V bot along with kit- 2 (in order to test for communication)

(ii) XBee Modules - 3 (1 each for the laptop and 2 bots)

### 3.4 Software Requirements

### Netbeans

### AVR Studio 4

### JDK 1.6

### AVRDude (for burning code to the bot in case the bot doesn’t have bootloader installed)

## 4. System Design Overall Design: The big picture is the following. The Java code generates the interface to build programs by connecting blocks and also generates the C code which is passed on to AVR Studio for generation of a hex file which is ultimately burnt onto the bot. The major chunk of the project is the Java-based programming interface design which will be explained here. We define blocks using the lang\_def.xml file.

The language definition file (lang\_def.xml) describes two parts:

* block language
* Workspace, which is the programming environment for the specified block language.

(Note: The lang\_def.dtd files specifies rules, elements, attributes within lang\_def.xml.)

Now the various elements of lang\_def.xml are explained in greater detail under the following sub-headings

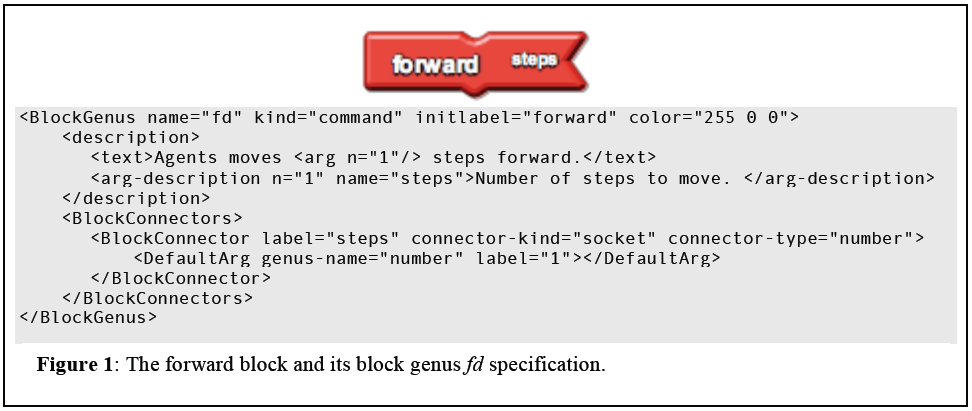
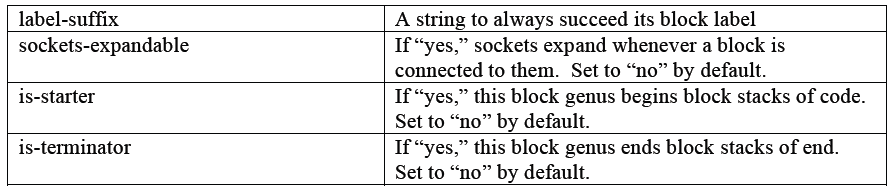
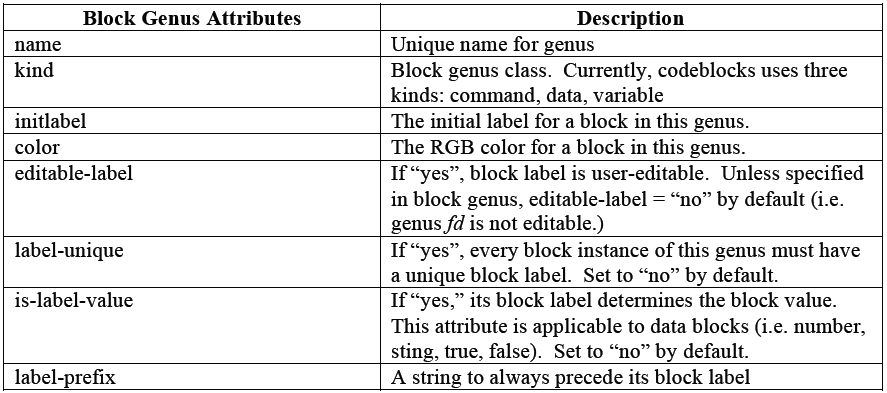
**Block Connector Shape:**We want different kinds of blocks to have different shapes, each of which is designed and drwan in BlockConnectorShape.java. A particular shape is identified by its name and number.

**Block Genus**

A block genus describes the properties that define a common set of blocks. For example,

fd is a block genus that describes all forward blocks in Starlogo. For each block in your

block language, you must specify a BlockGenus.  
An example is shown below.

**Block Family:**

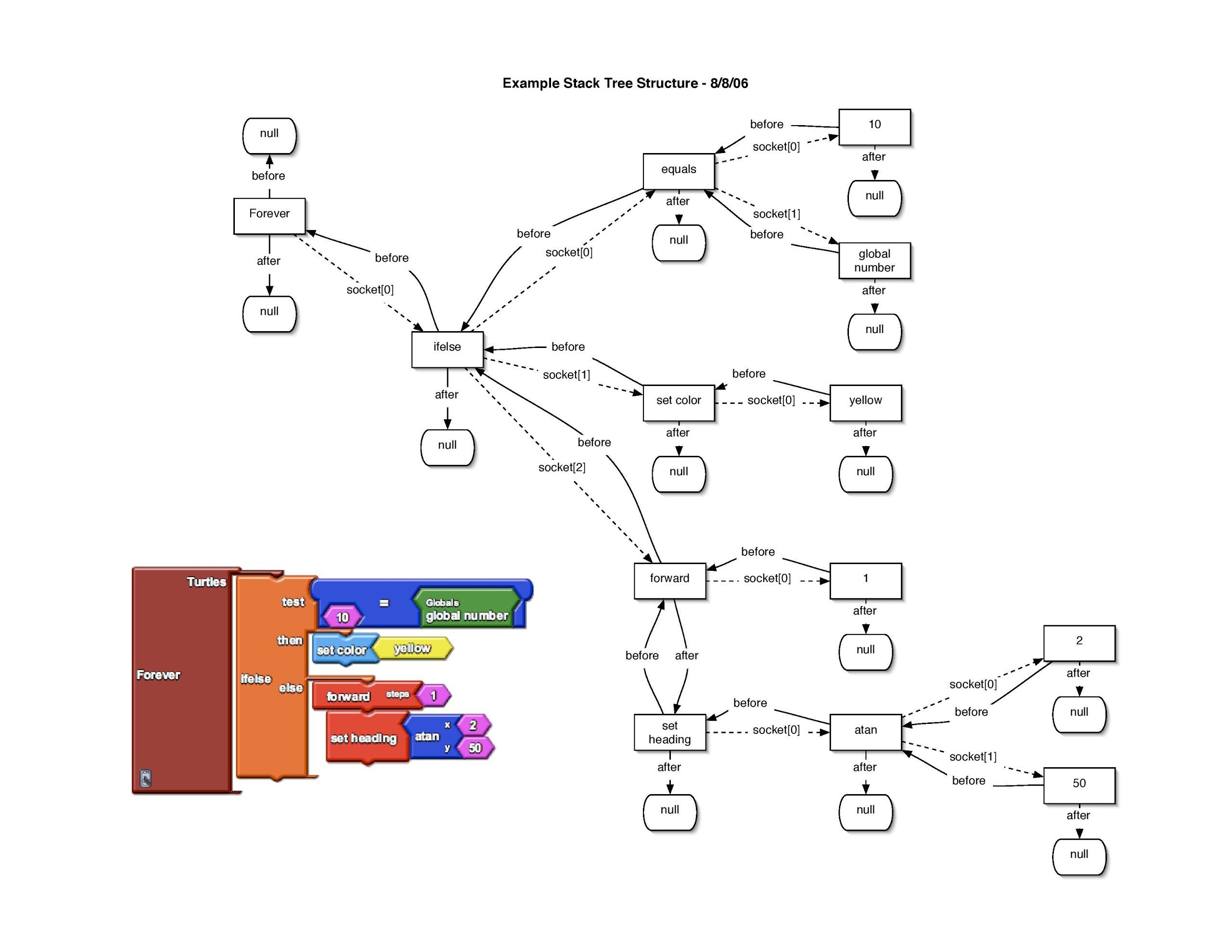
A set of blocks are put in the same family if they are of a similar kind. For example all the mathematical functions form a family similarly all the boolean functions form a different family. The usefulness of putting blocks in families is not only classification but also blocks from the same family can be transformed into one another by the interface without having to place a new block from the search bar. This enhances usability of the product.

**Block Drawer:**

The block drawer decides what blocks are to be shown under what headings in the GUI. As of now, we have   
  
A more detailed description of the block language can be found in the folder Java Code Documentation inside the PDF lang\_def\_spec and redesignV4.

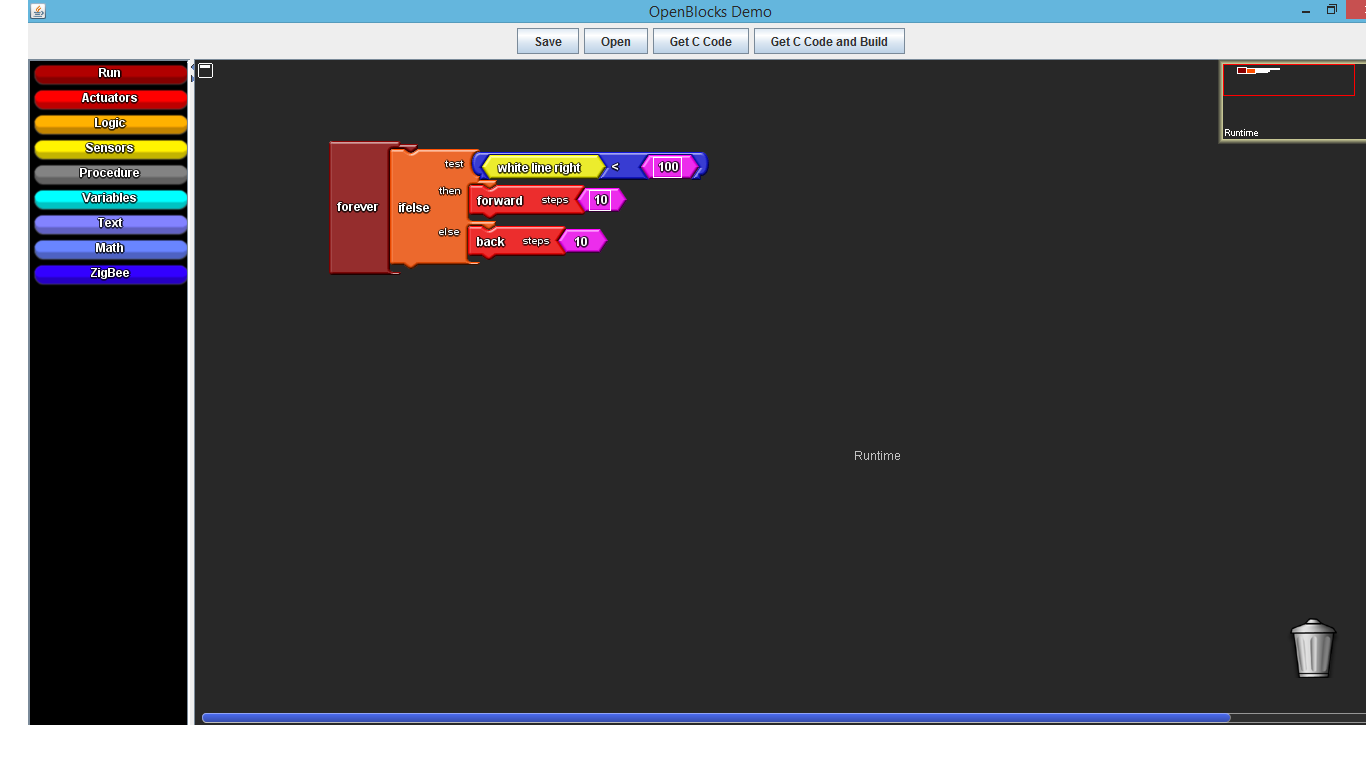
**WorkspaceController.java:**  
This file contains the main function which parses the lang\_def.xml file to create the blocks for the interface. To do so, it uses a number of files in the remaining sub-folders of the src folder. For example, the shape of each block is designed and drawn in the file BlockConnectorShape.java. After creation of all the blocks in the drawer set, they are added to the search bar and 4 buttons are created and placed at the top of the window. These are ‘Save’, ‘Open’, ‘Get C Code’ and ‘Get C Code and build’ buttons respectively.

The generation of C code happens via the generateCode function in the file WorkspaceController.java. It takes the C file given by the user as input and writes code snippets onto it depending on the type of block it is handling. This function is called recursively within blocks to generate code or any arbitrary length.  
The stack structure for a sample program (shown in color in the diagram) is given below.



## 5. Working of the System and Test results

There are two parts to the system - a Java part which handles the graphical interface and generates the C code which is handed over to AVR studio for compilation and creation of a hex file which is ultimately burnt onto the bot. GUI is shown below:



Above Logic generates following code :

#include <firebird.h>

unsigned int count;

void main(){

init\_devices();

while(1){

if((white\_line\_right() < 100)){

move\_forward(10);

}

else{

move\_backward(10);

}

count++;

}

}

The testing strategy for any feature in the programmin interface is to build the appropriate program and check if it generates correct C code. To check for the correctness of the C code we burn it onto the bot and test if the behavior of the bot is as expected from the program. These two requirements were satisfied for all the developments added to the interface.

Sample (XBee and line follower) XML files submitted in separate folder - (they work as expected)

## 6. Discussion of System

a) What all components of your project worked as per plan?

1. Adding Zigbee support
2. Adding sensor support
3. Addition of programming support for procedures, variable declaration and assignment statements.

b) What we added more than discussed in SRS?

1. Support for ultra-sound sensor.
2. LCD display functions.
3. Support for procedures
4. Support for variable declaration and assignment statements.

c) Changes made in plan from SRS:

1. Animated testing interface removed from plan. The reason being that it took us a lot of time to understand the huge code-base and to get accustomed to ATMEGA instead of ARM processor. So, we decided to first move towards functional completeness by adding modules for communication and other sensors, and providing test-case scenarios for future users and developers, instead of going for a validity-testing interface.
2. Automation of build underway (yet to be completed): We made a windows batch file containing the commands to generate the hex file but due to some reason the cmd terminal crashes when the file is run. We are trying to debug this and we are confident this feature will be added by the end of the semester.

## 7. Re-usability and Future Extensions

**Re-usability:**

Our application has been designed specifically for the ATMEGA processor. However, the bulk of our code, except for the C Code generation modules can be re-used exactly as it is for working with other processor architectures like ARM, etc. Thus the major chunk of our code is reusable.

**Extensions:**

1. **Extension to other processor architectures** - The project can be extended to be able to create the C Code for other processor architectures. Infact the application could be modified to generate only an abstract API code, with the processor-specific code abstracted in libraries.
2. **Animated testing interface** - the bot can be projected as a dot on screen and then, via an input interface, we feed data about the environment in the form of different sensor readings and observe the simulation of the bot’s actions on screen. This will be immensely helpful in checking validity of code without actually burning onto the bot.
3. **Functional completeness** - add support for all IR sensors on the bot, position encoders, TSOP sensor and LED display.
4. **Error handling within the interface itself** - currently, through the fixed number of parameters and shapes that can fit into a block, we are able to ensure syntactic correctness on the visual interface itself. In future, this can be extended to ensure semantic validity too, for example, a variable should be assigned a value and not vice-versa.
5. **Better user experience** by addition of features to the interface such as: making copy of blocks already created on the page, a drop-down menu on right clicking to delete the selected block or perform other operations on it etc.
6. **Automation of AVR build** - currently user has to burn the code on the bot manually, It can be done automatically. This will be immensely helpful for the school kids.
7. More complicated case studies and tutorials for the non-trivial cases.

## 8. Conclusions

We have improved upon the Visual Programming Interface for Firebird V developed by Akshar and Shweta. We faced some challenges such as unfamiliarity with development for the ATMEGA-2560 processor and also with the huge pre-existing code-base. With help from the course associates and lab personnel, we have been able to overcome these and significantly add to the earlier version. The XBee communication module was an important component that was missing earlier and has now been added. Then, we have incorporated ultra-sonic range sensors into the interface and also given sample xml schematics to enable future user to understand and appreciate the project quicker and better. Once automation of AVR build is successful, this project has the potential to be launched as an open-source product on the lines of the similar interface by Lego.

## 9. References

* SCRATCH: <http://info.scratch.mit.edu/Research>
* OpenBlocks: <http://education.mit.edu/drupal/openblocks>
* AVR Studio
* Akshar’s documentation