ONLINE MODEL BASED VISUAL PROGRAMMING LANGUAGE (OMBVPL)

By

Usama Ahmed

Rabia Naghman

Abeera Zainab

Submitted to the Faculty of Computing Software Engineering
National University of Sciences and Technology, Islamabad
in partial fulfillment for the requirements of a B.E Degree in
Computer Software Engineering

JUNE 2018

ABSTRACT

In computing, a **visual programming language** (**VPL**) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols, used either as elements of syntax or secondary notation. For example, many VPLs (known as *dataflow* or *diagrammatic programming*) are based on the idea of "boxes and arrows", where boxes or other screen objects are treated as entities, connected by arrows, lines or arcs which represent relations.

CERTIFICATE FOR CORRECTNESS AND APPROVAL

It is certified that work contained in the thesis “Online Model Based Visual Programming Language”was carried out by Usama Ahmed, Rabia Naghman and Abeera Zainab under supervision of Dr. Tauseef Rana for partial fulfilment of Degree of Bachelor of Software Engineering is correct and approved.

Approved by

_____

(Assistant Professor Tauseef Ahmed Rana, PhD)

Department of Computer Software Engineering

Project Directing Staff (DS)

Military College of Signals

National University of Sciences and Technology

Dated: ___ June 2018

This page is left intentionally blank.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

To Allah belongs the dominion of the heavens and the earth; He creates what he wills. He gives to whom He wills female [children], and He gives to whom He wills males. Or He makes them [both] males and females, and He renders whom He wills barren. Indeed, He is Knowing and Competent.

(Chapter25: Surah Ash-Shura: Ayat 49-50)

Dedicated to our beloved families and our country Pakistan.

ACKNOWLEDGEMENT

We are grateful to Allah Almighty for giving us strength to keep going on with this project, irrespective of many challenges and troubles.

Next, we are grateful to all our families. Without their consistent support and prayers, a work of this magnitude wouldn't have been possible.

We are very grateful to our Project Supervisor Assistant Professor Dr. Tauseef Rana who supervised the project in a very encouraging and helpful manner. As a supervisor, his support and supervision has always been a valuable resource for our project.

Last but not the least special acknowledgement to all the members of this group who tolerated each other throughout the whole year.

This page is left intentionally blank.

Table of Content

1. Chapter 1: Introduction 1
1.1 Overview 1
1.2 Problem Statement 1
1.3 Approach 1
1.4 Scope 1
1.5 Objectives 2
1.6 Deliverables 3
2. Chapter 2: Literature Review 4
2.1 Introduction 4
2.1.1 Liberherr 4
2.1.2 VanHilst and Notkin: 4
2.1.3 Model Driven Visual programming language 4
3. Chapter 3: Software Requirement Specification 5
3.1 Introduction. 5
3.1.1 Purpose 5
3.1.2 Document Conventions 5
3.1.3 Intended Audience and Reading Suggestions 5
3.1.4 Product Scope 6
3.2 Overall Description 6
3.2.1 Product Perspective 6
3.2.2 Product Function 7
3.2.3 User Classes and Characteristics 7
3.2.4 Operating Environment 8
3.2.5 Design and Implementation Constraints 8
3.2.6 User Documentation 8
3.2.7 Dependencies 9
3.3 External Interface Requirements 9

3.3.1 User Interfaces 9
3.3.2 Hardware Interfaces 9
3.3.3 Software Interfaces 9
3.3.4 Communication Interfaces 9
3.4 System Features 10
3.4.1 Tool Bar 10
3.4.2 Tool Box 10
3.4.3 Work Space 10
3.4.4 Error Detection 10
3.5 Non-Functional Requirements 10
3.5.1 Performance Requirements 10
3.5.2 Safety Requirements 10
3.5.3 Software Quality Attributes 11
4. Chapter 4: Design and Development 12
4.1 Introduction 12
4.1.1 Purpose of this Document 12
4.1.2 Scope of the Development Project 12
4.1.3 Definitions, acronyms, and abbreviations 13
4.1.4 Overview of document 13
4.2 System Architecture Description 15
4.2.1 Overview of the modules 15
4.2.2 User Interface Issues 21
4.3 Detailed Description of Components 28
4.3.1 Tool Bar 28
4.3.2 Tool Box 28

4.3.3 Work Space 29
4.3.4 Components and Connectors 29
4.3.5 Output 30
4.4 Reusability and Relationships to Other Products 30
4.5 Design decisions and tradeoffs 30
4.6 Pseudo Code for Components 31
4.6.1 New Program 31
4.6.2 Execute the Program 31
4.6.3 Delete Program 31
5. Chapter 5: Project Analysis and Evaluation 32
5.1 Introduction 32
5.1.1 Testplan Identifier 32
5.1.2 Test Items 32
5.1.3 Features tobe tested 33
5.1.4 Features not tobe tested 33
5.1.5 Approach 33
5.1.6 Itempass/failcriteria 34
5.1.7 Suspension criteriaandresumption requirements 34
5.1.8 Test deliverables 35
5.1.9 Testing tasks 39
5.1.10 Environmental needs 40
5.1.11 Responsibilities 40
5.1.12 Staffing and training needs 40
5.1.13 Schedule 41
5.1.14 Risks and contingencies 41

5.1.15 Approvals 41

6. Chapter 6: Future work 42

6.1 Introduction 42

6.1.1 GUI feature 43

For now we are prompting user for input via textbox. In future we can add GUI feature by which user can develop beautiful GUI and can offer more user friendly input method. User can also develop beautiful GUI to make different type of games. 43

6.1.2 Database 43

In future we can add database feature of any one of the famous database. By using which user can add his data to database and even make login page for other users who are going to use. E.g. we can add firebase feature to our ombvpl tool. Firebase provide many features such as firebase login, storage space, realtime database, cloud messaging and much more. 43

6.1.3 More component 43

We can add more components to existing ombvpl tool in future. Components such as math functions like square, square root can be added. We can add event listeners also. 43

Future work also includes the evaluation of the framework using models that utilize the support for IEEE floating-point arithmetic. 43

Further research will include the development of test-suite generation algorithms that exploit the strong isolation of the components and the information about the structure of their composition. 43

7. Bibliography 44

8. Appendix 45

Table of Figures/Tables

1 TABLE 1 - 1 3
2 FIGURE 4 - 1 OVERVIEW OF MODULES 15
3 FIGURE 4 - 2 SYSTEM ARCHITECTURE 16
4 FIGURE 4 - 3 OVERALL STRUCTURE OF THE SYSTEM 17
5 FIGURE 4 - 4 USE CASE DIAGRAM 18
6 FIGURE 4 - 5 CLASS DIAGRAM 19
7 FIGURE 4 - 6 USER INTERFACE ISSUES 21
8 FIGURE 4 - 7 EXECUTE PROGRAM ACTIVITY 23
9 FIGURE 4 - 8 EDIT PROGRAM ACTIVITY 23
10 FIGURE 4 - 9 DELETE PROGRAM ACTIVITY 24
11 FIGURE 4 - 10 EXECUTE PROGRAM SEQUENCE 25
12 FIGURE 4 - 11 EDIT PROGRAM SEQUENCE 25
13 FIGURE 4 - 12 DELETE PROGRAM SEQUENCE 26
14 FIGURE 4 - 13 PROTOCOL USAGE 26
15 FIGURE 4 - 14 PROTOCOL 27
16 FIGURE 4 - 15 OMBVPL INTERFACE 27
17 TABLE 4 - 1 28
18 TABLE 4 – 2 28
19 TABLE 4 - 3 29
20 TABLE 4 -4 29
21 TABLE 4 - 5 30
22 TABLE 5 – 1 35
23 TABLE 5 - 2 36
24 TABLE 5 - 3 36
25 TABLE 5 - 4 37
26 TABLE 5 – 5 37
27 TABLE 5 – 6 38
28 TABLE 5 – 7 38
29 TABLE 5 - 8 39
30 TABLE 5 – 9 39
31 TABLE 5 - 10 40
32 TABLE 5 - 11 40
33 TABLE 5 – 12 41

This page is left intentionally blank.

Chapter 1: Introduction

1.1 Overview

Innovations like the graphical user interface have exposed basic elements like the filesystem to a wider audience, and the Internet has become increasingly democratized as user-friendly tools like WordPress, Youtube and Soundcloud allow anyone to create, publish and distribute content without writing a line of code. Today an explosion of accessible prototyping kits is making it possible for amateurs and hobbyists to sink their teeth into the growing Internet of Things by cobbling together connected computing projects.

But when it comes to making that hardware do your bidding, most tinkerers will still encounter a "language barrier". Even the most user-friendly development boards need to be programmed; and even the simplest programming languages still look like alphabet soup to the uninitiated.

Fortunately, developers like us have started to step in and provide user-friendly, visual programming tools. These platforms abstract away the functions, variables and idiosyncratic syntax rules of the underlying code and give users a simple drag-and-drop interface for building apps out of discrete chunks of logic.

This project provides an environment for students to learn the basic concepts of programming by using the graphical elements and connecting them to create a program.

1.2 Problem Statement

Our aim is to develop a visual programming language which will provide the students of computer science a platform to learn the basic concepts of programming.

1.3 Approach

To create a visual programming language which is based on a model. The model-based approach makes it extendable and new functionality can be added as needed.

1.4 Scope

The motivation behind this project is to help out anyone who doesn't know how to code by moving the focus of work from programming to solution modeling. It will be done by constructing and transforming models that can be round-trip engineered into code which will increase development productivity and quality by describing important aspects of a solution with more human-friendly abstractions and by generating common application fragments with templates.

The one who will be using this product will have no involvement in the background coding being done by the developer. The user will only be able to use visual expressions, spatial arrangements of text, graphic symbols and will drag & drop the objects, being provided by this

product but will have no access to the backend processes and hence will be able to create programs without having any knowledge regarding programming.

Thus “Online Model Based Visual Programming” provides a set of predefined components with different libraries for performing the functionalities and connectors which users can use to develop a program by just DRAG and DROP without writing any line of code.

In computing, a visual programming language is any programming language that lets users create programs by manipulating program elements graphically rather than writing a code.

Our aim is to develop a web based tool an online environment where the beginner or programmers with basic understanding of concepts like variables and logic can be assisted

This tool may appeal to more advanced programmers for rapid prototyping or code development

1.5 Objectives

During the course of this project, all the aspects of software engineering will be covered i.e. requirement gathering, software design, implementation and testing along with documentation (SRS, SDS, Test Document, Final Report and User manual).

This project intends to implement the fundamentals of the x-man model designed by University of Manchester. X-MAN component model is a framework that’s laid down on the foundation that defines “Separation of concern”, i.e. that the computation and control are being dealt with at two separate levels of encapsulation.

We aim to develop the IDE based on X-Man model that will provide the user with atomic components encapsulated in separate files and the composition connectors which will provide coordination control among a set of components of a program.

In the X-MAN component-based approach, components are constructed from two kinds of basic entities: (i) computation units, and (ii) connectors. A computation unit U encapsulates computation. It provides a set of methods. Encapsulation means that U’s methods do not call methods in other computation units; rather, when invoked, all their computation occurs inside U. Thus, U can be thought of as a class that does not call methods in other classes.

A composition connector encapsulates control. It is used to define and coordinate the control for a set of components. Connectors form a hierarchy i.e., composition is performed in a hierarchical manner. Furthermore, each composition preserves encapsulation. This kind of compositionality is the distinguishing feature of the X-MAN approach. A single component encapsulates computation, namely the computation encapsulated by its computation unit. A composite component encapsulates computation and control. The computation it encapsulates is that encapsulated in its sub-components; the control it encapsulates is that encapsulated by its composition connector. In a composite, the encapsulation in the sub-components is preserved. Indeed, the hierarchical nature of the connectors means that composite components are self-similar to their sub-components, i.e., composites have the same structure as their sub-components; this property provides a basis for hierarchical composition. In general, a system

constructed using this approach consists of a hierarchy of composition connectors sitting atop a flat layer of decoupled single components

1.6 Deliverables

Sr.	Tasks	Deliverables
1	Literature Review	Literature Survey
2	Requirements Gathering	SRS Document
3	Application Design	Design Document (SDS)
4	Implementation	Implementation on computer with a live test to show the accuracy and ability of the project
5	Testing	Evaluation plan and test document
6	Training	Deployment Plan
7	Deployment	Complete application along with necessary documentation

1 Table 1 - 1

Chapter 2: Literature Review

2.1 Introduction

Previous work done on this idea are discussed in this chapter. There were a few projects that were based on the idea of thought recognition following is a detailed description of projects previously carried out in this context.

Liberherr

For adaptive OOP, a system was produced by Liberherr who used graph-based customization.Adaptive OOP helps to view such items which are important for an application as they do not commit themselves to a class structure of the respective application. In order to develop a resultant system, a class structure which is compatible can be used which can be automatically integrated.

VanHilst and Notkin:

Using templates of the class for programming in an unstructured manner, a method was generated by VanHilst and Notkin. Some aspect/behavior was expressed by each class used. When combine the class templates or behaviors by using inheritance, an object can be created which is the result. Thus, a small piece of code has this structure whereas the rest of the code has unstructured pieces relatively.

Model Driven Visual programming language

This visual language was developed by umer namdar and his team. This is visual programming language made using eclipse. They have used model driven approach. They were generating code in java to be run in backend. They were also providing different component to the user which user can drag and drop and connect to make new program. But there project was not based on any model.

Chapter 3: Software Requirement Specification

3.1 Introduction.

In this chapter we are going to discuss purpose of SRS. Its document conventions, who are intended audience. In general this chapter is going to provide overview of ombvpl in term of explanation of what we are going to develop. It also contain some diagram which explains ombvpl even further. It contains class diagram by which one can get brief overview of how the user will interact with the ombvpl tool.

Purpose

This document details the software requirements specification for the Online Model Based Visual Programming Language. It reflects all the requirements, constraints and design activities of this project. The release number of the software is 1.0.

In designing a software product model based approach is used which is a set of rules that defines the basic structure and logics of the language, expressed as a model, used in the project.

A Visual Programming Language is a programming language which enables the user to create programs by using pre-defined graphical objects/elements instead of using the conventional way of creating programs i.e. by writing the lines of code in any programming language.

The main purpose of developing the Online Model Based Visual Programming Language is to provide those people with the platform to create programs who have no or very little knowledge about software development and who do not have any level of skill in any programming language.

Document Conventions

Italics: The words in italics are further explained in the glossary.

Intended Audience and Reading Suggestions

The Intended audience for this document is listed below:

3.1.3.1 Examiners/Evaluators

The document will provide the FYP evaluators with the scope, requirements and details of the project to be built. It will also be used as basis for the evaluation of the implementation and final project.

3.1.3.2 Developers

The document will provide guidance to the developers to determine what the requirements are and how they should continue with the project.

3.1.3.3 Project Supervisor

This document will be used by the project supervisor to check whether all the requirements have been understood and in the end whether the requirements have been implemented properly and completely.

3.1.3.4 Project Testers

Project testers can use this document as a base for their testing strategy as some bugs are easier to find using a requirements document. It will help in building up test cases for the testing process. This way testing becomes more methodically organized.

3.1.3.5 Up gradation Engineers

Up gradation engineers can review projects capabilities and more easily understand where their efforts should be targeted to improve or add more features to it. It sets the guidelines for future developments.

3.1.3.6 End Users

This document can be read by the end users if they wish to know what the project is about and what requirements have been fulfilled in this project.

Product Scope

This project will assist anyone who has no knowledge about coding by orienting their focus of work from programming to solution modeling. It will be done by constructing a model which will be engineered into code in such a way that will help to produce such abstractions and graphical templates which will help a novice to get along with this language very easily.

The one who will be using this product will have no involvement in the background coding being done by the developer. The user will only be able to use visual expressions, spatial arrangements of text, graphic symbols and will drag & drop the objects, being provided by this product but will have no access to the backend processes and hence will be able to create programs without having any knowledge regarding programming.

3.2 Overall Description

Product Perspective

In order to write a software program, one does not need to have a good know how of the popular programming languages which include C, C++, and Java etc. At the student level the aim should be to develop basic understanding of programming concepts such as making decisions, writing code in loop etc. Thus, in order to solve this problem, we introduce this

“Online Model Based Visual Based Programming” which contains set of predefined components which users can develop basic understanding of programming concepts by using drag and drop. Hence, the program/software can be developed using only the atomic components and the connectors without writing any line of code.

Product Function

This Visual Programming Language have the following functionalities:

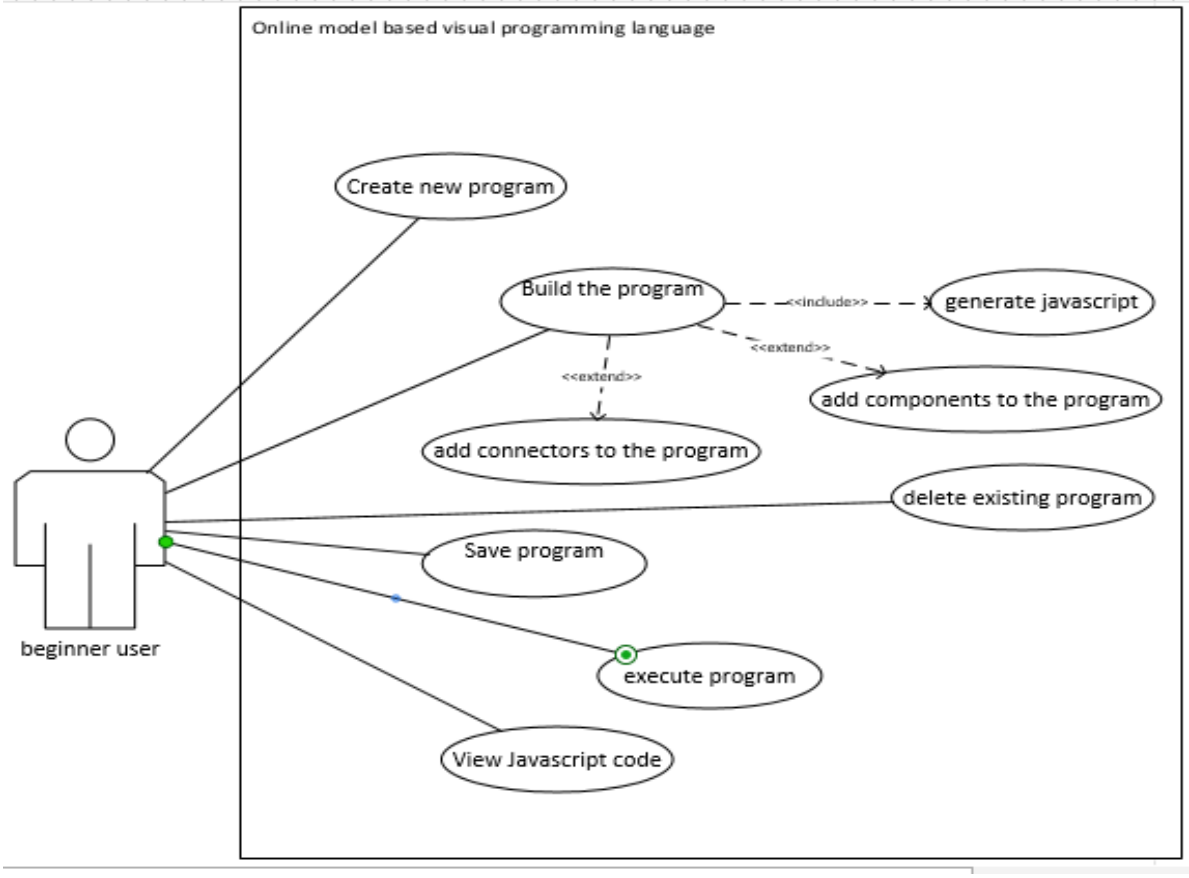
- Provides graphical units/elements which users can drag and drop.
- Each unit/element performs a specific function.
- These graphical units/elements can be integrated together to create a working program or software.
- There is no need to write any line of code.
- All the code is generated automatically at the back end.

User Classes and Characteristics

Following are our targeted users:

- Beginners: The students of school or colleges who have no knowledge of programming.

User and his interaction with the system is shown in Figure A.



1 Figure 3 - 1

Operating Environment

- Microsoft Windows/Mac OS/Linux
- HTML Code Editor: Visual Studio Code
- JavaScript enabled web browser

Design and Implementation Constraints

- The user can only use the predefined graphical elements to create a program
- The language based on the X-man component model
- User can create programs by following the tutorials given in the user manual.

User Documentation

A user manual will be provided which will help new users to get started with the Online Visual Programming Language. The user manual will provide the instructions on how to work with this Online Visual Programming Language.

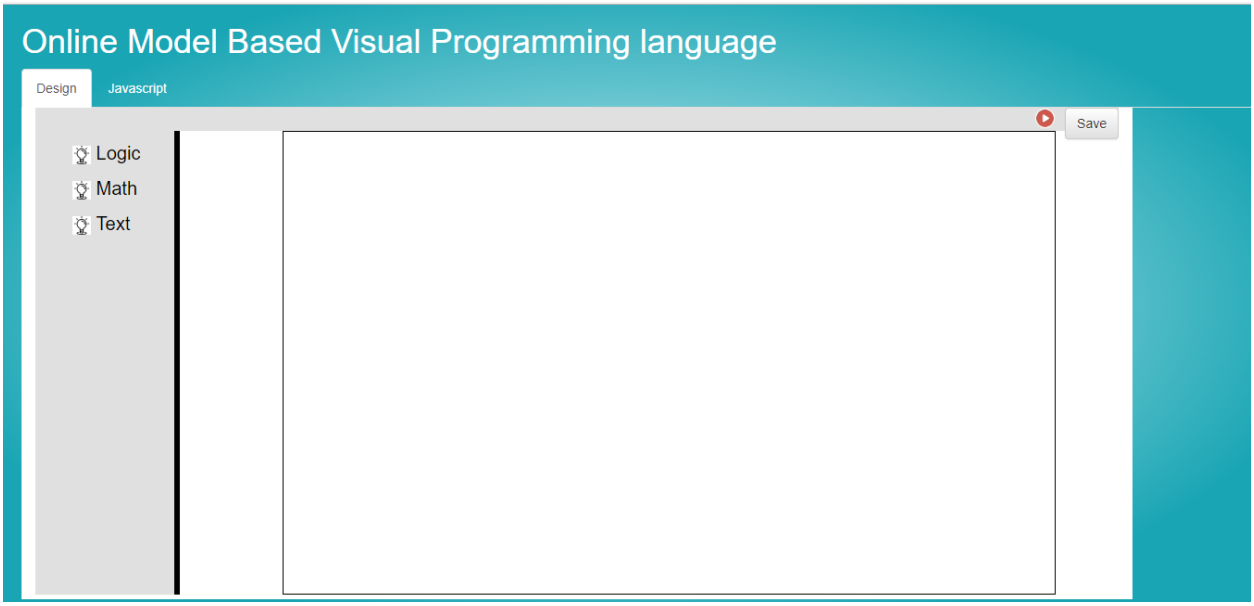
A summary will also be provided to the user which will highlight the features and limitations of this language.

Dependencies

- The system requires JavaScript enabled web browser to run.
- The system also requires bootstrap framework files.

3.3 External Interface Requirements

User Interfaces



2 Figure 3 - 2

Hardware Interfaces

We do not require any hardware for this project.

Software Interfaces

This Online Model Based Visual Programming Language will be developed in HTML, CSS and JavaScript using visual studio code.

JavaScript enabled web browser will be required to run this tool.

Communication Interfaces

N/A - No communication interfaces are required.

3.4 System Features

Tool Bar

The tool bar shall provide the options to save current project. The tool bar will also have the button to run the program.

Tool Box

The tool box shall contain all the graphical objects/elements i.e. connectors and components which the user will drag and drop to make programs.

Work Space

The editor where the graphical objects will be dropped to interact with each other.

In the work space the user will create his/her own logic and implement the objects/elements according to the requirements to obtain the desired program/software.

Error Detection

Whenever the user will make any syntax error or logical error or give invalid input the error will be detected and the error will be displayed on the screen.

3.5 Non-Functional Requirements

Performance Requirements

3.5.1.1 Response Time

The response time of the system is very less.

3.5.1.2 Capacity

At a time user would be able to execute only one program.

Safety Requirements

The system editor shall not crash accidently even if a program fails to execute.

The user program will be saved automatically after some time, so in case of system shutdown the user will not lose the program.

Software Quality Attributes

3.5.3.1 Extensibility and Maintainability

The graphical user interface of app is to be designed with usability as the first priority. The app will be presented and organized in a manner that is both visually appealing and easy for the user to navigate or play.

3.5.3.2 Portability

This tool can be used and moved to any device having JavaScript enable web browser.

3.5.3.3 Reliability

This software will not fail in any condition.

3.5.3.4 Availability

The application will always be available unless user close the browser or his personal computer.

3.5.3.5 Flexibility

The design and architecture of the application will be flexible enough for catering any new requirements, if any at some later stage or for the application enhancement.

3.5.3.6 Usability

This tool provides easy to understand components and connectors with name of each component and connector written above each components/connector.

Chapter 4: Design and Development

4.1 Introduction

This design document contains all functional requirements and displays their relationships with each other conceptually. This document also shows our planning towards implementation of our project. It contains diagrams which shows the design of the language and user interaction with the language followed by their responses. This document also consists of some tradeoffs of few aspects of the design, intended to be included.

Purpose of this Document

The aim of this document is to present the detailed design description of our project. It will explain the aim and features of the language, the interface of the IDE, which graphical objects it provides, the model on which it will work, the constraints under which it must operate and how the language will run the different programs. This document is intended for both the stakeholders and the developers of the system. It will explain that how users with no programming skills create programs by dragging and dropping objects from predefined set of elements.

,

Scope of the Development Project

A Visual Programming Language is a programming language which enables the user to create programs by using pre-defined graphical objects/elements instead of using the conventional way of creating programs i.e. by writing the lines of code in any programming language.

The main purpose of developing the Model Based Visual Programming Language is to create an online integrated development environment in order to help out the beginner programmers to create programs visually who have very little knowledge about codingand do not have any level of skill in any programming language. This project will assist anyone who has no knowledge about coding stuff by orienting their focus of work from programming to solution modeling. It will be done by using the basic principles laid down by the X-man component Model. Basically, the scope of our software defines “Separation of concern”, i.e. that the computation and control are being dealt with at two separate levels of encapsulation. The IDE will provide the user with atomic components encapsulated in separate files and the connectors which will provide control to the program. In this way the programming will be made easier to understand for the programmer. The coder who will be using this tool will have no involvement in the background coding being done by the developer. The user will only be able to use visual expressions, spatial arrangements of text, graphic symbols and will drag & drop the objects, being provided by this tool but will have no access to the back-end processes and hence will be able to create programs without having any knowledge regarding programming.

Definitions, acronyms, and abbreviations

Visual Programming language: In computing, a visual programming language (VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually.

Our project is based on X-MAN component model.X-MAN component model is a framework that's laid down on the foundation that defines "Separation of concern", i.e. that the computation and control are being dealt with at two separate levels of encapsulation.

4.1.3.1 Visual Programming language:

In computing, a visual programming language (VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually.

4.1.3.2 Model Based Architecture:

Our project is based on X-MAN component model.X-MAN component model is a framework that's laid down on the foundation that defines "Separation of concern", i.e. that the computation and control are being dealt with at two separate levels of encapsulation.

Overview of document

The document is divided into sections and is already listed in the table of contents and figures list. However, here is a brief description of all the sections.

4.1.4.1 System Architecture Description

Here, the overall architecture of our language is described, including the introductionof various components. It has a system Architecture diagram which shows an insider's perspective of the project by describing the high level software components that perform the major functions to make the project operational.

4.1.4.2 Structure and relationships

In this section, the interrelationships and dependencies among various components are discussed. It is mainly described by a UML Class diagram also helps us understanding the system structure.

4.1.4.2.1 UML Class diagram

UML Class diagram further manifests the description of low level components of the software that include the language editor and graphical elements, thus making the system adequately comprehensible.

4.1.4.3 User Interface Issues

This section ponders upon the main principles of the project's user interface. Not touching about the technical details, the section is described by an overall diagram. Moreover, UML Activity diagrams, UML Sequence diagrams, and UI Design diagrams also elaborate the User Interface issues in a more intelligible manner.

4.1.4.3.1 UML Activity diagrams

UML Activity Diagrams follow a workflow-based approach to describe the overall functioning of the project. They are a very good means to see how various steps are involved in major tasks inside our project using a flow chart pattern without getting into the technical details.

4.1.4.3.2 UML Sequence diagrams

UML Sequence diagrams show how different steps are involved in the completion of a functionality of the project. They have a unique format that allows the reader to see how many graphical objects are used in a sequence for the completion of a system requirement.

4.1.4.4 UI Design

Some snapshots of graphical user interfaces are shown in this section that prototype the way a user shall be interacting with the system.

4.1.4.5 Detailed description of components

This section contains detailed description of all the major components of the system in a structured pattern (table), comprising of 10 x rows. The pattern (table) maintains symmetry in the document structure; and therefore it is followed for each of the components. Each part/row of the table is identified by a label, explaining the purpose of each point. The description of each point vis-à-vis the component being discussed, ponders upon the detailed account of it in the system.

4.1.4.6 Reusability and relationships to other products

This section highlights the Reusability aspects of the various components of the system. Since the project in hand is all new and doesn’t carry out any enhancement work in the already existing system, so Reusability is just a recommended strategy to be employed while organizing various system components.

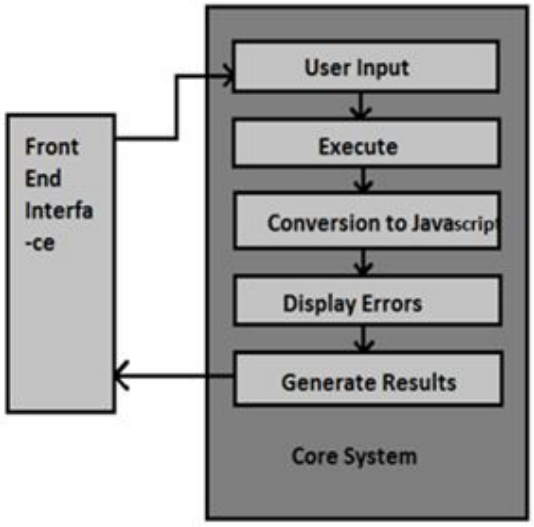
4.1.4.7 Design decisions and tradeoffs

This section focuses upon various design decisions and the ideas behind those. It enables the reader to understand the important crux of the design that is being usedwhile excavating a bit more about the motivations behind those decisions.

4.2 System Architecture Description

Overview of the modules

The system will be architected mainly in 2 fundamental modules “Front End Interface” and “Core System” having other sub modules too as shown in the following abstract diagram:



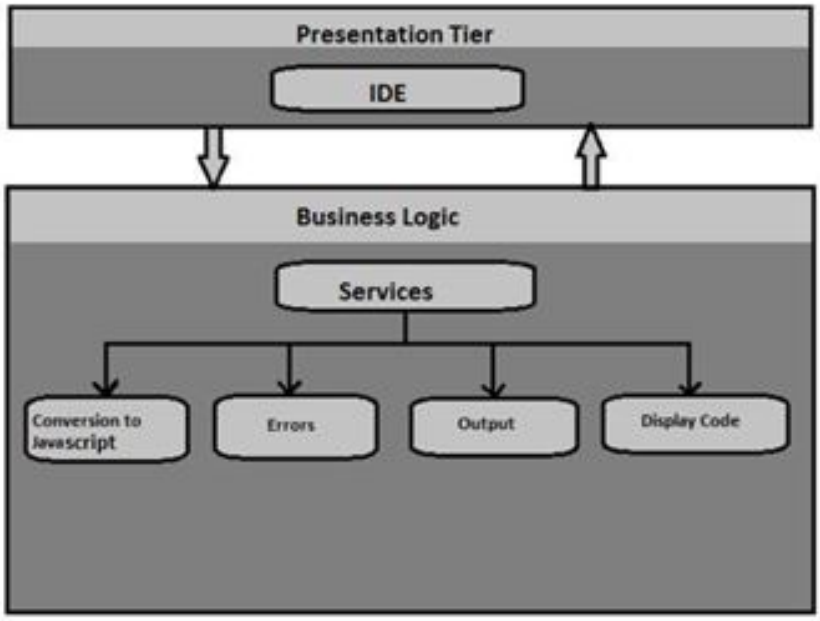
2 Figure 4 - 1 Overview of Modules

4.2.1.1 Description of the modules

The “Front End Interface” will be used by the user to give input by dragging and dropping elements. Once user has created his program, the program will be executed. The elements are converted into JavaScript code, and processed information will lead to generation of final results in “Generate Results” module; where from the result will be sent back to the “Front End Interface”.

4.2.1.2 System Architecture

Layered Architecture (2-Tiers) will be used to implement OMBVPL. First layer is the Presentation layer which comprises of our Language Editor (IDE). Second layer which is the Business Logic layer comprises of various services provided by OMBVPL.



3 Figure 4 - 2 System Architecture

4.2.1.3 Layers Details

The details of the layers have been discussed below.

4.2.1.3.1 Presentation Layer

The presentation layer provides the platform for interaction of the users (Programmer) with the system. It displays Graphical Elements to the user and accepts input from the user. The Presentation layer can only receive input from and return responses to, an outside agent. The Presentation layer also sends acquired input to the Business Logic layer.

4.2.1.3.2 Business Logic

Business Logic has been used as a service layer to expose the business functionality of the tool. Comprising of elements of “Conversion to JavaScript”, “Display Errors”, “Display Output” and “Display Code”, the layer caters for the core functionality of the system.

4.2.1.3.3 Structure and Relationships

Focusing upon the internal structure of the system, this section ponders upon the interrelationships and dependencies among various components.

4.2.1.4 Overall Structure of the system

The diagram shows the main components of the system along with their interactions with each other. It mainly describes the system structure which is further augmented by the explanatory text as follows:

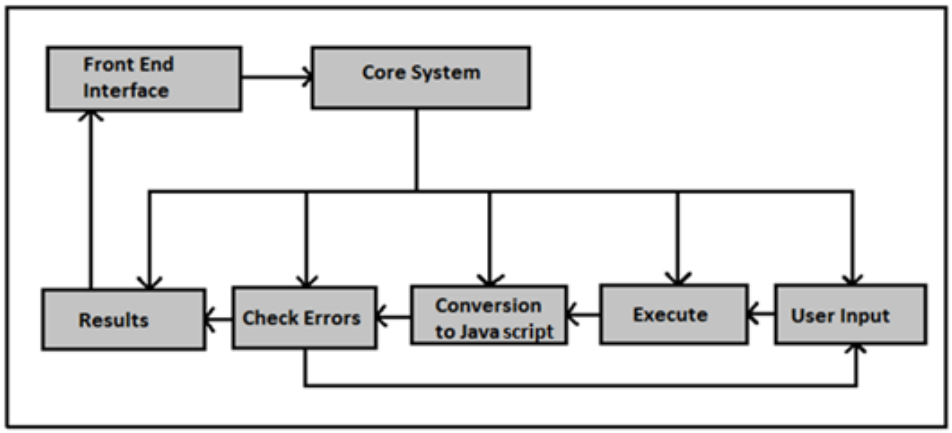


Figure 3 - Overall Structure of the System

4 Figure 4 - 3 Overall Structure of the System

4.2.1.4.1 Front End Interface

Front End Interface caters for the visual needs of the tool, wherein the Human-Computer-Interaction aspects are considered to enable the user to communicate with the system profoundly. It is connected with the Core System, and Generate Results modules to pass on the user inputs as well as display the output feedback to the user respectively.

4.2.1.4.2 Core System

This Core System comprises of execution of User Input by converting user program into JavaScript code. Errors and Generated Results are send to Front End Interface for user to view.

4.2.1.4.3 User Input

User Input is received in the first module (Front End Interface) and then sent to Second module (Core System).

4.2.1.4.4 Execute

In this, User Input is mapped with the model and therefore converted to the corresponding JavaScript code.

4.2.1.4.5 Java Compiler

The converted JavaScript Code is compiled and executed.

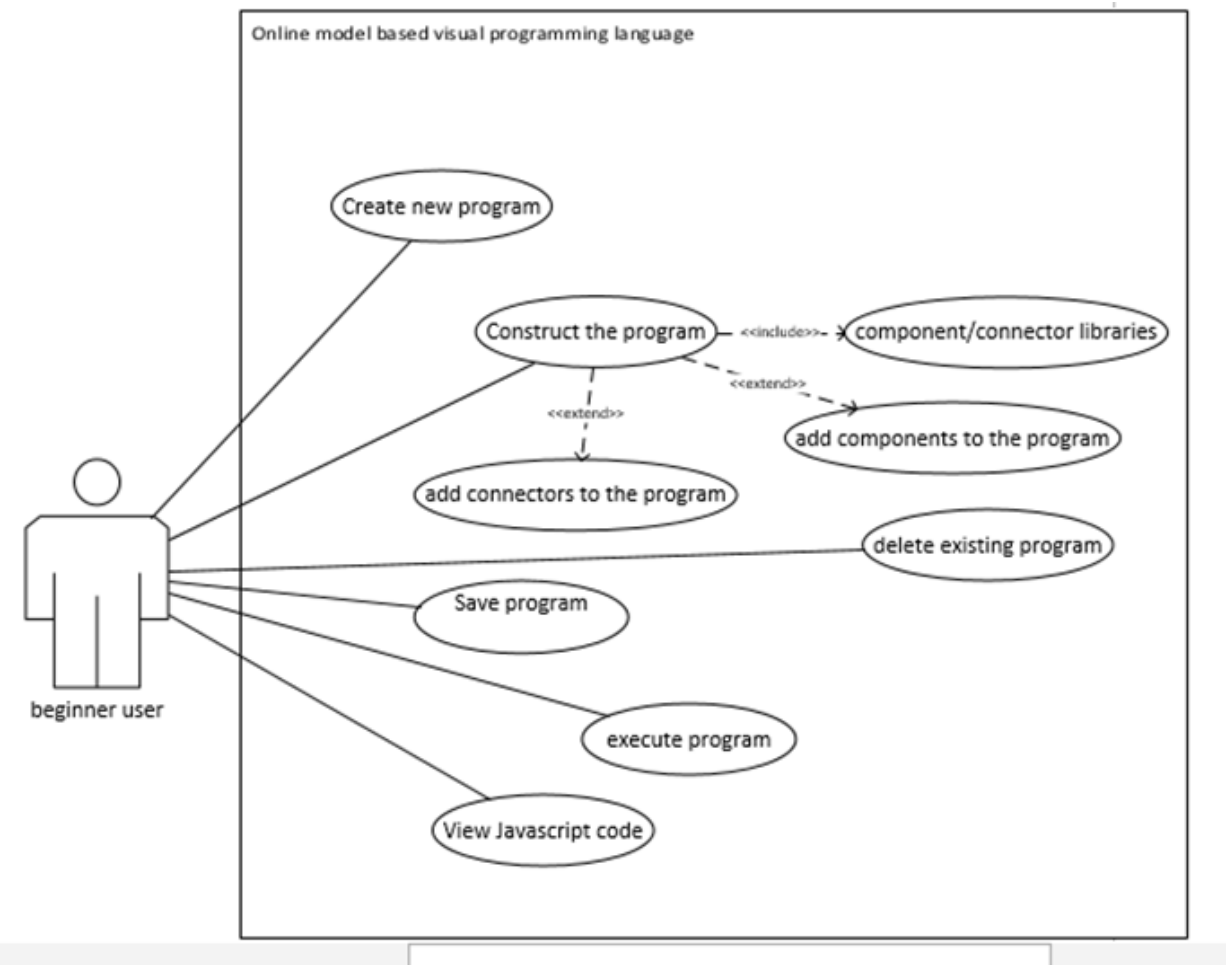
4.2.1.4.6 Check Errors

Errors are checked and displayed to the user. User can edit their program.

4.2.1.4.7 Results

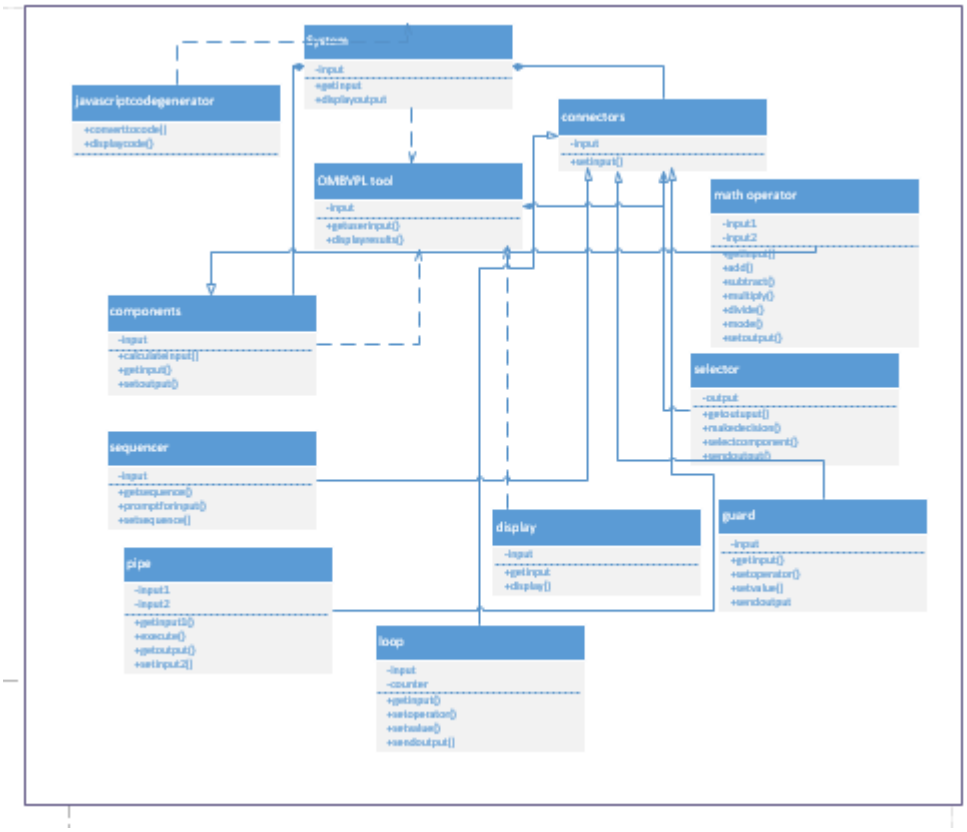
Output is generated and sent to the “Front End Interface” module.

4.2.1.5 Use Case Diagram



5 Figure 4 - 4 Use Case Diagram

4.2.1.6 Class Diagram with Description



6 Figure 4 - 5 Class Diagram

4.2.1.6.1 Description of Diagram

4.2.1.6.1.1 OMBVPL tool

This Class Provides the interface to the user and sends User input to respective classes.

4.2.1.6.1.2 Atomic components

Defines the whole component used for computation and results. It contains invocation connector and computation unit.

4.2.1.6.1.3 Composition Connector

4.2.1.6.1.4 Math operator

Defines components for math operation.

4.2.1.6.1.5 Selector

Based on the input value this component selects amongst the components and the path it has to follow. It behaves just like a switch statement. After the execution of the composed component the results are passed on to the originator connector.

4.2.1.6.1.6 Sequencer

On receiving control, it takes the data from component A and executes it for the output. Then takes data from component B and performs its services, holds the results in the possible sequences for the user to select in which order they expect the execution to be in, then pass the held results to the control originator.

4.2.1.6.1.7 JavaScript Code generator

Defines a class which generates JavaScript code from graphic elements.

4.2.1.6.1.8 Guard

Defines a class which act as an if statement. It corresponds to an if statement, it needs a condition that must be true in order to pass the control to the connected component.

4.2.1.6.1.9 Loop

Defines a class which acts as a for loop or while loop in OMBVPL. Basically, it is a Loop connector with a terminating condition.

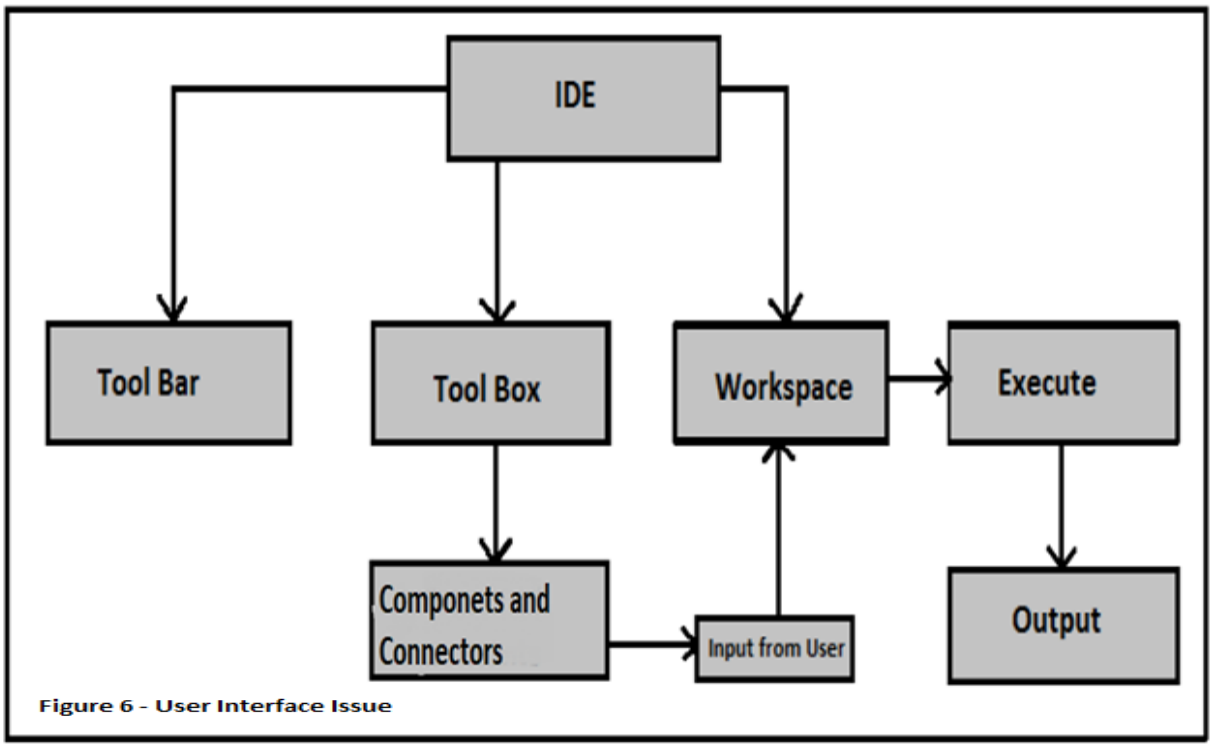
4.2.1.6.1.10 Display

Defines a class which display results on browser.

4.2.1.6.1.11 Pipe

Defines a class which coordinates flow of control in program. Pipe takes the output of component A and gives it as an input to component B.

User Interface Issues



7 Figure 4 - 6 User Interface Issues

4.2.2.1 Description of the diagram

4.2.2.1.1 IDE

It harnesses all the functionalities of the application, including all the interactions involved between the user and the system. It is supported by various buttons to meet all the requirements of the users and enable them to interact with the system.

4.2.2.1.2 Tool Bar

It consists of various functionalities, such as, Creating Project, Saving Project, Open existing Project, Deleting Project, Executing Project, Help about the tool, etc.

4.2.2.1.3 Tool Box

It contains all the Graphical Elements which User can use to create program.

4.2.2.1.4 Components and Connectors

These elements are chosen by the user as per his requirement to create program. They are dragged and then dropped on to workspace.

These elements comprise of:

CONNECTORS:

Sequencer: On receiving control, it takes the data from component A and executes it for the output. Then takes data from component B and performs its services, holds the results in the possible sequences for the user to select in which order they expect the execution to be in, then pass the held results to the control originator.

Selector: Based on the input value this component selects amongst the components and the path it has to follow. It behaves just like a switch statement. After the execution of the composed component the results are passed on to the originator connector.

Pipe: Pipe takes the output of A and gives it as an input to B.

ADAPTERS: (Since this connecter is used directly in the system construction)

- Loop: connector with a terminating condition
- Guard: It corresponds to an if statement, it needs a condition that must be true in order to pass the control to the connected component.
- Invocation connector: It's used at the lowest level to invoke a method in the computation unit. Invocation connector connects to the computational unit to create an atomic component thus providing encapsulation at the component level with separation of concern.

4.2.2.1.5 Workspace

This is used by the user to create his program. This is where he drags and then drops the graphical elements.

4.2.2.1.6 Output

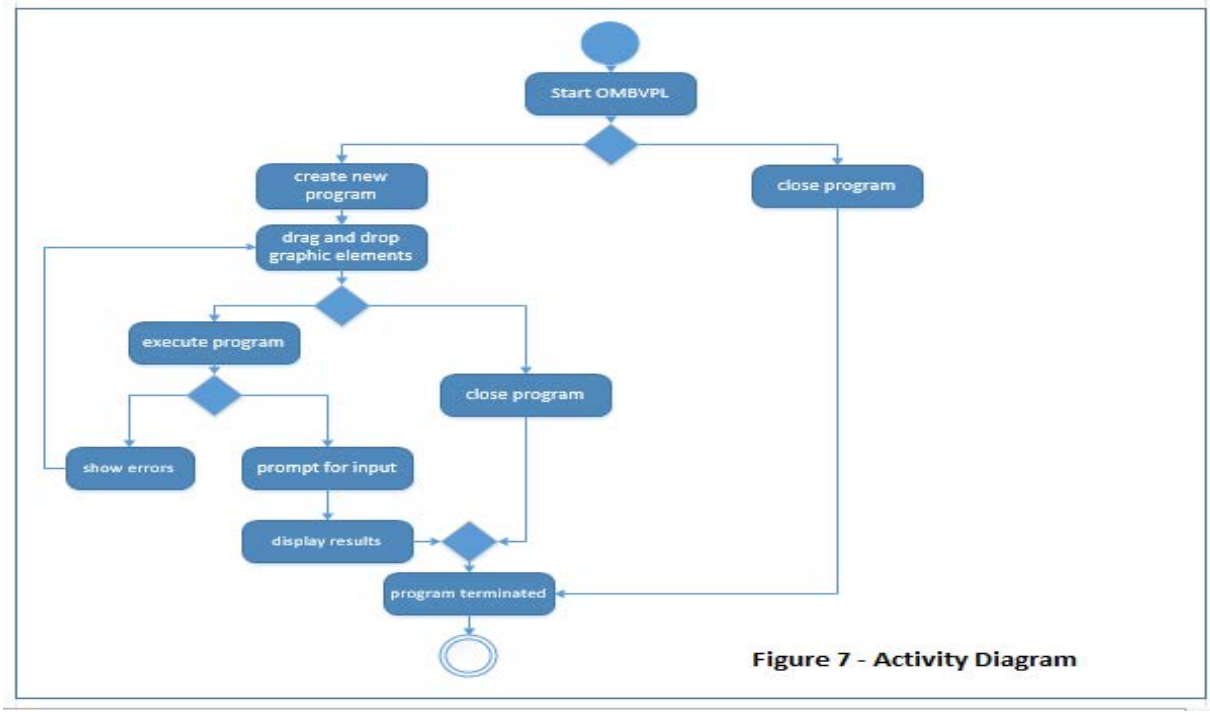
The output generated, once the program is executed, is displayed on the output window.

4.2.2.2 UML Activity Diagram

This section shows the activities that a user need to perform to accomplish a task.

4.2.2.2.1 Execute Program

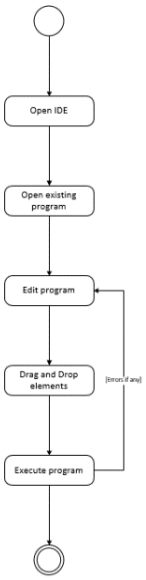
Description: This scenario describes the flow of activities necessary for the user to execute a program.



8 Figure 4 - 7 Execute Program Activity

4.2.2.2.2 Edit Program

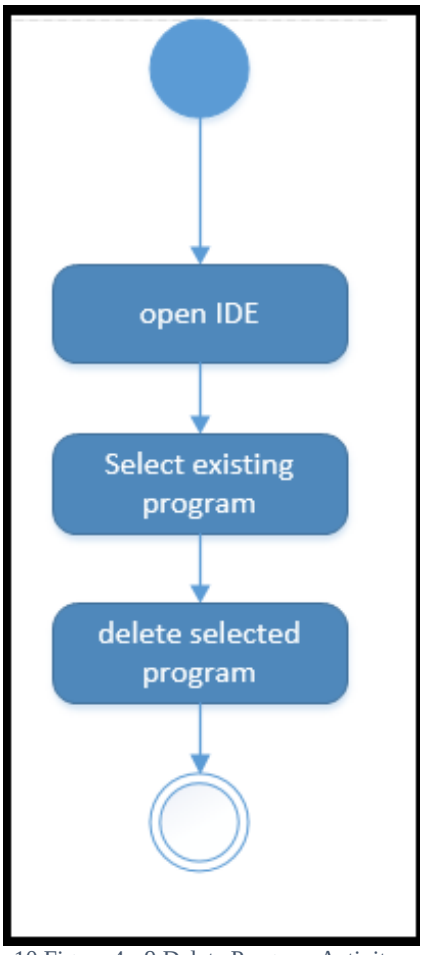
Description: This scenario describes the flow of activities necessary for the user to edit an existing program.



9 Figure 4 - 8 Edit Program Activity

4.2.2.2.3 Delete Program

Description: This scenario describes the flow of activities necessary for the user to delete an existing program.



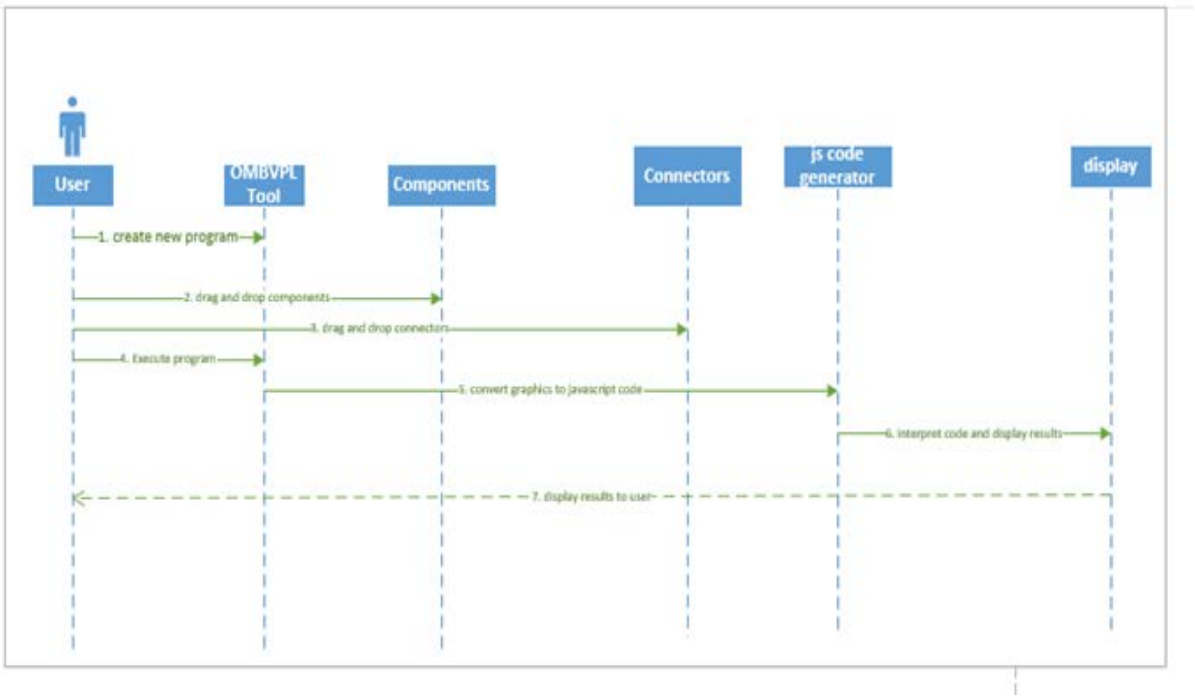
10 Figure 4 - 9 Delete Program Activity

4.2.2.3 UML Sequence Diagrams

Different Scenarios and their corresponding events are discussed in this section with the help of sequence diagrams.

4.2.2.3.1 Execute Program

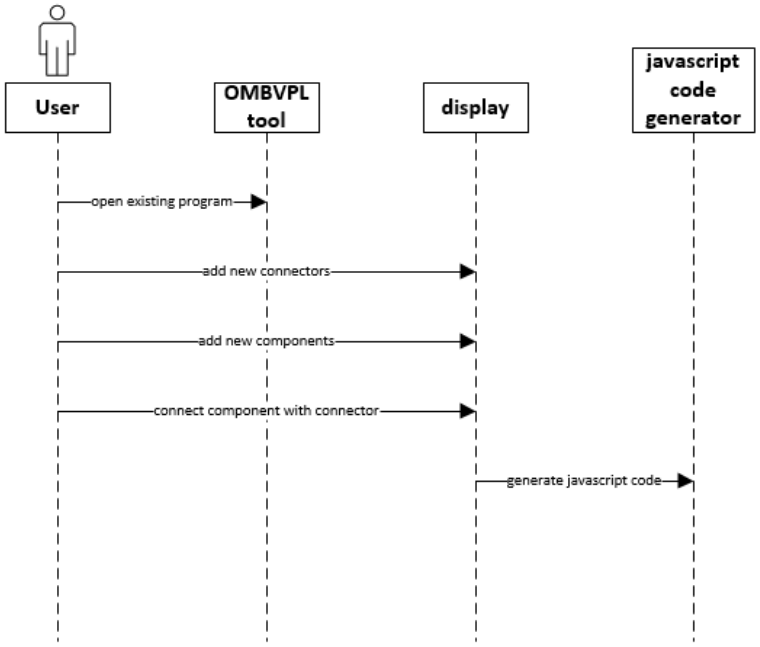
Description: This scenario describes the sequence of events that take place when user executes a program.



11 Figure 4 - 10 Execute Program Sequence

4.2.2.3.2 Edit Program

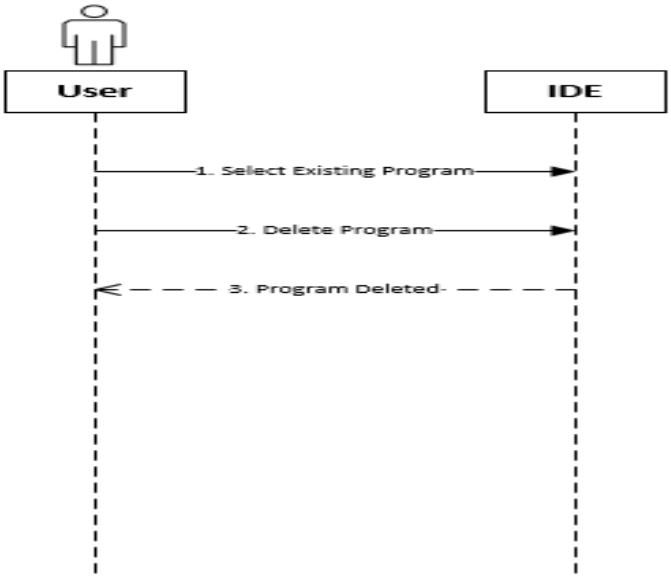
Description: This scenario describes the sequence of events that take place when user edits an existing program. The alternative prospects of the events have also been catered for in case his program contains error.



12 Figure 4 - 11 Edit Program Sequence

4.2.2.3.3 Delete Program

Description: This scenario describes the sequence of events that take place when user deletes an existing program.



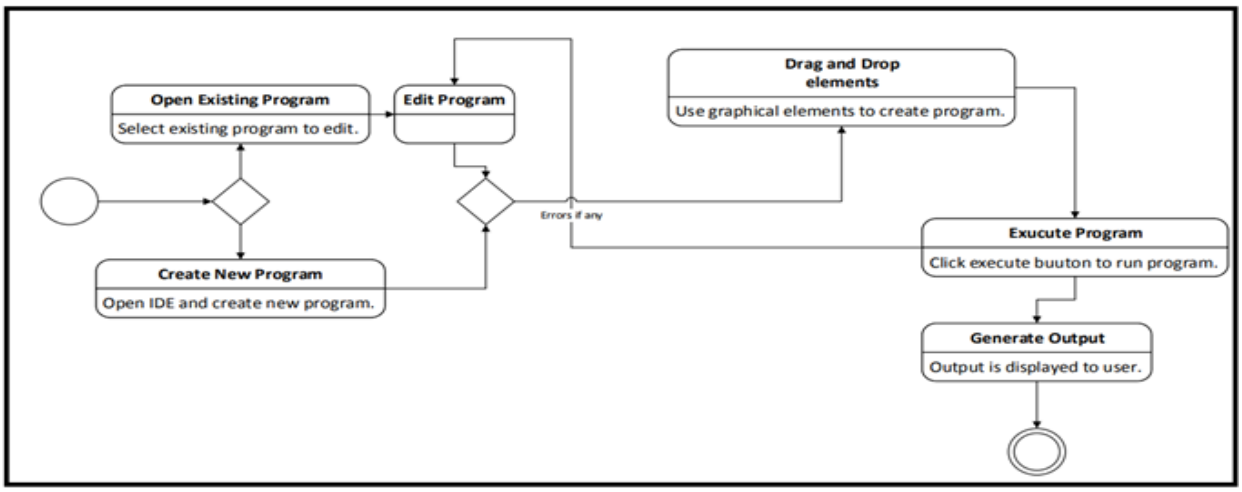
13 Figure 4 - 12 Delete Program Sequence

4.2.2.4 UML State Machine Diagrams

Different Scenarios and their corresponding events are discussed in this section with the help of state machine diagrams.

4.2.2.4.1 Execute Program

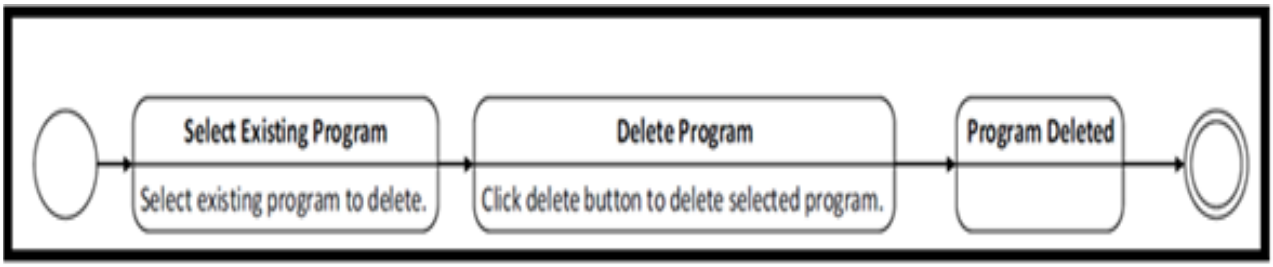
Behavior (discrete) of the parts of the system is represented by a behavior diagram using finite state transitions. Protocol usage of a part of a system can also be shown by this diagram.



14 Figure 4 - 13 Protocol Usage

4.2.2.4.2 Delete Program

In order to delete program, user will select the required program and will then click the delete button. Hence the selected program will be deleted.

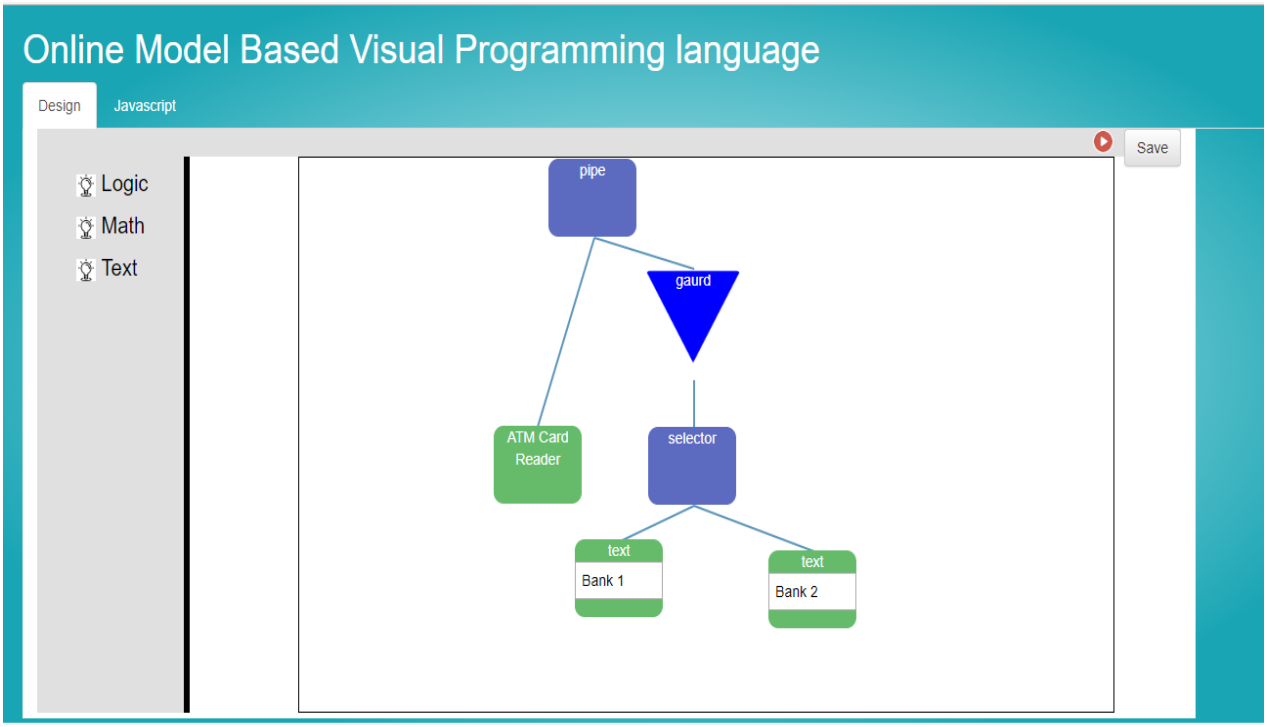


15 Figure 4 - 14 Protocol

4.2.2.5 UI Design

OMBVPL is an online application and intended to be used by the users from diverse background knowledge. This requires that the interface of OMBVPL should have an easy learning curve for the user. Most of the important features should be visible to the user and no functionality should be hidden.

Please note that the interface provided is just for demonstration purposes. Actual interface may be different.



16 Figure 4 - 15 OMBVPL Interface

4.3 Detailed Description of Components

Tool Bar

Identification	Toolbar
Type	Component
Purpose	User interface to access different features of the IDE
Function	Displays a bar of different buttons and each button performs its own function.
Subordinates	An array will hold different buttons, one for each option of the Tool Bar.
Dependencies	Calls instances of the button class.
Interfaces	Performs an action when any button is clicked.
	JavaScript code that runs when the system is started
Processing	When the IDE will start it will appear on top of the IDE web browser.
	Array of button classes that will be used to hold different buttons in the Tool Bar.
Table 1 – Toolbar description	

17 Table 4 - 1

Tool Box

Identification	Toolbox
Type	Component
Purpose	User interface to access different elements and connectors of the language.
Function	Displays a list of different elements which user will use to create a component and the connectors he would use to define logic
Subordinate	A panel will hold all the elements.
Dependencies	Calls the method of each elements.

18 Table 4 – 2

Work Space

Identification	Workspace
Type	Component
Function	Convert the graphical objects into JavaScript code in backend and then show output.
Subordinate	A panel will hold the graphical objects of all the programs.
Dependencies	Uses the model to convert the graphical elements into executable code in the backend.
Interfaces	Connects different graphical elements to create a logical program.
Resources	A panel on which user will create a program from graphical elements.
Processing	When the IDE starts it appears in the center of the IDE when the interface () method is called.
Data	A panel in work space will hold the graphical objects of all the programs.

19 Table 4 - 3

Components and Connectors

Identification	Output
Type	Class
Purpose	Creation of components and functionality is added through the exogenous connectors (guard, loop, sequencer, pipe, selector etc.)
Function	To perform different operations on the components through separation of control through the connectors.
Subordinate	A panel will hold all the text fields and drop-down menus.
Dependencies	Atomic components are the basic building block of the language. These units along with the hierarchy given by the connectors creates all the programs and all the sub components are inter connected through the exogenous connectors.
Interfaces	Uses user input or input from other parts of program and performs a certain action on that input data and gives the corresponding output.
Resources	An input from the user or any other part of the program is required by any component to perform its action.
Processing	Each connector is called by its specific call method when the user drags and drops it in the work space.
Data	Each component will keep the input in specific variables and pass it to other variables in other connected components.

20 Table 4 -4

Output

Identification	Output
Type	Class
Purpose	To display the output of the user program
Function	Show the user with the output of the created program.
Subordinate	The output will be displayed in the output dialog box.
Dependencies	Depends on the program created.
Interfaces	Gets the input from the graphical objects of the program and gives the corresponding output.
Resources	A dialog box will be used to create output dialog box.

21 Table 4 - 5

4.4 Reusability and Relationships to Other Products

Since our IDE is made in accordance with the X-Man model with is already defined but its implementation has not yet been done and our model in many ways does this model's implementation. Our Online X-Man driven visual programming language is a new language and no enhancement is being done in any already existing language. Therefore, no explicit component is being reused. Although the components, once defined can be reused in the program compilation. However, the strategic aspects of future reusability of this model are supposed to be considered. As our language is domain specific, it can be extended and further developed whenever required.

4.5 Design decisions and tradeoffs

We are creating an online visual programming language by using the X-Man model principles due to which a user can only create programs by simply dragging and without getting into language syntax complexities.

X-man follows the component-based software which in turn is the key to short time-to-market, high productivity and cost saving through the components reuse. X-MAN is a framework for compositional software design. Its features include:

- Strong separation of components, that is, components are strictly separated with respect to data and side-effects of execution at the functional level that coordination control among a set of components is entirely governed by composition connectors.
- A simple user interface has been incorporated, so as to extend the usability of the language to all the novice users.
- Thus "Online Model Based Visual Programming" provides a set of predefined components which users can use to develop some basic understanding of programming concepts by using drag and drop.
- In this way a program can be developed using only the atomic components and the connectors without writing any line of code.

4.6 Pseudo Code for Components

New Program

Begin

Open IDE

Select new Program

Drag and drop graphical objects

End

Execute the Program

If Program is correct

Display the Output

Else

Display the error

Go back to Drag/drop phase

Delete Program

Begin

Open IDE

Select existing Program

Delete program

End

Chapter 5: Project Analysis and Evaluation

5.1 Introduction

The purpose of this document is to outline the test strategy and overall testing approach that has been used for the OMBVPL Project. This includes test methodologies, traceability, and resources required, and estimated schedule.

This plan will include testing of the modules created for the visual programming language. These include the components which are already defined in the library: Math (ADD, SUBTRACT, MULTIPLY, DIVIDE), Text (gets the string as input for if/else statements).

It will also test the functionality of the connectors. These include:

- Pipe
- Selector
- Sequencer
- Guard
- Loop

The type of testing being used is incremental integration testing where we have tested math functionalities continuously as they are added both individually and then together.

Testplan Identifier

This test plan aims for Online Model Based Visual Programming Language Project. It is named as “OMBVPL Functionality Testing”.

Test Items

Math library elements are being tested which are ADD, SUBTRACT, MULTIPLY, DIVIDE.

Text component is being tested whether it are correctly displayed by the code generator.

Also, the connectors are being tested which include pipe, selector, sequencer, loop, guard.

Hierarchy of the functions being called by the components in the model will be tested.

Error displaying will also be tested.

The transmittal media used for OMBVPL is through the web browser. Since the project is based on online model based visual programming.

The code has been written in HTML, CSS and JavaScript and the document has been made in accordance with the libraries and classed defined in the design specification document. The user installation guide is given below:

As per the bug report, there were around twenty bugs reported but most of them were fixed. One of the bugs included the selection of only one entry for the math function which is not possibly correct; this bug was fixed by adding a conditional statement in the code. On the other hand, another bug was reported where the control connectors which actually should not be connected to each other in the hierarchy were being connected through a line. This bug is yet to be fixed.

The user interface is not to be tested specifically since the aim of the OMBVPL is to provide separation of concern and control through the implementation of the connectors and components defined by the user.

Features tobe tested

The workspace is to be tested which consists of two categories as defined below:

Drag and drop operations of the elements are being tested which connects different graphical elements to create a logical program. This is done in the workspace as defined in title 3.3 in the SDS document. Here the graphical objects are converted into JavaScript code in the backend and the output is then displayed.

The code generated that is displayed in the workspace dropdown is also tested.

Connections made via connectors are tested. Each connector is called by its specific call method when the user drags and drops it in the workspace. All the design specifications that concern the connections is defined in title 3.4 of SDS document version 1.1.

Also, mathematical operations being performed are checked that whether they are operating correctly or not.

Loops, selection, and control functionalities provided via pipe and sequencer are tested.

The outputs of the code will be tested. The output is integrated in two categories as mentioned in section 3.5 of SDS document:

1. Result display
2. Error display

Features not tobe tested

The user interface is not to be tested specifically since the aim of the OMBVPL is to provide separation of concern and control through the implementation of the connectors and components defined by the user.

Approach

General Test Strategy**:** Component testing will be performed on the components as they are developed. Each component will be individually tested by using several test cases and then tested as a whole system in order to find out any inconsistencies in the system.

Component testing**:** Tests will be conducted to verify the implementation of the design for one module. The purpose of component testing is to ensure that the program logic is complete and correct and ensuring that the component works as designed.

Integration Testing**:**As the components will be developed from the bottom-up, the test strategy will also align to the order of development of components. After the unit testing phase modules are integrated incrementally and tested to ensure smooth interface and interaction between modules. In this approach, every module is combined incrementally, i.e., one by one till all modules or components are added logically to make the required application, instead of integrating the whole system at once and then performing testing on the end-product. Afterwards, integrated modules are tested as a group to ensure successful integration and data flow between modules.

Browser Compatibility Testing**:** Browser Compatibility Testing is performed for web applications and it ensures that the software can run with the combination of different browser and operating system.

Black Box Testing**:** Internal system design is not considered in this type of testing. Tests are based on the requirements and functionality.

Itempass/failcriteria

The criteria are as follows

- The pre-conditions are met
- Inputs are carried out as specified
- Test case will pass if it produces the desired output for a specified input
- Test will fail otherwise

Suspension criteriaandresumption requirements

If the tests are not giving appropriate results according to the expected outputs then further testing is stopped. The criteria for pausing of testing are given below.

- If other than numeric values given to the Entry element.
- If the connectors aren't used in an appropriate way.
- If operators aren't working properly.
- If an operator is directly connected to the other operator.
- If a connector is directly connected to another connector.

If the program does not reconstruct after the changes have been saved, the program may be constructed again and saved again then the testing may resume.

Test deliverables

5.1.8.1 Testing tasks

- Develop Test Cases.
- Execute tests based on the test cases developed.
- Report defects during tests if any.
- Manage the changes made after testing.

5.1.8.2 Test cases

Following are the test cases that will be delivered as per test plan.

5.1.8.2.1 Input Data Testing for math element

Test Case Name	**Input Data Testing for math element**
Test case number	1
Description	In this test case numeric values entered in math elements are tested.
Testing technique used	Black box testing
Preconditions	There are no values inside the entry entity
Input values	Any value like b,5 etc.
Valid inputs	Valid inputs are of type integer like 2,3,4 etc.
Steps	1. Connect connector with math operator 2. Press play button to run program 3. Input an integer 4. Input alphabet
Expected outputs	No output will be obtained because all values are not numeric
Actual output	No result obtained
Status	Passed

22 Table 5 – 1

5.1.8.2.2 Input Data Testing for text element

Test Case Name	**Input Data Testing for text element**
Test case number	2
Description	In this test case string value entered in text element is tested
Testing technique used	Black box testing
Preconditions	No value is entered
Input values	Text string e.g. “Hello world”
Valid inputs	Any kind of text which includes symbols, integer and alphabets

Steps	1. Connect selector with text 2. Input conditions 3. Input string like “Hello world” in text box 4. Run the program
Expected outputs	The text entered (Hello World) will be displayed on the screen in alert box
Actual output	“Hello World”
Status	Passed

23 Table 5 - 2

5.1.8.2.3 Connector’s functionality testing

Test Case Name	**Connector’s functionality testing**
Test case number	3
Description	In this test case functionality of connectors in general will be tested
Testing technique used	Black box testing
Preconditions	The desired connector should not be already dragged in workspace
Input values	Attach only one desired function in connector
Valid inputs	Math function and text function. There will be more than one functions attached to the sequencer.
Steps	1. Drag sequencer and subtract function 2. Attach sequencer with subtract function according to the required hierarchy. 3. Run program 4. Input values for the function
Expected outputs	Error
Actual output	Error: attach more than one items to sequencer
Status	Passed

24 Table 5 - 3

5.1.8.2.4 Sequencer Functionality Testing

Test Case Name	**Sequencer’s functionality testing**
Test case number	4
Description	In this test case functionality of sequencer will be tested
Testing technique used	Component testing
Preconditions	Sequencer should not be connected to the components
Input values	Attach more than one components e.g. add, subtract, multiply, divide etc. to the sequencer
Valid inputs	More than one components attached
Steps	1. Drag and drop sequencer, add and subtract

	2. Attach connector with components 3. Run program 4. It takes the data from add and executes it for the output 5. Then takes data from component B and performs its services 6. Then pass the held results to the control originator
Expected outputs	It takes the data from add and executes it for the output Then takes data from component B and performs its services Then pass the held results to the control originator
Actual output	Functions will be called in sequence which can be seen in code generated by code generator
Status	passed

25 Table 5 - 4

5.1.8.2.5 Pipe Functionality Testing

Test Case Name	**Pipe Functionality Testing**
Test case number	5
Description	In this test case functionality of pipe is tested
Testing technique used	Component testing
Preconditions	Pipe should not be connected to the components
Input values	Attach more than one connectors along with their functions.
Valid inputs	More than one different connectors with their functions are attached
Steps	1. Drag pipe, sequencer, selector, add, subtract and text. 2. Attach selector with two texts. 3. Enter text in both text box 4. Enter conditions for which text will be displayed. 5. Attach add and subtract with sequencer 6. Attach sequencer and selector with pipe 7. Run program
Expected outputs	It takes the output of sequencer and gives it as an input to selector in sequence
Actual output	Data will be passed from sequencer to selector as seen in the code generated by code generator.
Status	passed

26 Table 5 – 5

5.1.8.2.6 Selector's Functionality Testing

Test Case Name	Selector Functionality Testing
Test case number	6
Description	In this test case functionality of selector is tested
Testing technique used	Component testing
Preconditions	Selector should not be connected to the components
Input values	Functions from the library such as add, subtract or text will be attached
Valid inputs	Text function
Steps	1. Drag pipe, sequencer, selector, add, subtract and text. 2. Enter "less than 20" in first text box 3. Enter "greater than 20" in second text box 4. Attach selector with two texts. 5. Enter condition "<20" for first textbox and ">20" in second textbox. 6. Attach add and subtract with sequencer. 7. Attach sequencer and selector with pipe. 8. Run program. 9. Input 5 to each prompt
Expected outputs	Less than 20
Actual output	Less than 20
Status	passed

27 Table 5 – 6

5.1.8.2.7 Save functionality Testing

Test Case Name	Save Functionality Testing
Test case number	7
Description	Program once developed can be saved
Testing technique used	Integration testing
Preconditions	Save button shall not be pressed
Input values	Program is developed
Valid inputs	Program is developed according to hierarchy
Steps	1.Develop a program by dragging one selector and any two math elements 2. Press save button 3. Select folder in which you want to save program
Expected outputs	Program will be saved in specified folder
Actual output	Program saved in specified folder
Status	Passed

28 Table 5 – 7

5.1.8.2.8 Load functionality Testing

Test Case Name	Load Functionality Testing
Test case number	8
Description	User can load program once he save it
Testing technique used	Incremental integration testing
Preconditions	Ombvpl is opened in browser
Input values	Select file to open
Valid inputs	Select file having .html extension
Steps	1. Click open program button in menu 2. Select folder and then file to open 3. Load file in workspace
Expected outputs	Program will be opened successfully
Actual output	Program is opened successfully
Status	

29 Table 5 - 8

5.1.8.2.9 Browser Compatibility Testing

Test Case Name	Browser Compatibility Testing
Test case number	9
Description	In this testing program will be testing in browser and checked whether it is loaded correctly
Testing technique used	Browser Compatibility Testing
Preconditions	OMBVPL is not loaded in browser
Input values	URL of OMBVPL
Valid inputs	URL of visual programming language
Steps	1. Open browser 2. Input URL of OMBVPL 3. Check design of tool
Expected outputs	The OMBVPL is successfully executed in browser.
Actual output	OMBVPL is executed successfully in browser
Status	passed

30 Table 5 – 9

Testing tasks

Testing Tasks	Assigned to
Component Testing	Developer
Incremental Integration Testing	Developer
Acceptance Testing	Customer/Project Manager
Reports Verification	Customer/Project Manager

UI Requirements Verification	Customer
Defect Reporting	Customer/Project Manager/Developer

31 Table 5 - 10

Environmental needs

- Hardware: No hardware required
- Software: HTML, CSS, JavaScript.

Responsibilities

Roles	Responsibilities
Test Manager	Generates test plan & test resources Reviews the requirement analysis, system architecture design & object design Generates Test Cases Periodically updates the Program Director on the progress of test execution
Test Leads	Create detailed test specifications Manage day-to-day progress of subcomponents and compile and report the metrics to the test manager Ensures testers makes adequate progress & follow strategy defined by test manager
Component Testers	Responsible for test execution on daily basis Leads the effort during most of the integration test cycle and hand off the testing to the System Testers during the last states of incremental integration testing.
System Testers	Responsible for functional testing Responsible for performance testing.

32 Table 5 - 11

Staffing and training needs

Basics knowledge of testing strategies and techniques is needed for the testing of the project.

Techniques such as Black Box testing, integration testing should be known to developers.

All the developers will be testing each other's work and will be actively participating in the development and testing of the project simultaneously.

Schedule

Test Phase	Time	Owner
Test plan creation	1 week	Test manager
Test specification creation	2 weeks	Test leads
Test Specification Team Review	1 week	Project team
Component Testing	4 weeks	Component testers
Integration testing	4 weeks	System and component tester

33 Table 5 – 12

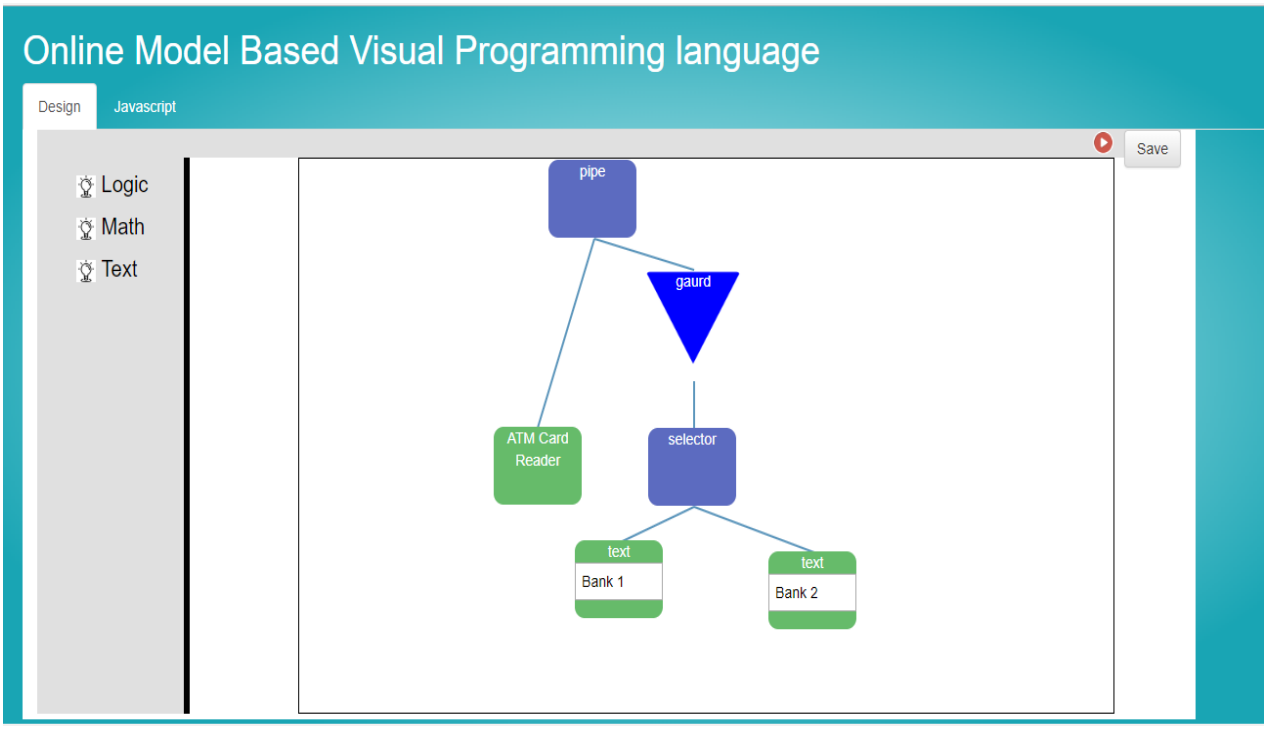
Risks and contingencies

- Internet connection may not work correctly.
- Program may not load correctly due to slow internet connection.
- May not get the expected output.

Approvals

It is approved by all the team members of OMBVPL Project

Working example of the online IDE can be taken from the bank ATM card reading system. It can be demonstrated in the example below:



It realizes a simple atm example consisting of three basic components which are Card Reader, Bank A and Bank B. Bank system has one atm to serve two banks. Selector connector routes the request to one of the component at a time.

Here atm component reads the atm card and get the authentication. As soon as the component reads the pin code entered by the user the control passed on to the pipe passes the value to the guard which authenticates the card and on the basis of authentication the control is passed on to one of the bank selected by the selector condition connector.

The composite for bank system is then adapted by non-terminating infinite loop connector so after serving one client the system is ready to serve another.

Chapter 6: Future work

6.1 Introduction

In this chapter we are going to discuss different possibilities that we can integrate with this ombvpl tool to make this tool more user friendly. This chapter describes different types of future work that we can use to make our tool even more complex.

GUI feature

For now we are prompting user for input via textbox. In future we can add GUI feature by which user can develop beautiful GUI and can offer more user friendly input method. User can also develop beautiful GUI to make different type of games.

Database

In future we can add database feature of any one of the famous database. By using which user can add his data to database and even make login page for other users who are going to use. E.g. we can add firebase feature to our ombvpl tool. Firebase provide many features such as firebase login, storage space, realtime database, cloud messaging and much more.

More component

We can add more components to existing ombvpl tool in future. Components such as math functions like square, square root can be added. We can add event listeners also.

Future work also includes the evaluation of the framework using models that utilize the support for IEEE floating-point arithmetic.

Further research will include the development of test-suite generation algorithms that exploit the strong isolation of the components and the information about the structure of their composition.

Bibliography

Microsoft Visual Programming Language.
(https://msdn.microsoft.com/en-us/library/bb483088.aspx)

Scratch Visual Programming Language (Offline Editor).
(https://en.wikipedia.org/wiki/Scratch_(programming_language))

Model Based Architecture
(https://en.wikipedia.org/wiki/Model-based_architecture)

Visual Programming Language
(https://en.wikipedia.org/wiki/Visual_programming_language)
(http://users.dcc.uchile.cl/~rbaeza/cursos/vp/todo.html)
(http://dictionary.reference.com/browse/visual+programming+language)

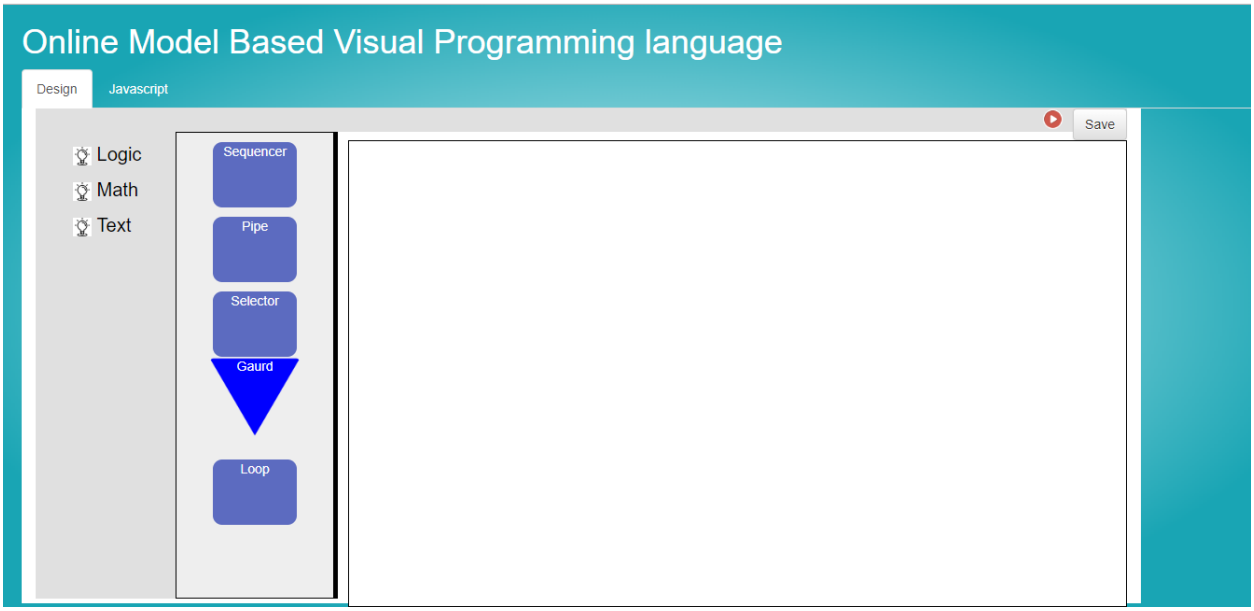
Appendix

Tutorial for OMBVPL

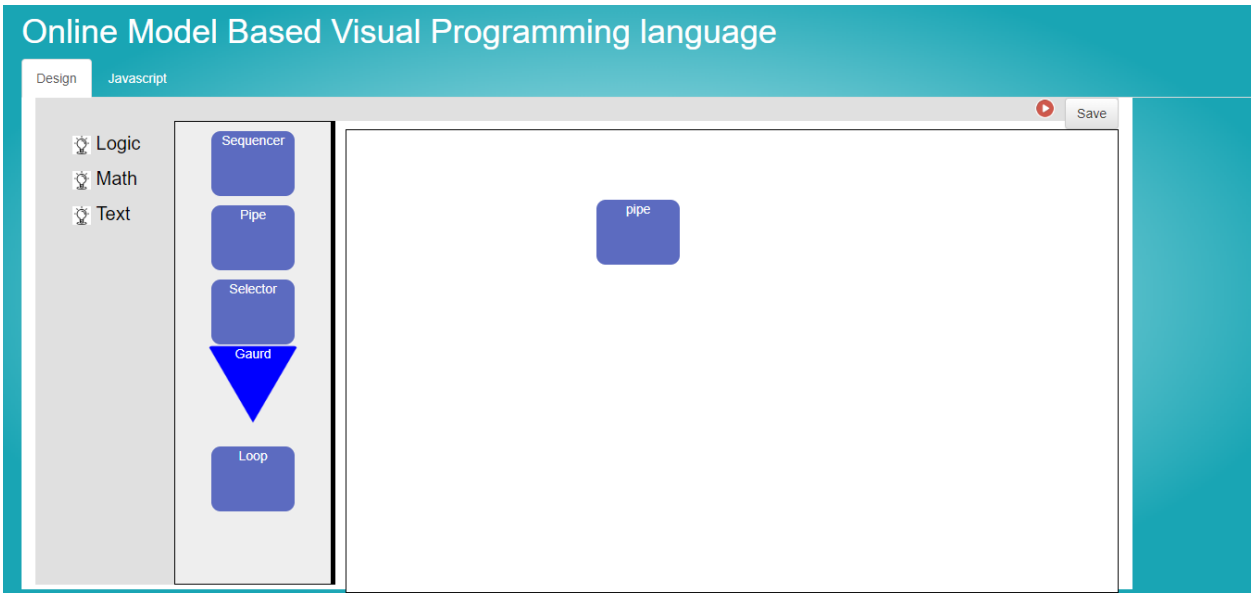
Go to this link: https://usama4745.github.io/

Here you will see workspace. Here we are going to develop an example known as atm machine.

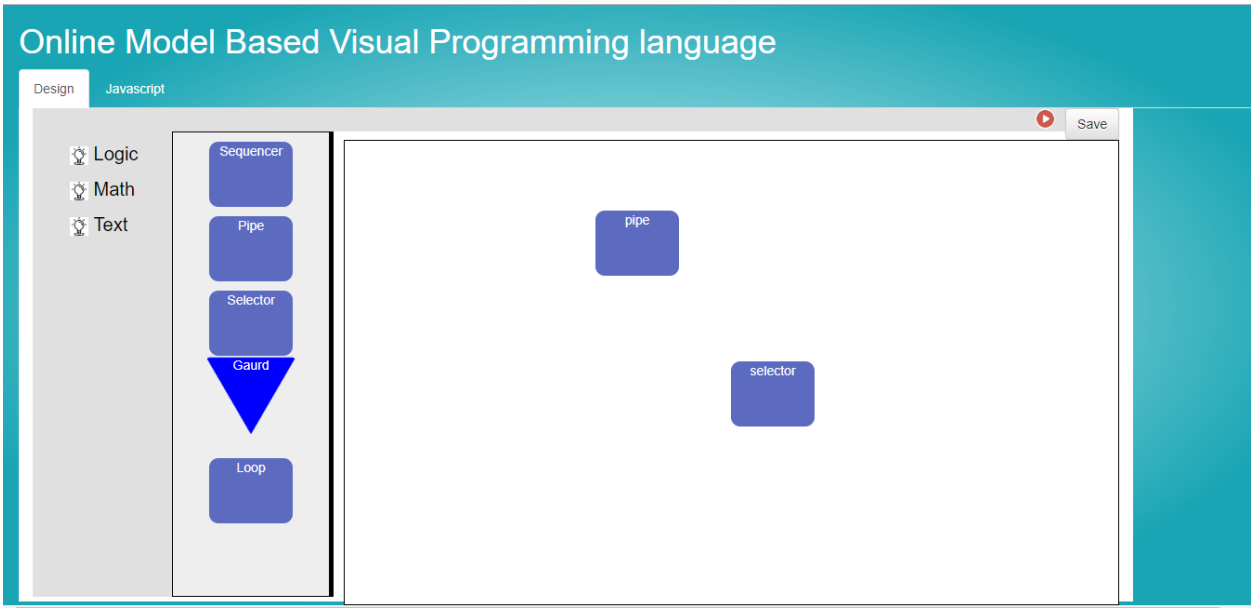
- Click on logic section.



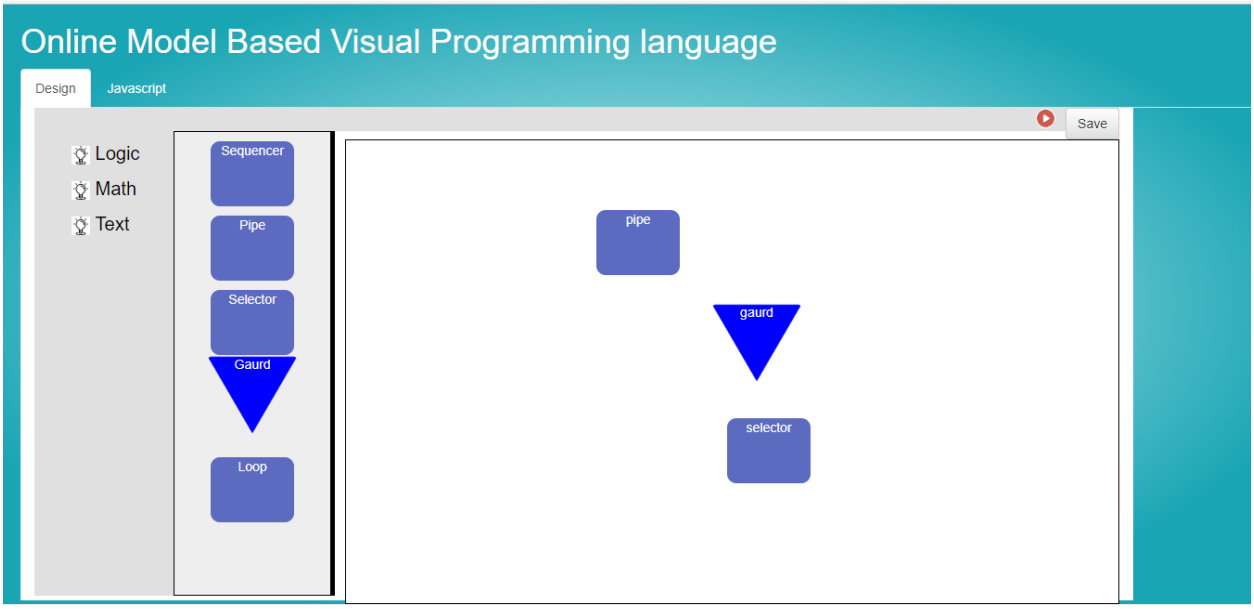
- Click on pipe.



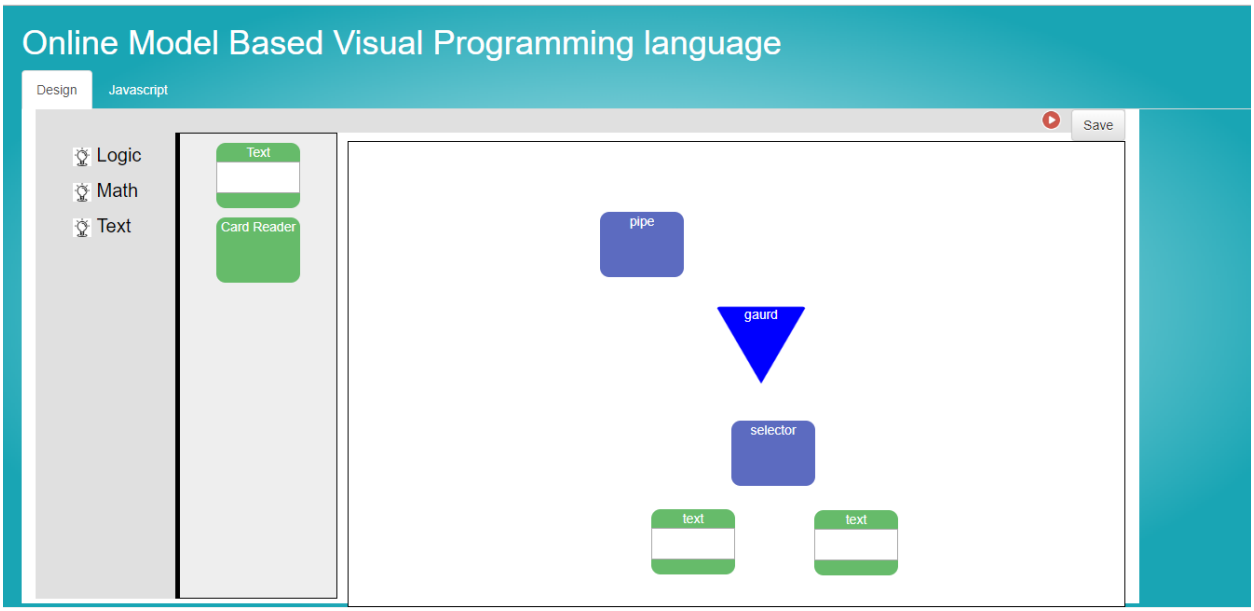
- Click on selector.



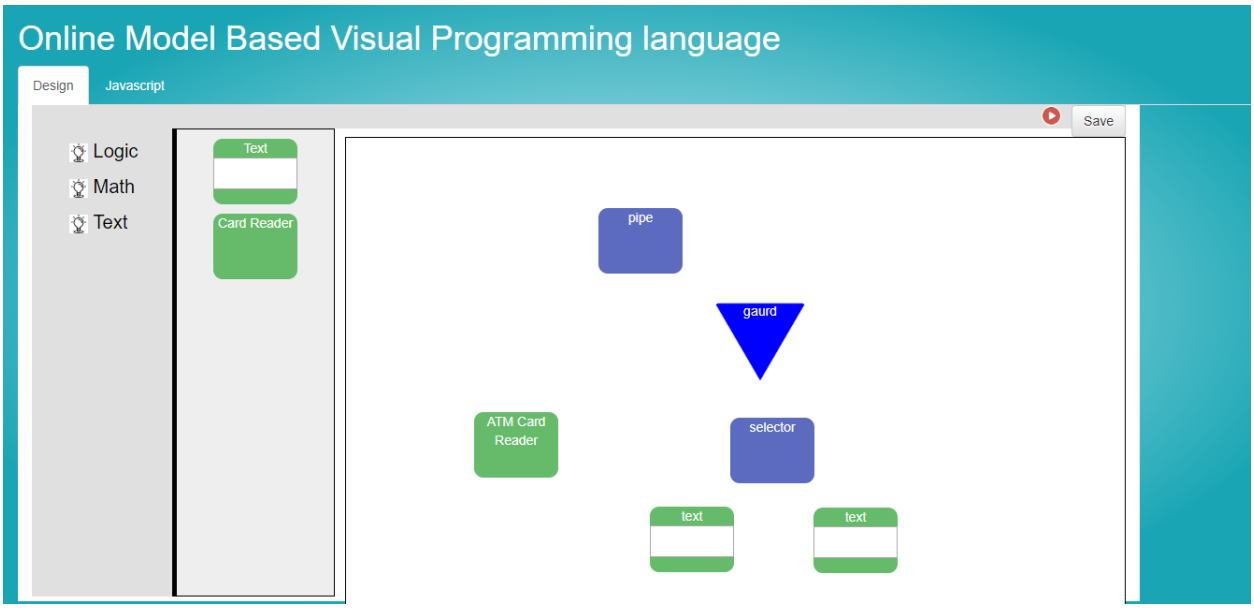
- Click on guard.



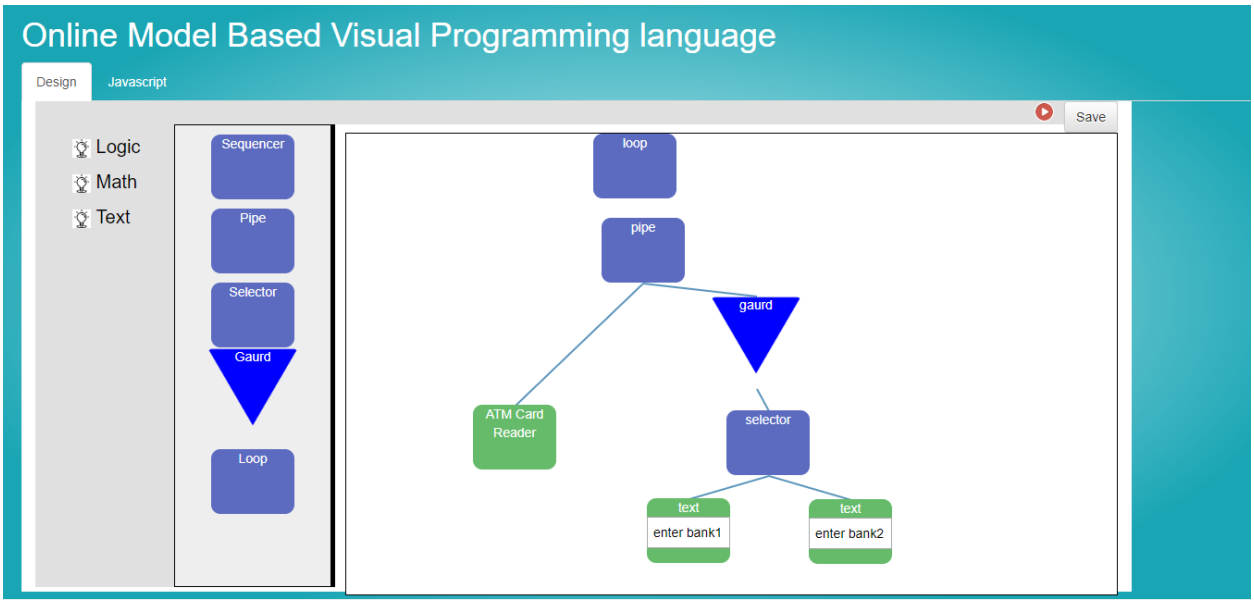
- Now go to text section and click on text two times.



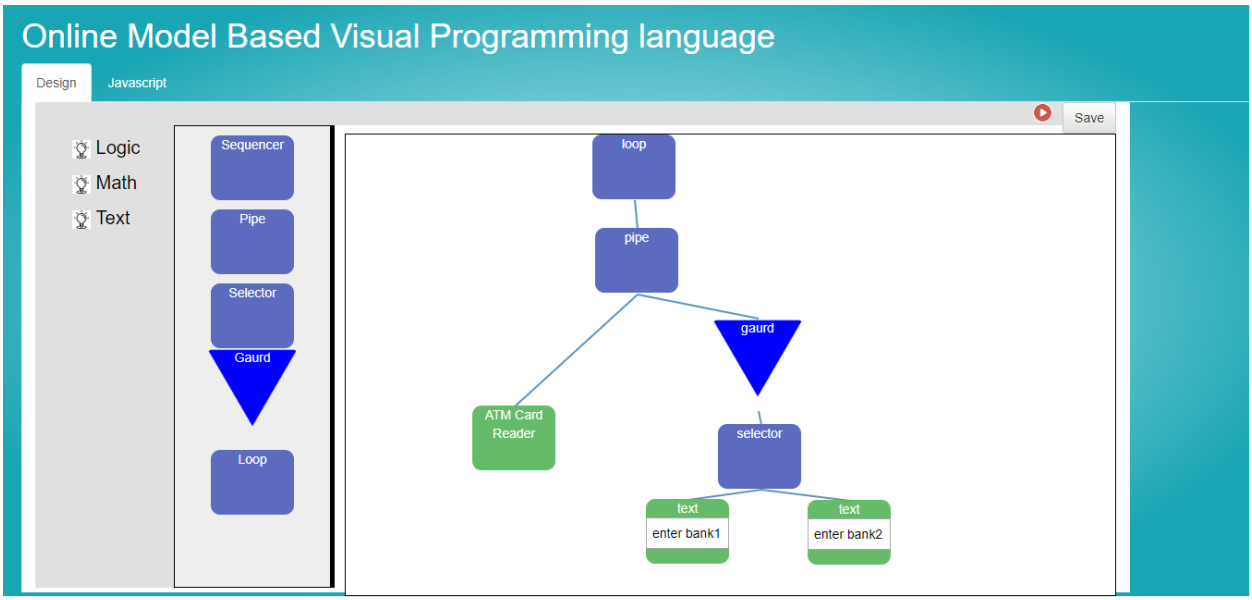
- Now click on card reader.



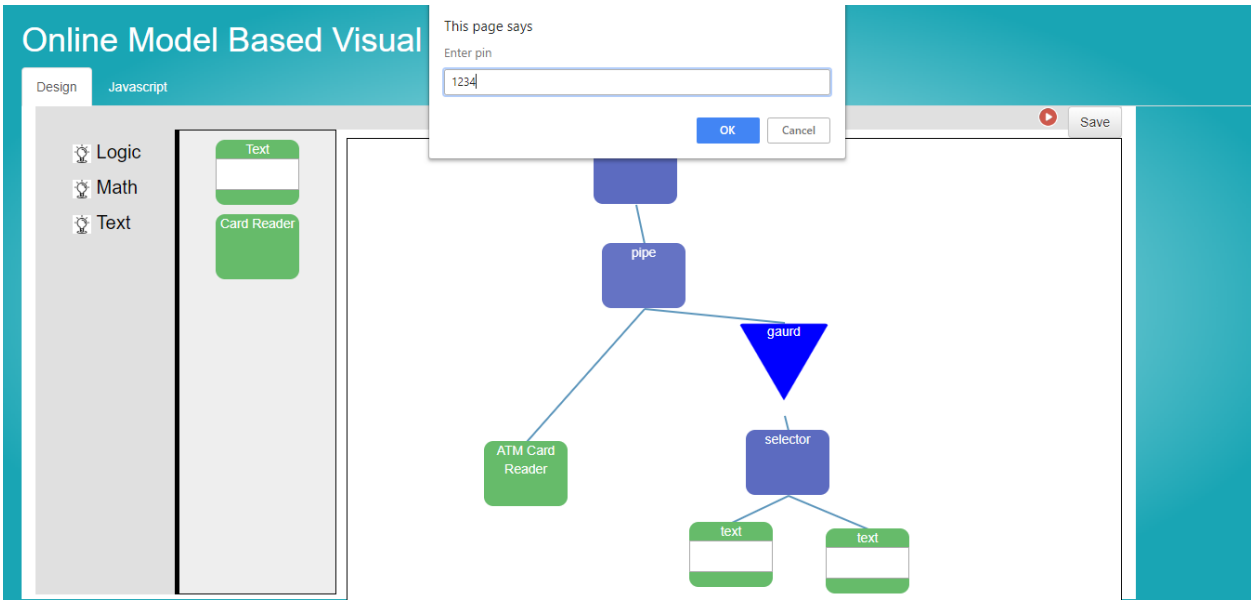
- Now go to logic section again and click on loop and type “enter bank1” in first text box and “enter bank2” in second text box.



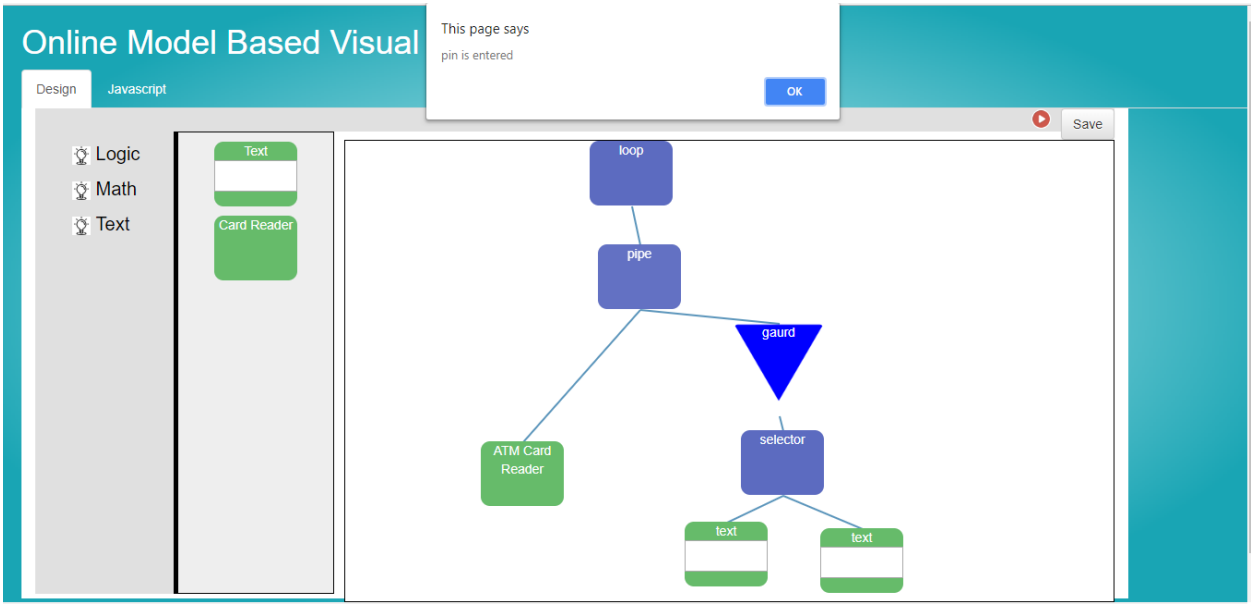
- Now connect loop with pipe.



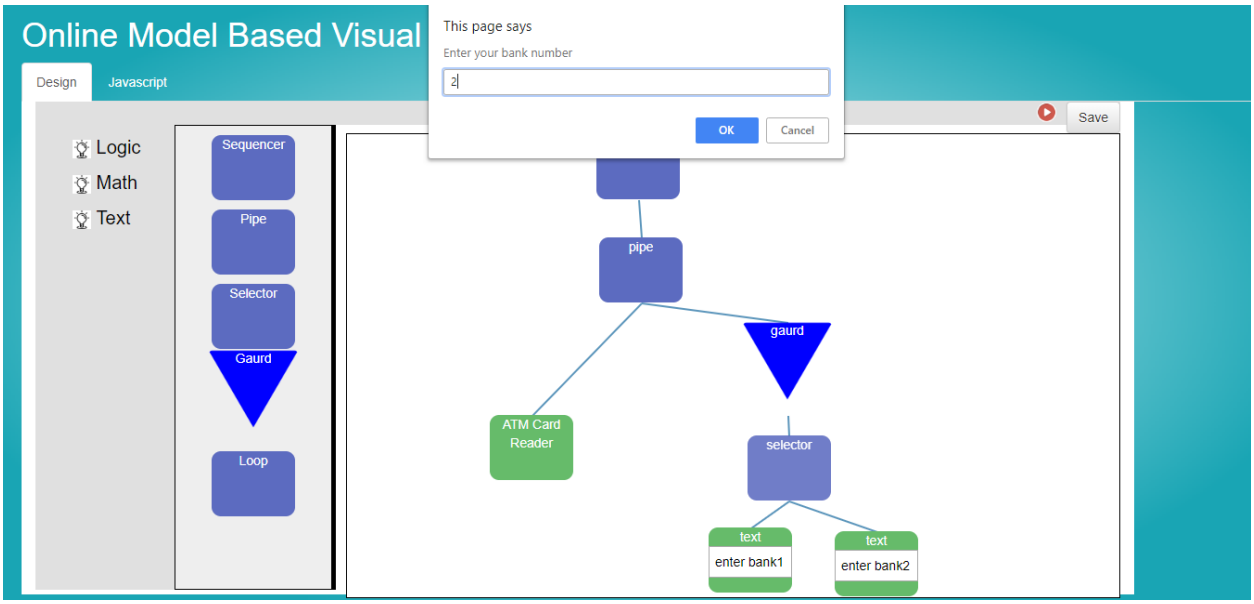
- Now click on red play button so the program will run. Now enter pin code 1234 in dialog box and press enter.



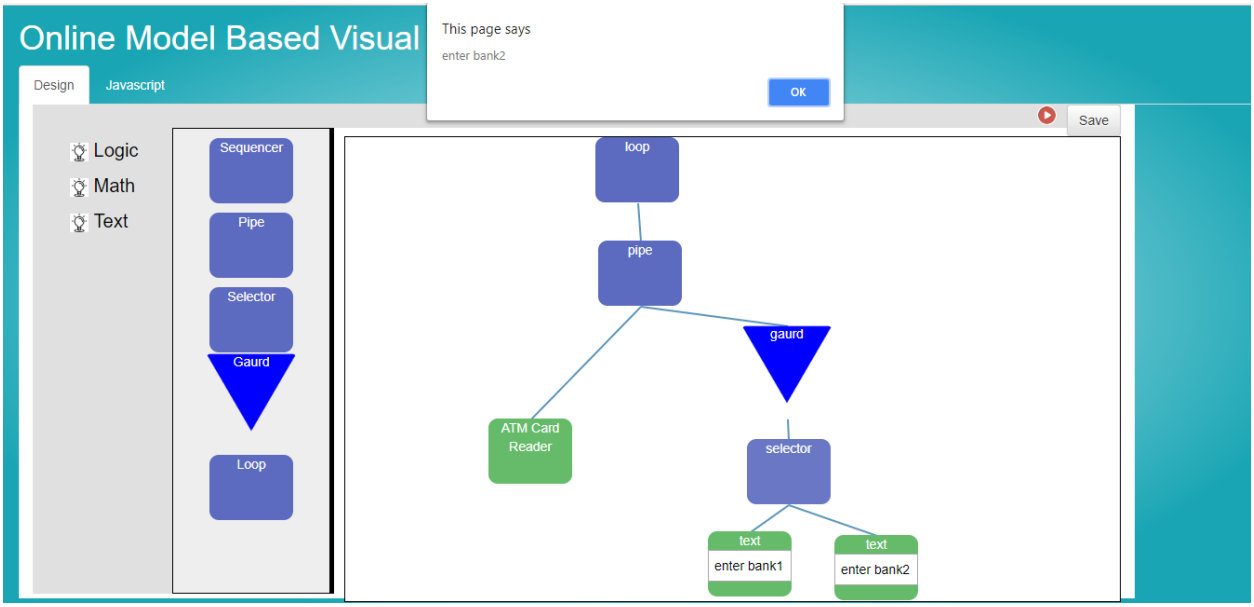
- Message will be shown which confirms that pin is entered correctly.



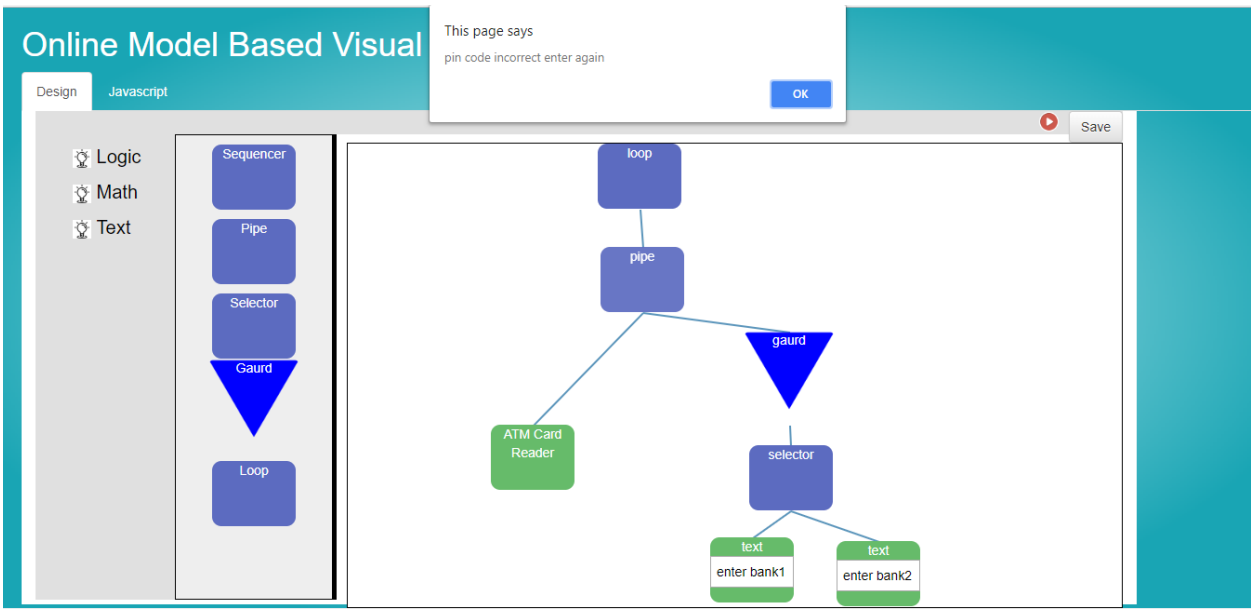
- Now in dialog box type bank number you want to enter. For now we are typing 2.



- Now the message will be shown which confirms that we have entered bank2



- Suppose in previous step we have entered wrong pin that message will be shown and program will run again from start.



This page is left intentionally blank.

