MODEL DRIVEN VISUAL PROGRAMMING LANGUAGE

(MDVPL)

By

NC Aisha Saeed

PC Asna Suhail Zaman

ASC Muhammad Umer

Submitted to the Faculty of Computing Software Engineering

National University of Sciences and Technology, Islamabad in partial fulfillment

For the requirement of a B.E. Degree in Computer Software Engineering

MAY 2016

ABSTRACT

MODEL DRIVEN 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. 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

Certified that work contained in the thesis – Model Driven Visual Programming Language carried out by Aisha Saeed, Asna Suhail Zaman, Muhammad Umer under supervision of Dr. Tauseef Rana for partial fulfilment of Degree of Bachelor of Software Engineering is correct and approved.

Approved by

Dr. Tauseef Rana

CSE

DEPTARTMENT

DATED:__________________ MCS

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

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose unflinching support and unstinting cooperation,

a work of this magnitude would not have been possible

ACKNOWLEDGEMENTS

There is no success without the will of ALLAH Almighty. We are grateful to ALLAH, who has given us guidance, strength and enabled us to accomplish this task. Whatever we have achieved, we owe it to Him, in totality. We are also grateful to our parents and family and well-wishers for their admirable support and their critical reviews. We would like to thank our supervisor Dr. Tauseef Rana, for his continuous guidance and motivation throughout the course of our project. Without their help we would have not been able to accomplish anything.

Chapter 1: Introduction

1.1 Overview

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 driven 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.

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).

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

Chapter 2: Literature Review

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.

1) **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.

2) **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.

Chapter 3: Software Requirement Specification

3.1 Introduction

3.1 .1 Purpose

This document details the software requirements specification for the Model Driven 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 driven 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 Model Driven 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.

3.1.2 Document Conventions

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

3.1.3 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.

3.1.4 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

3.2.1 Product Perspective

Computer programming is a skill which requires a lot of practice and time to master it. Maintaining huge amount of data and performing big calculations can be very hectic and time consuming. By programming in certain language we can develop such software which can help us maintain huge data and perform big calculation in very short time and efficiently.

But what if someone wants to develop such software but he/she do not have any knowledge about any programming language and do not have any skills in programming? How can such a person develop software? To solve this problem we introduce a “VISUAL PROGRAMMING LANGUAGE” which contains set of predefined elements which users can drag and drop into the work space and a complete working program/software can be developed using only these elements without writing any line of code.

3.2.2 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.

3.2.3 User Classes and Characteristics

Following are our targeted users:

- Beginners: The students of school or colleges who have no knowledge of programming.
- Novice: The first year students of universities who have no or very little knowledge about programming.
- Both users and their interaction with the system is shown in Figure A.

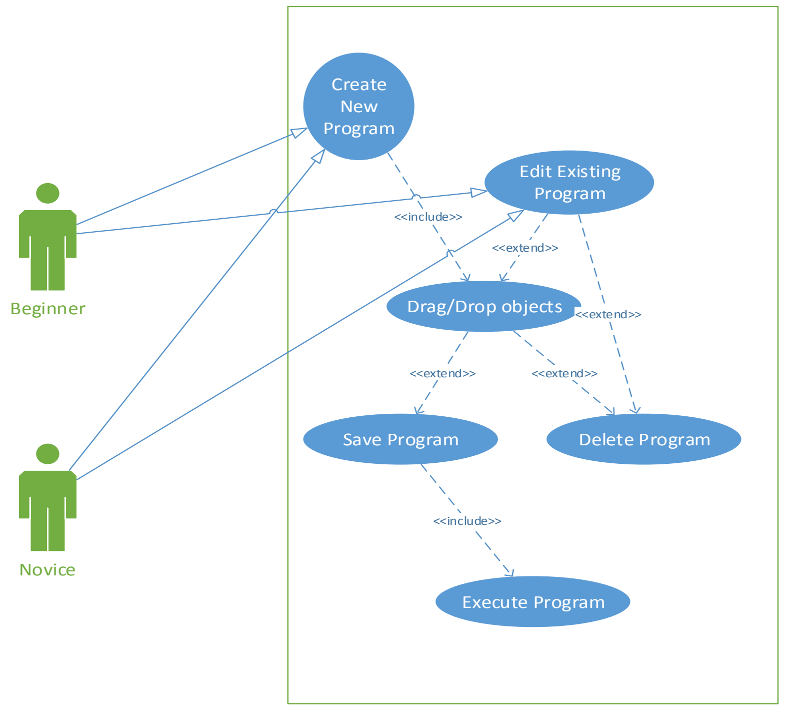


Figure A.

3.2.4 Operating Environment

- Microsoft Windows/Mac OS/Linux
- Eclipse IDE
- Graphical Modeling Framework
- Eclipse Modeling Framework

3.2.5 Design and Implementation Constraints

- The user can only use the predefined graphical elements to create a program.
- The language is using a model driven approach.
- User can create programs by following the tutorials given in the user manual.

3.2.6 User Documentation

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

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

3.2.7 Dependencies

- The system requires Eclipse IDE to run the program.
- The system also requires two Eclipse modeling framework (EMF and GMF) to run the program.

3.3 External Interface Requirements

3.3.1 User Interfaces

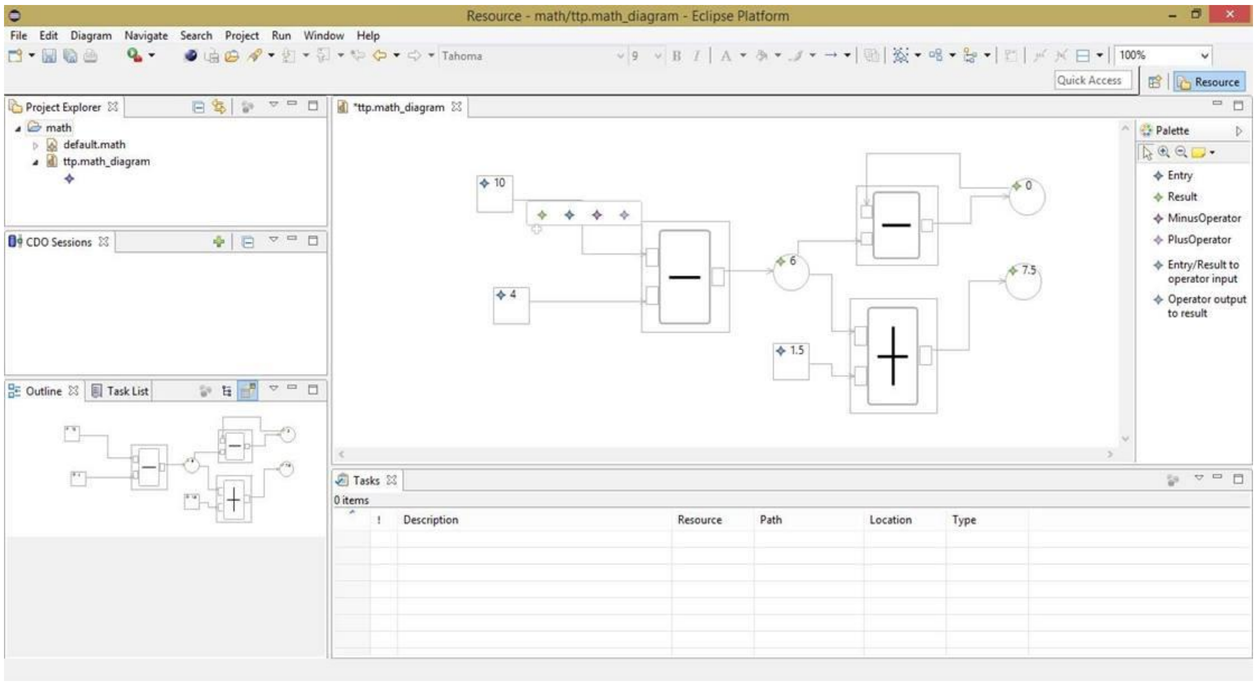


Figure B.

3.3.2 Hardware Interfaces

We do not require any hardware for this project.

3.3.3 Software Interfaces

- This Model Driven Visual Programming Language will be developed in Java and we will be using the Eclipse IDE.
- Microsoft Windows OS/Mac OS/Linux will be required to run this language.

3.3.4 Communication Interfaces

- N/A - No communication interfaces are required.

3.4 System Features

3.4.1 Tool Bar

The tool bar will provide the options to save a current project or open any previous project. The tool bar will also have the button to compile and run the program.

3.4.2 Tool Box

The tool box will contain all the graphical objects/elements which will be used for the programming.

3.4.3 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.

3.4.4 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.

3.4 Non-Functional Requirements

3.5.1 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.

3.5.2 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.

3.5.3 Software Quality Attributes

3.5.3.1 Extensibility and Maintainability

As model driven architecture is inherently extensible therefore the model used in this software can be extended and further developed.

3.5.3.2 Portability

As the programming on back end is in java so with the help of JVM (Java Virtual Machine) the program written on one platform can be ported to other platforms without making any changes to the program.

3.5.3.3 Robustness

The system shall not terminate in case the user tries to create multiple programs because the system does not allow the user to perform two tasks simultaneously.

3.5.3.4 Usability

The new user will be able to create program after following a quick tutorial given in the user manual.

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.

4.1.1 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.

4.1.2 Scope of the Development Project

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 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.

4.1.3 Definitions, acronyms, and abbreviations

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 Driven Architecture:

Model-driven architecture (MDA) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems.

4.1.4 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 introduction of 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 used while excavating a bit more about the motivations behind those decisions.

4.2 System Architecture Description

4.2.1 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:

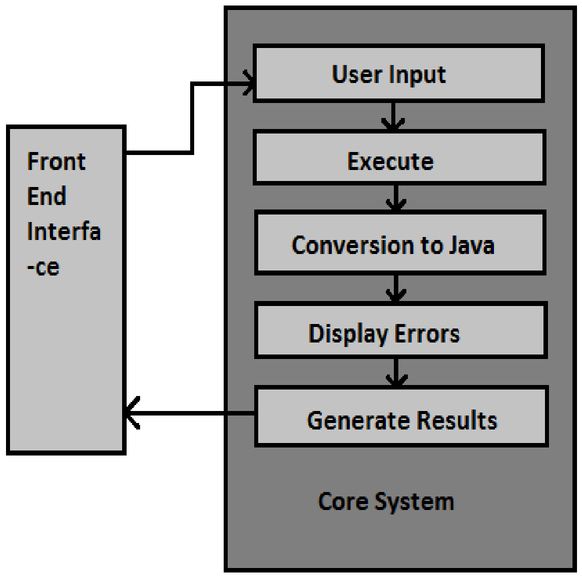


Figure 1 - Overview of the Modules

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 Java 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 MDVPL. 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 MDVPL.

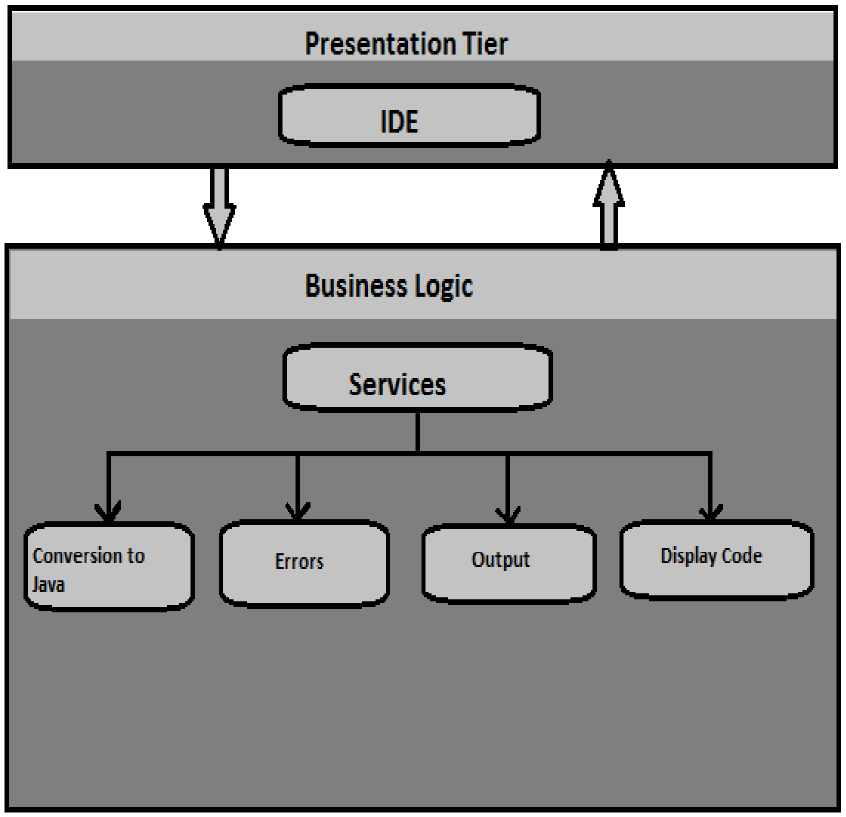


Figure 2 - System Architecture

4.2.1.3 Layers Details

The details of the layers have been discussed below.

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.

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 Java”, “Display Errors”, “Display Output” and “Display Code”, the layer caters for the core functionality of the system.

4.2.2 Structure and Relationships

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

4.2.2.1. 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:

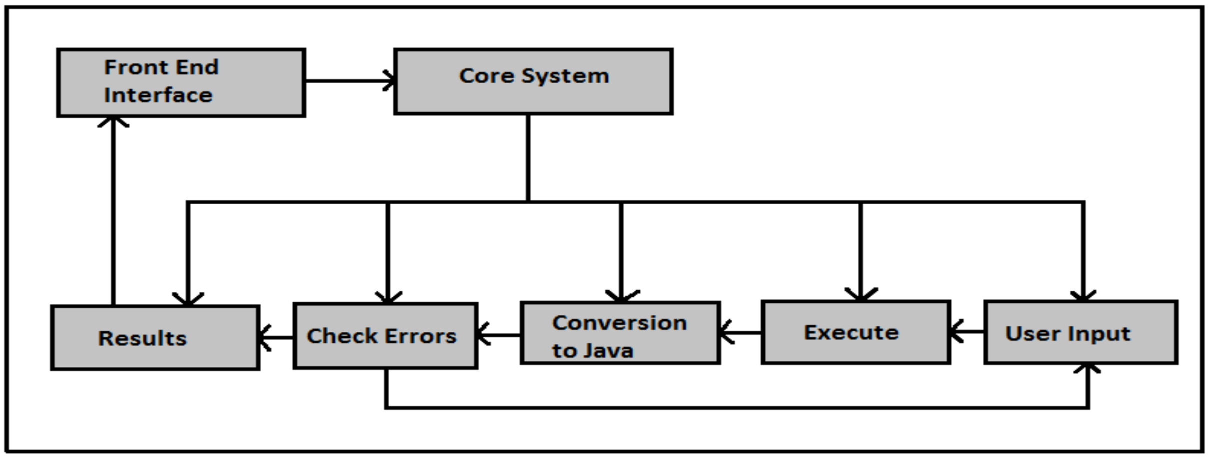


Figure 3 - Overall Structure of the System

4.2.2.1.1 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.

4.2.2.1.2 Core System

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

4.2.2.1.3 User Input

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

4.2.2.1.4 Execute

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

4.2.2.1.5 Java Compiler

The converted Java Code is compiled and executed.

4.2.2.1.6 Check Errors

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

4.2.2.1.7 Results

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

4.2.2.2 Use Case Diagram

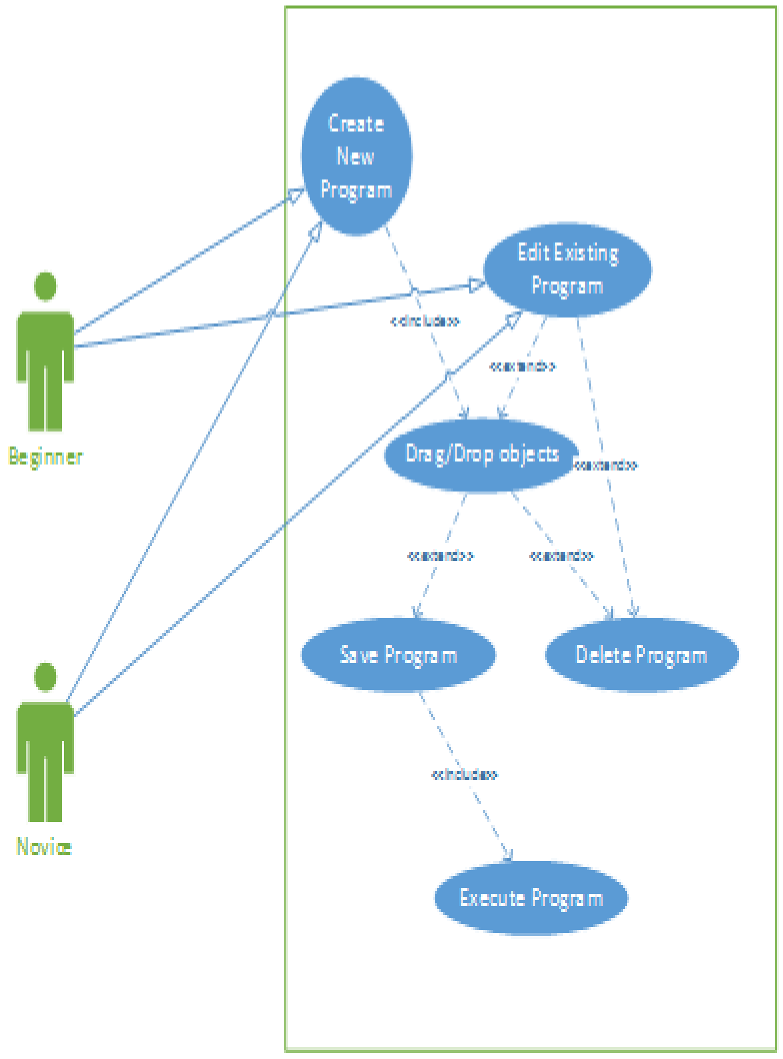


Figure 4 – Use Case Diagram

4.2.2.3 Class Diagram With Description

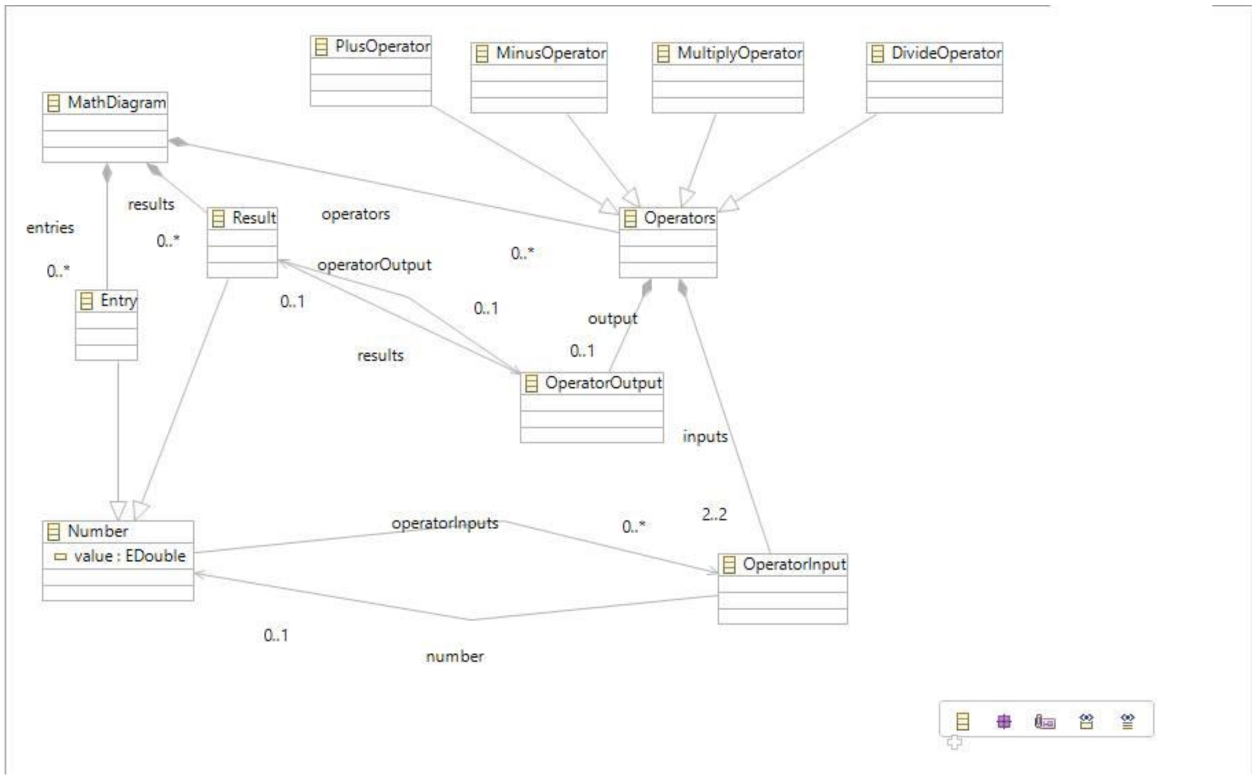


Figure 5 – Class Diagram

4.2.2.3.1 Description of Diagram

4.2.2.3.1.1 Math Diagram

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

4.2.2.3.1.2 Logical Operators

This class provides the functionality of logical operators i.e., AND, OR, NOT, XOR.

4.2.2.3.1.3 Math Operators

This class provides the functionality of math operations i.e., +, -, *, /, %.

4.2.3 User Interface Issues

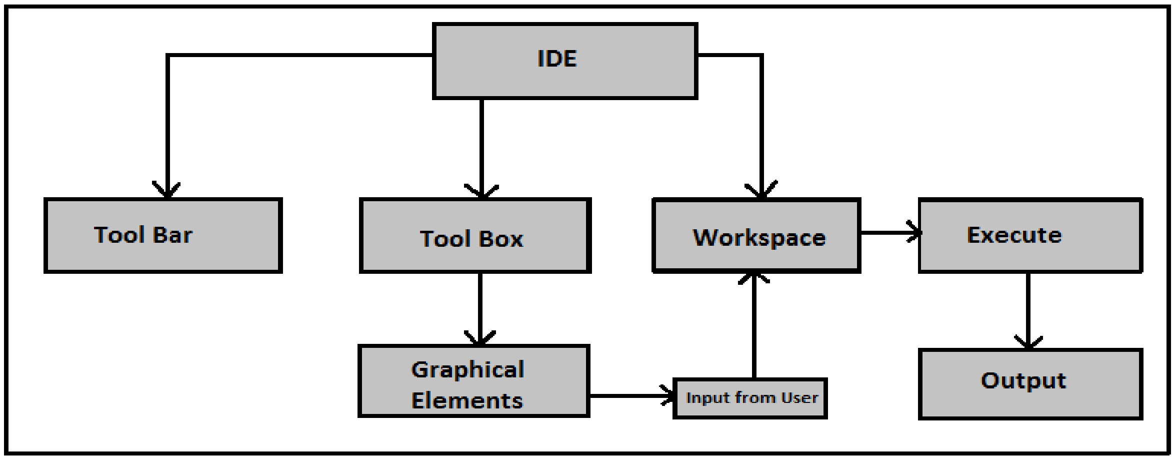


Figure 6 – User Interface Issues

4.2.3.1 Description of the diagram

4.2.3.1.1 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.

4.2.3.1.2 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.

4.2.3.1.3 Tool Box

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

4.2.3.1.4 Graphical Elements

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

4.2.3.1.5 Workspace

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

4.2.3.1.6 Output

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

4.2.3.2 UML Activity Diagram

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

4.2.3.2.1 Execute Program

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

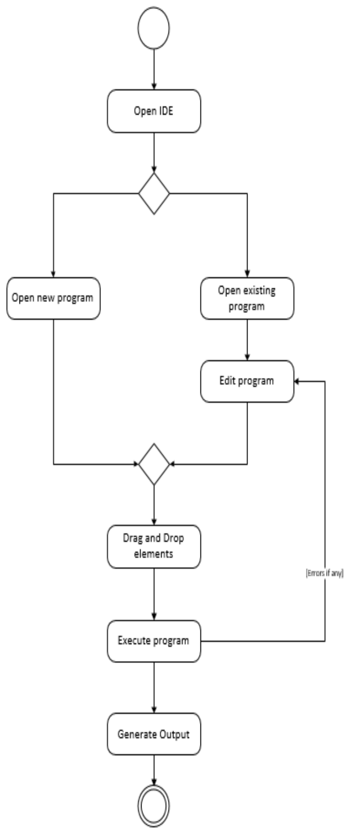


Figure 7 – Execute Program Activity

4.2.3.2.2 Edit Program

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

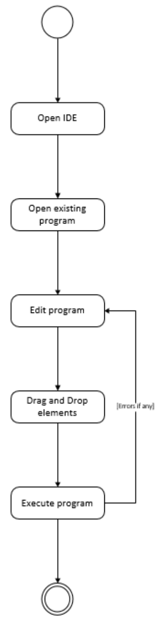


Figure 8 – Edit Program Activity

4.2.3.2.3 Delete Program

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

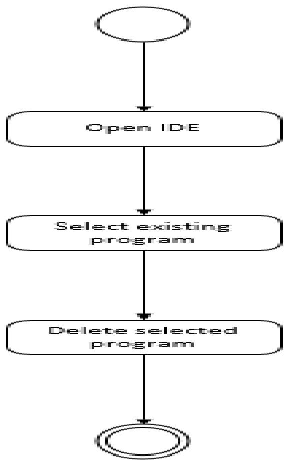


Figure 9 – Delete Program Activity

4.2.3.3 UML Sequence Diagrams

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

4.2.3.3.1 Execute Program

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

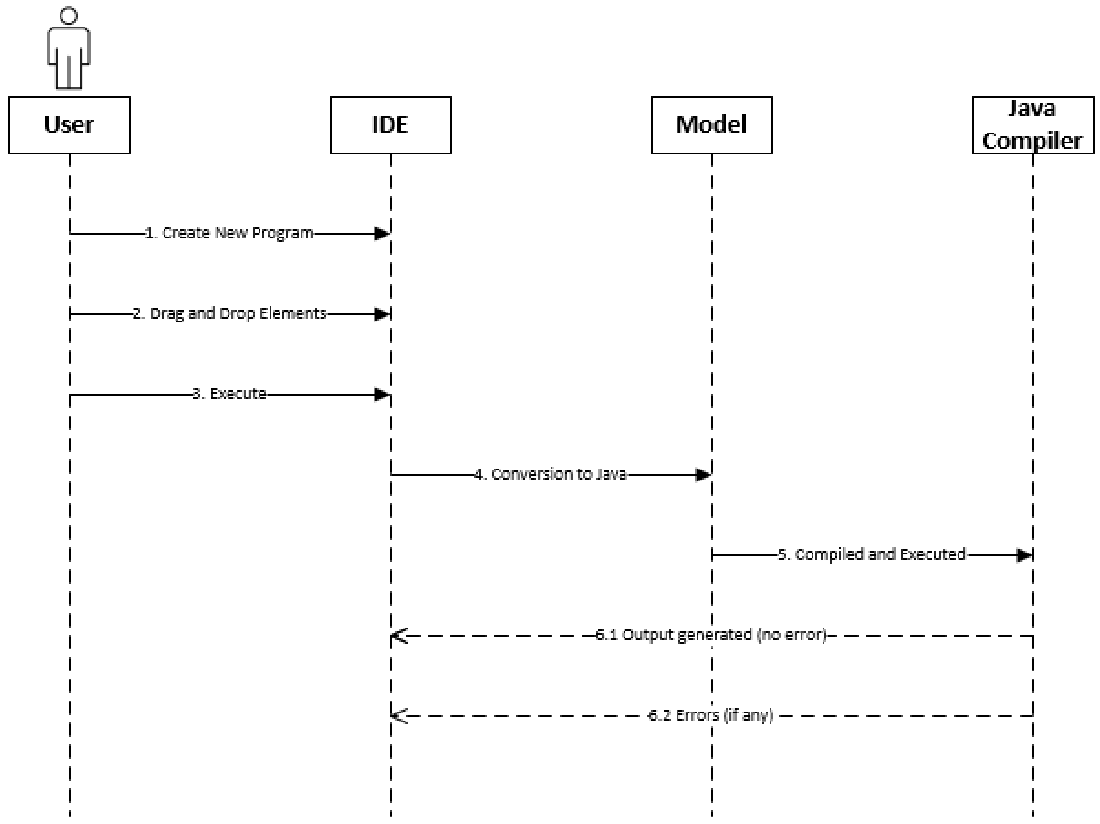


Figure 10 – Execute Program Sequence

4.2.3.3.2 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.

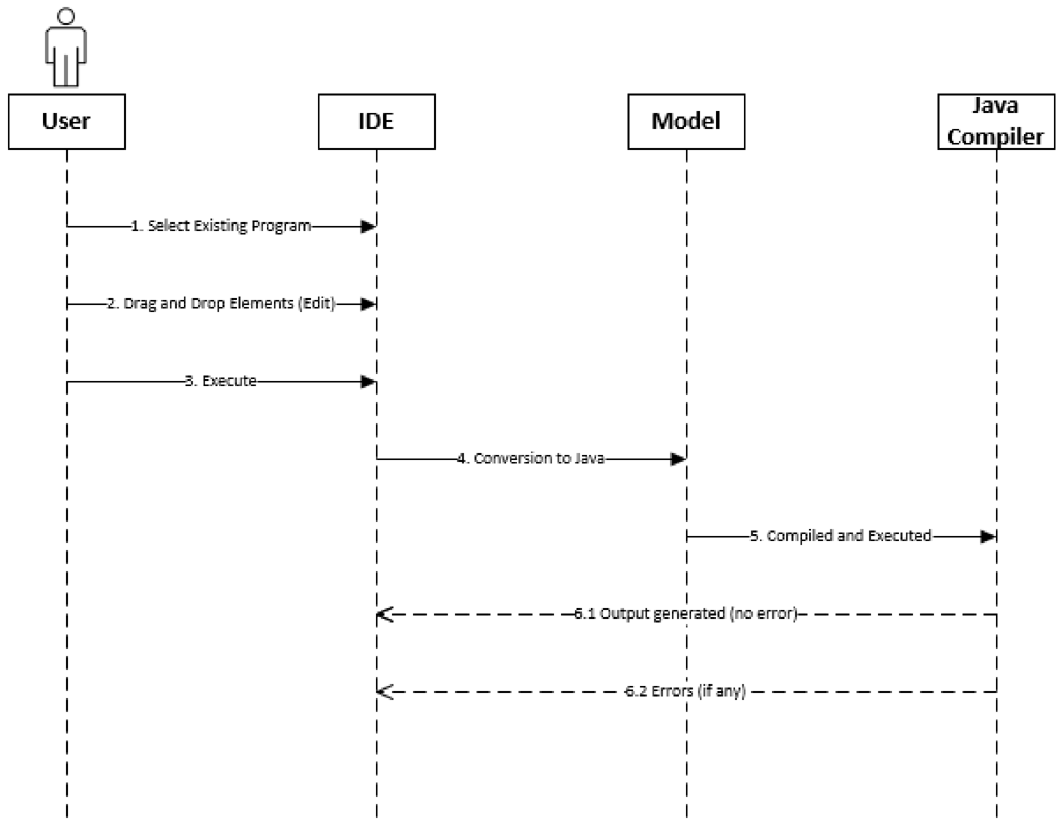


Figure 11 – Edit Program Sequence

4.2.3.3.3 Delete Program

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

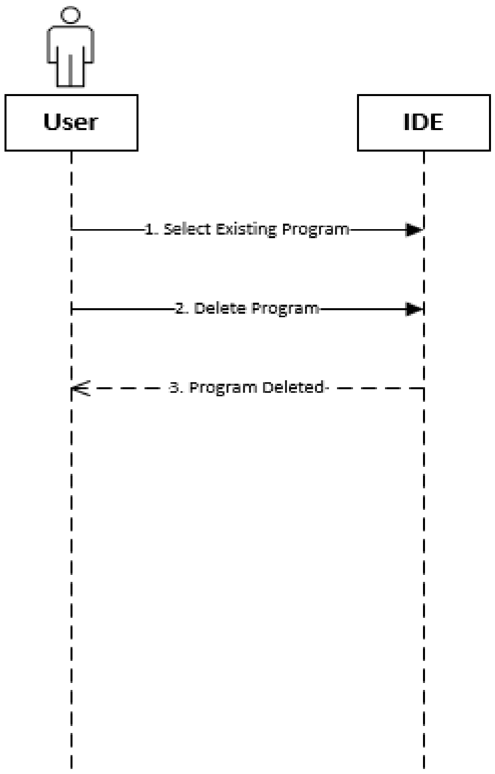


Figure 12 – Delete Program Sequence

4.2.3.4 UI Design

MDVPL is a desktop application and intended to be used by the users from diverse background knowledge. This requires that the interface of MDVPL 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.

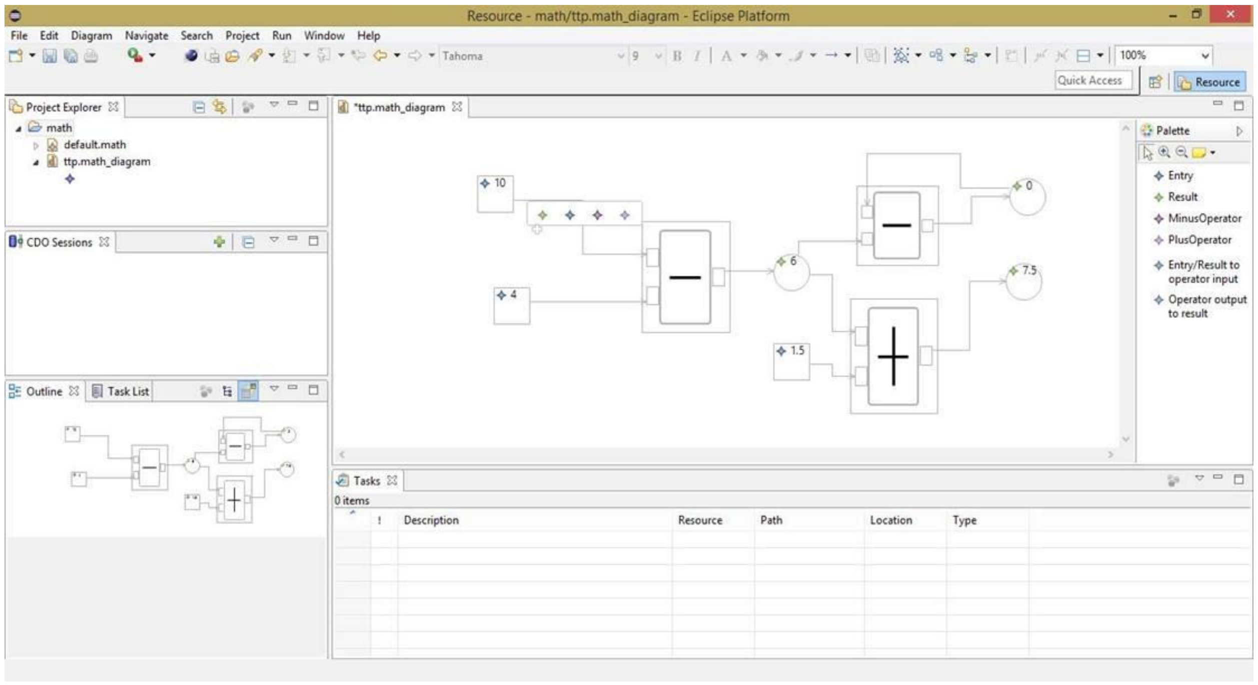


Figure 13 – MDVPL Interface

4.3 Detailed Description of Components

4.3.1 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.
Resources	A system with installed JDK.
Processing	When the IDE will start it will appear on top of the IDE.
Data	Array of button classes that will be used to hold different buttons in the Tool Bar.

4.3.2 Tool Box

Identification	ToolBox
Type	Component
Purpose	User interface to access different graphical elements of the language.
Function	Displays a list of different graphical objects which user will use to create a program.
Subordinate	A panel will hold all the graphical elements.
Dependencies	Calls the method of each graphical elements.
Interfaces	Selects a graphical element which is clicked.
Resources	Java Swing library to design the graphical elements.
Processing	When the IDE will start it will appear on the left side of IDE.
Data	The graphical elements will be available in a specific panel.

4.3.3 Work Space

Identification	Workspace
Type	Component
Purpose	Provide user with a space to create programs.
Function	Convert the graphical objects into java 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.

4.3.4 Graphical Elements

Identification	Graphics
Type	Class
Purpose	Provide different functionalities (if, while loop, logical operators, arithmetic operators) in the graphical form.
Function	To perform different operations from graphical elements.
Subordinate	A panel will hold all the text fields and drop down menus.
Dependencies	Graphical Elements are the basic building block of the language. These graphical units create all the programs and all the graphical elements are inter connected.
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 graphical element to perform its action.
Processing	Each graphical element is called by its specific call method when the user drags and drops it in the work space.
Data	Each graphical element will keep the input in specific variables and pass it to other variables in other connected graphical elements.

4.3.5 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.
Processing	The output display appears when the program executes by output () method.
Data	The output values will be stored in different variables types depending on the type of output data.

4.4 Reusability and Relationships to Other Products

Our Model 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. However, the strategic aspects of future reusability of its 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 a visual programming language by using model driven approach because in this way a user can only create programs by following the given model of the language without getting into language syntax complexities.
- It is model based so that it can become extendable and can be reused in future. Anyone can use it by just modifying the model according to his/her domain's requirements.
- A simpler user interface has been incorporated, so as to extend the usability of the language to all the novice users.
- To avoid the complexity of code, graphical elements are being provided to the user in the tool box.

4.6 Pseudo Code for Components

4.6.1 New Program

```
Begin
    Open IDE
    Click File tab
    Select new Program
    Drag and drop objects
    Execute the program
    If Program is correct
        Display the Output
    Else
        Display the error
        Go back to Drag/drop phase
End
```

4.6.2 Existing Program

```
Begin
    Open IDE
    Click File tab
    Select Existing Program
    Edit Program
    Drag and drop objects
    Execute the program
    If Program is correct
        Display the Output
    Else
        Display the error
        Go back to editing phase
End
```

4.6.3 Delete Program

```
Begin
    Open IDE
    Click File tab
    Select existing Program
    Delete program
End
```

Chapter 5: Project Analysis and Evaluation

5.1 Test plan Identifier:

This test plan aims for Model Driven Visual Programming Language Project. It is named as “MDVPL Functionality Testing”. The version for this test plan is Version 1.1.

5.2 Introduction:

The purpose of this document is to outline the test strategy and overall test approach for the MDVPL Project. This includes test methodologies, traceability, and resources required, and estimated schedule.

- This plan will include testing of mathematical, conditional statements, concatenation and looping functions, contained in MDVPL.
- It will also test the model of the language. It comes under component testing.
- 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.

5.3 Test Items:

- Math operators are being tested which are +,-,* /.
- Other graphical elements such as Entry, Result, Loops, Conditional statements, Concatenation and Connectors are also tested.
- Also model of the language, e.g. number of inputs given to the operator, is tested.

5.4 Features to be tested:

- Drag and drop operations of graphical entities are being tested.
- Connections made via connectors are tested.
- Also mathematical operations being performed are checked that whether they are operating correctly or not.
- Loops, conditional statements and concatenation functionalities are tested.

5.5 Features not to be tested:

The code generated behind the program is not tested. The complier provided by Eclipse is not tested.

5.6 Approach:

a. 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

b. Incremental Integration Testing

As the components will be developed from the bottom-up and top- down, the test strategy will also align to the order of development of components.This will utilize a mostly bottom-up integration test approach, but will also involve the sandwich integration test approach.

5.7 Item pass/fail criteria:

The result obtained from the applied tests are compared with the expected results. If the actual result is exactly the same as expected one then test case is marked as passed otherwise failed.

5.8 Suspension criteria and resumption 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 is 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

5.9 Test deliverables:

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

Test Case Name	Input Data Testing
Test Case Number	1
Description	In this test case, values entered in
Testing Techniques Used	Black Box Testing
Preconditions	There are no values inside the Entry
Input Values	Any value e.g. a,3
Valid Inputs	All numeric values e.g. 3,5
Steps	• Enter one input an integer • Enter other input an alphabet • Connect both with inputs of the operator
Expected Output	No result obtained(in case other
Actual Output	No result obtained
Status	Passed

Test Case Name	**Mathematical Operator**
Test Case Number	2
Description	In this test case, functionalities of the mathematical operators i.e. +,- ,*, / are tested
Testing Techniques Used	Black Box testing
Preconditions	No values are given to the input of the
Input Values	Any numeric value e.g. 4,3
Valid Inputs	All numeric values e.g. 4,3
Steps	• Enter integer values by using two entry elements. • Connect both with inputs of the math operator
Expected Output	The numeric values are
Actual Output	The numeric values are
Status	Passed

Test Case Name	**Connectors Connection Testing**
Test Case Number	3
Description	In this test case, the way the
Testing Techniques Used	Black Box testing
Preconditions	Values are in the Entry elements.
Input Values	Input to Entry/Result connector is
Valid Inputs	Entry/Result to Input connector is
Steps	• Enter integer values by using two Entry elements. • Connect both with inputs of the operator using Input to Entry/Result connector
Expected Output	Does not work
Actual Output	Does not work
Status	Passed

Test Case Name	Operators Connection Testing
Test Case Number	4
Description	In this test case, the way the
Testing Techniques Used	Black Box testing
Preconditions	Values are in the Entry elements.
Input Values	One operator connected directly
Valid Inputs	The output from operator to be connected with the Result entity first and then further connected to the other operator as input
Steps	• Enter integer values by using two Entry elements. • Connect both with inputs of the operator. • Connect the output of the operator with the input of another operator • Connect output of the second operator with the Result element
Expected Output	No Output
Actual Output	No Output
Status	Passed

5.10 Testing task

Testing Tasks	Assi gne
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

5.11 Environmental needs:

a. Hardware:

No hardware required b.

b. Software:

Eclipse luna

5.12 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

	• 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.

5.13 Staffing and training needs:

N/A

5.14 Schedule:

Test Phase	Time	Owner
Test Plan Creation	1 week	Test Manager
Test Specification	2 weeks	Test Leads
Test Specification Team	1 week	Project Team
Component Testing	4 weeks	Component Testers
Incremental Integration	4 weeks	Component & System

5.15 Risks and contingencies:

- The plugins may not be available. Need to download them
- JDK version compatibility issues may occur
- Eclipse may hang during execution of the program
- May not get the expected output

5.16 Approvals:

It is approved by all the team members of MDVPL Project

Chapter 6: Future work

The project developed could then be used as a basis for further work in the field of education for computer science students. The MDVPL is model based and therefore it can be enhanced and additional features can be added in to it by developer.

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))

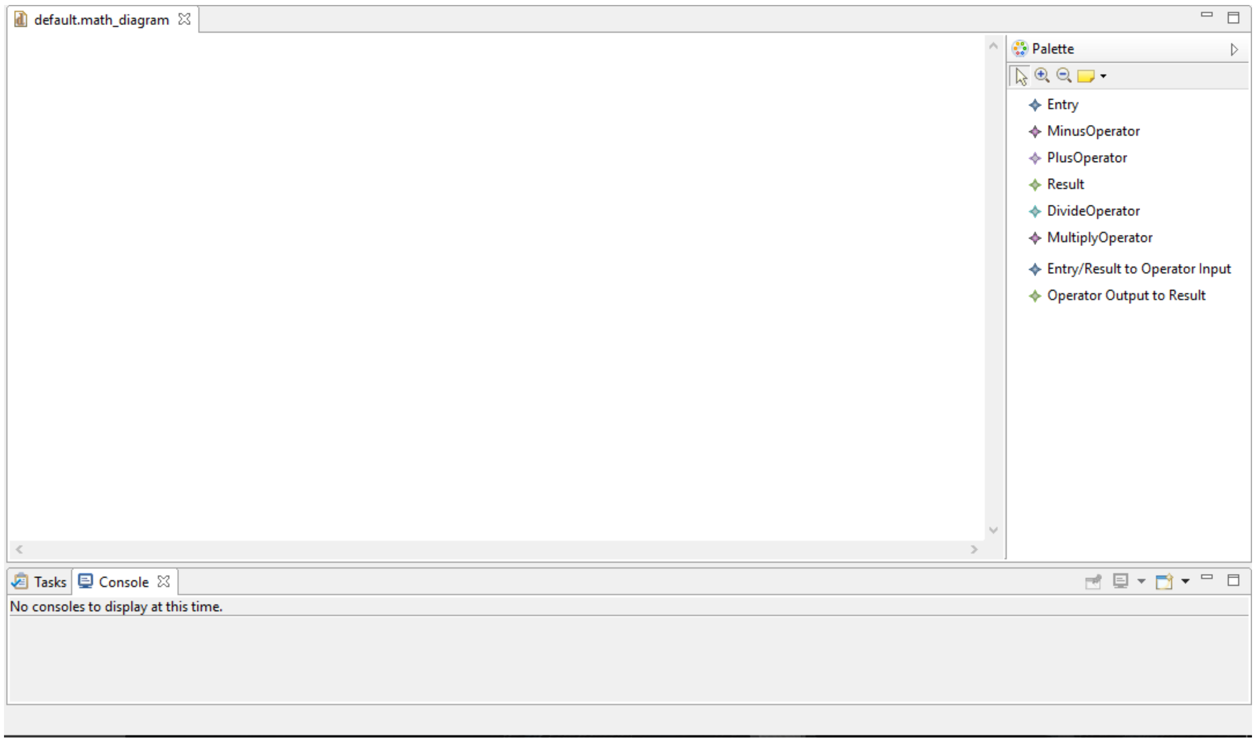
- Model Driven Architecture (https://en.wikipedia.org/wiki/Model-driven_architecture)

- 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)

Annex A

Tutorial for users

WORK SPACE TO CREATE PROGRAMS

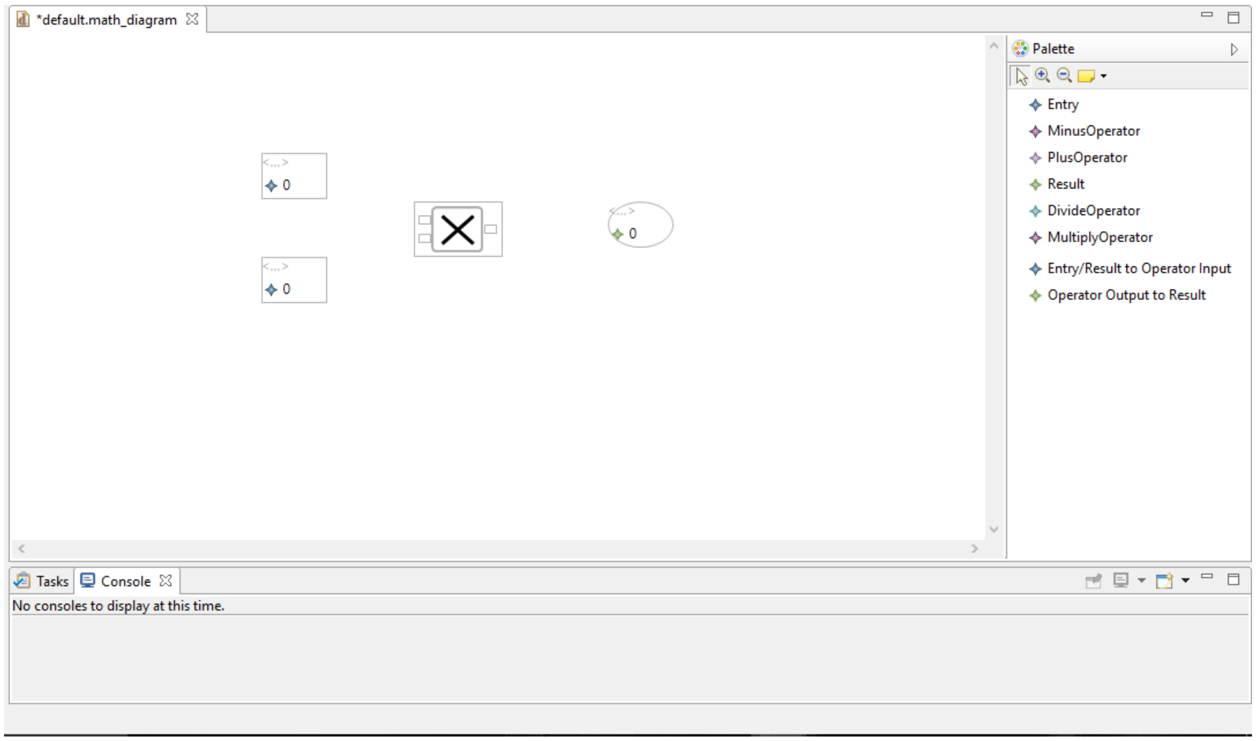


No consoles to display at this time.

<u>Create a Program Using Arithmetic Operators</u>

<u>STEP 1:</u>

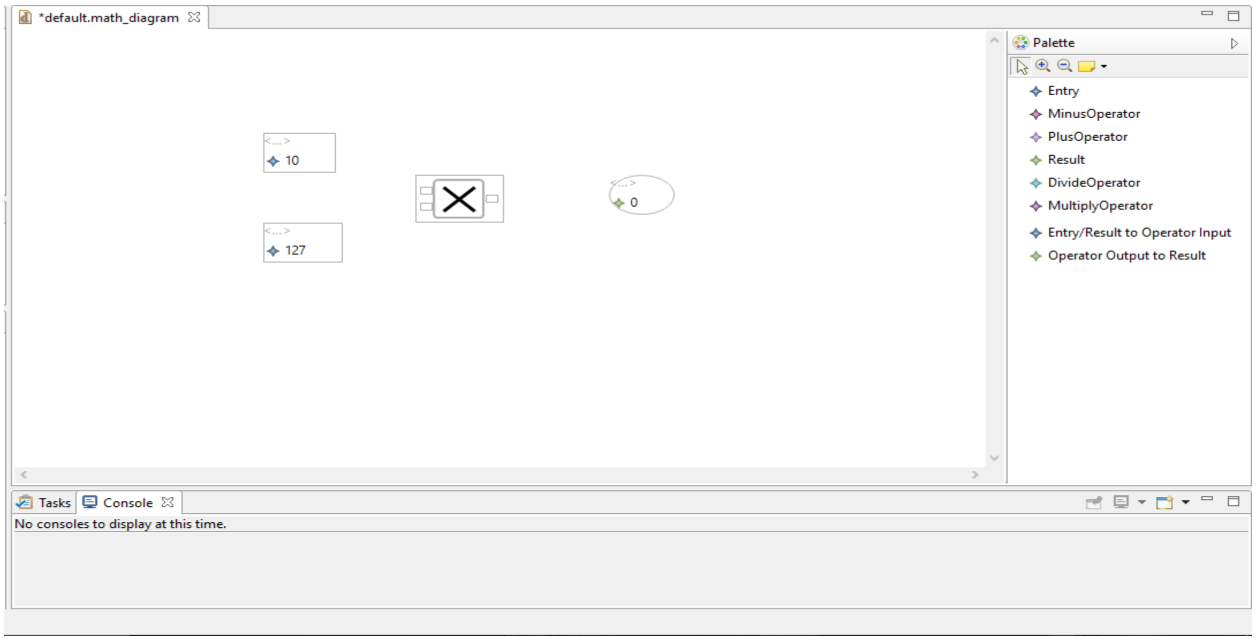
- Click on Entry element in the Palette.
- Then click anywhere in the workspace.
- Again Click on the Entry and repeat the same.
- Now select any operator and click in the workspace.
- Select Result element and drop it into the workspace.
- **You should get this output.**



No consoles to display at this time.

STEP 2:

- Now enter any integer in both entry elements.

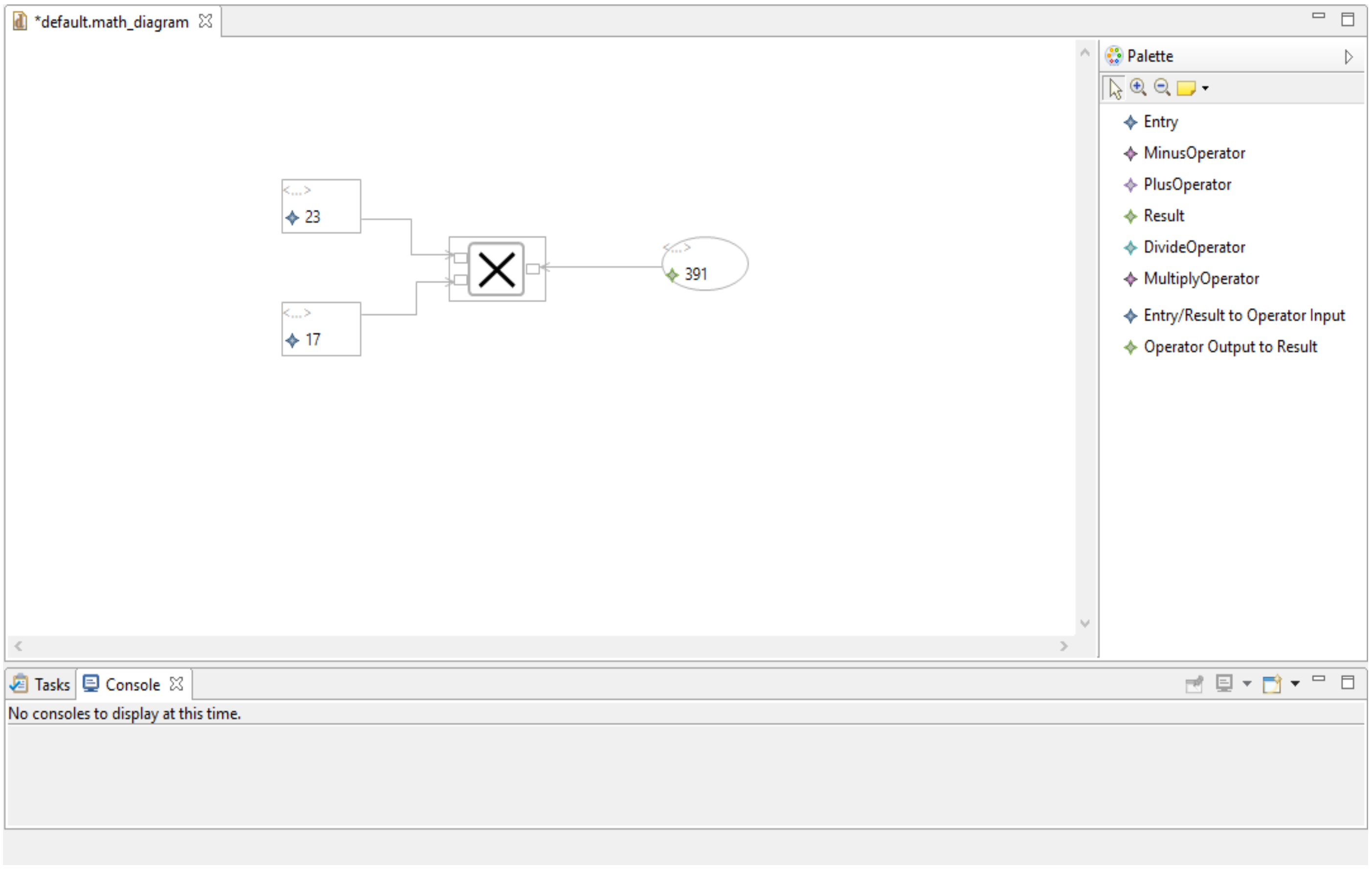


No consoles to display at this time.

STEP 3:

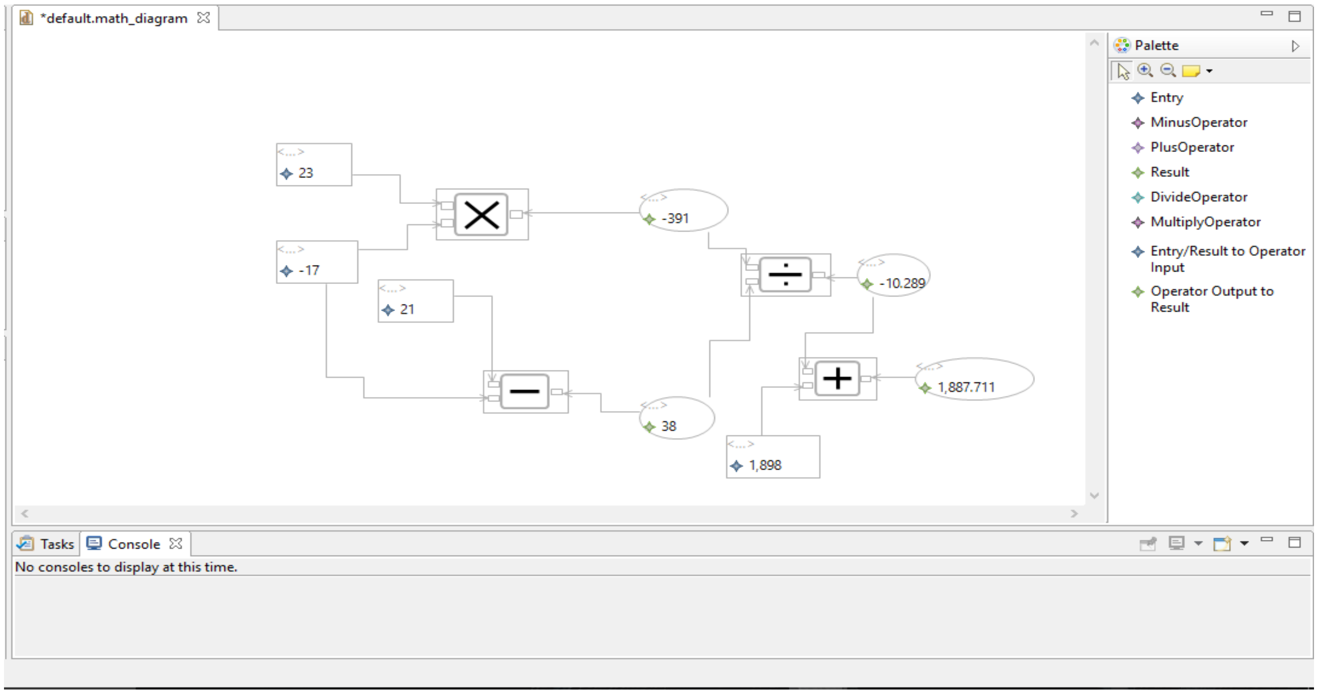
- Select Entry/Result to Operator Input element from palette.

- Click on any entry element and drag the connector to anyone of the input node of the operator.
- Again select Entry/Result to Operator Input element and click on the other entry element and drag the connector to the other input node of the operator.
- Next select Operator Output to Result element and click on the result element and drag the connector to the output node of the operator.
- Now you should get final result like this:



No consoles to display at this time.

- In this way you can make any number of big calculations using different type of multiple operators.
- And you can also use the result values as inputs for other operators.
- For example:



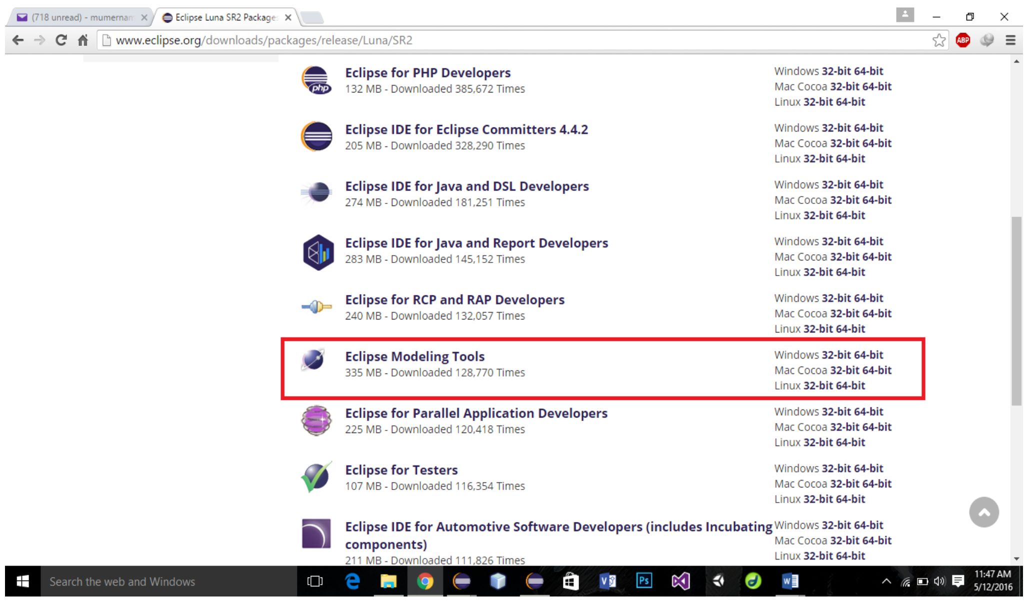
No consoles to display at this time.

Annex B

Tutorial for Developer

STEP No. 1: Setting Up the Environment

- Goto : www.eclipse.org/downloads/packages/release/Luna/SR2
- Find Eclipse Modelling Tools and select your operating system.



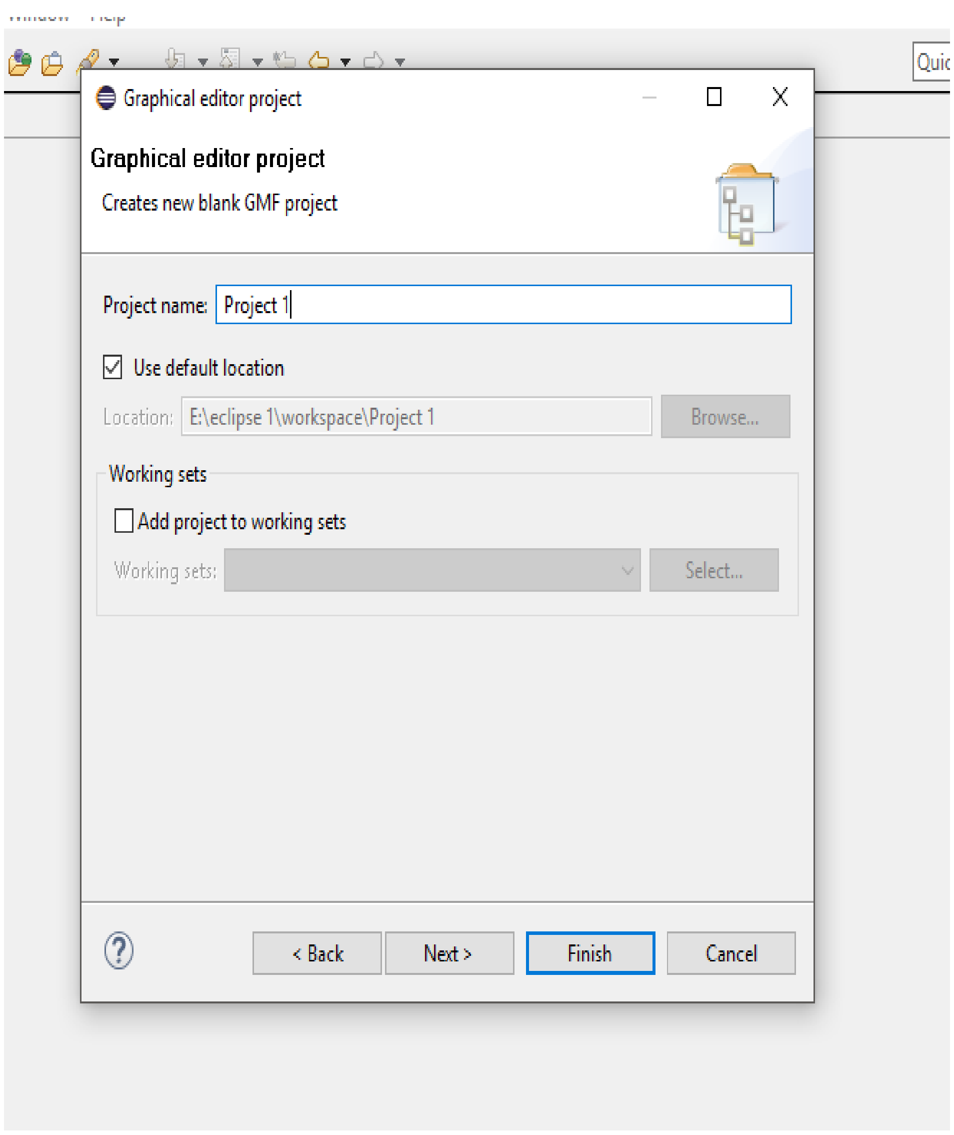
(718 unread) - mumernan
Eclipse Luna SR2 Package
www.eclipse.org/downloads/packages/release/Luna/SR2

Eclipse for PHP Developers 132 MB - Downloaded 385,672 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse IDE for Eclipse Committers 4.4.2 205 MB - Downloaded 328,290 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse IDE for Java and DSL Developers 274 MB - Downloaded 181,251 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse IDE for Java and Report Developers 283 MB - Downloaded 145,152 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse for RCP and RAP Developers 240 MB - Downloaded 132,057 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse Modeling Tools 335 MB - Downloaded 128,770 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse for Parallel Application Developers 225 MB - Downloaded 120,418 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse for Testers 107 MB - Downloaded 116,354 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**
Eclipse IDE for Automotive Software Developers (includes Incubating components) 211 MB - Downloaded 111,826 Times	Windows **32-bit 64-bit** Mac Cocoa **32-bit 64-bit** Linux **32-bit 64-bit**

Search the web and Windows
11:47 AM
5/12/2016

- After Download is complete extract the file in any drive.
- Now open the eclipse ide from the extracted folder.
- Go to File

  New → Others → Graphical Modelling Framework →Graphical Editor Project
- After selecting Graphical Editor Project, on the next screen type project name and click finish.



Graphical editor project

Graphical editor project

Creates new blank GMF project

Project name: Project 1

☑ Use default location

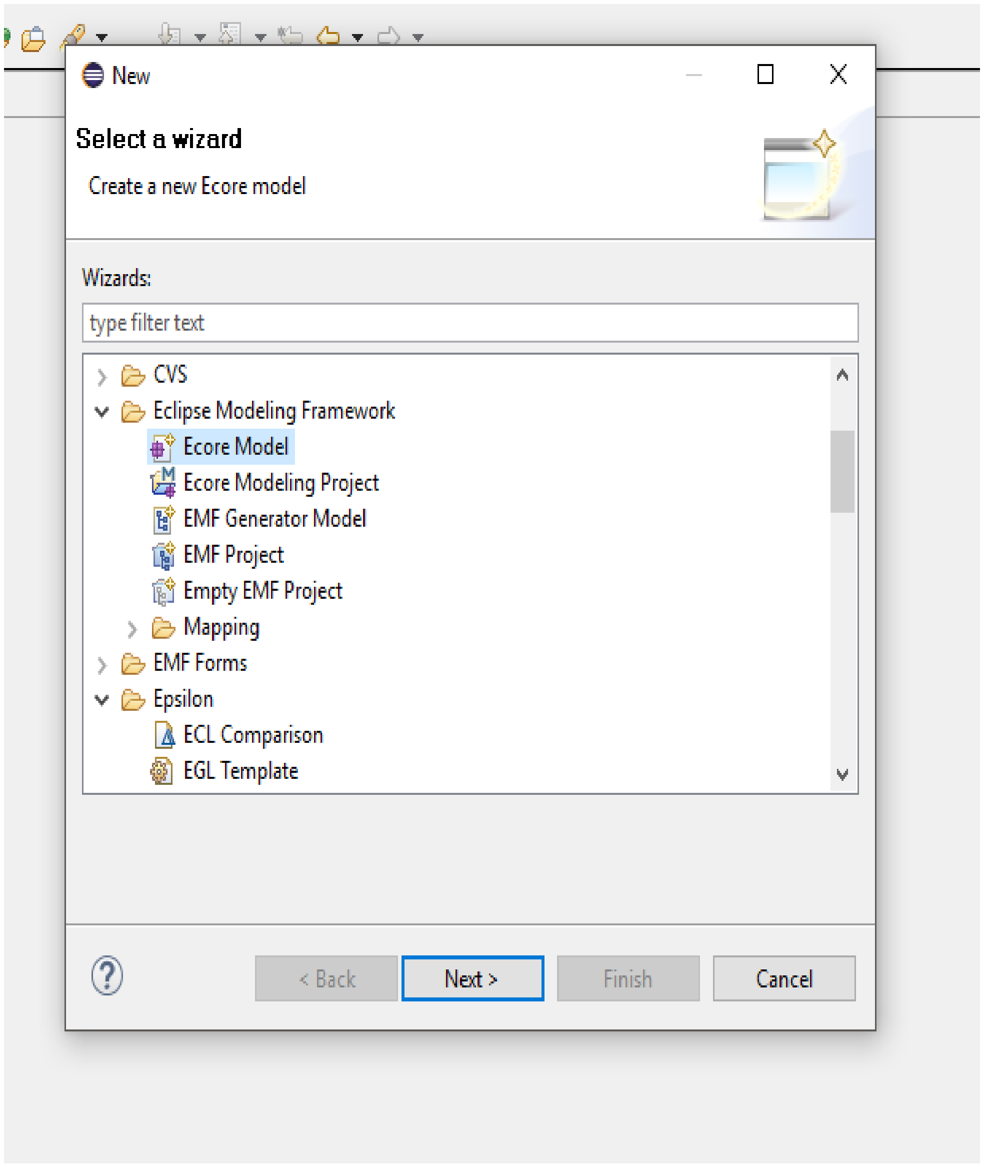
Location: E:\eclipse 1\workspace\Project 1

Working sets

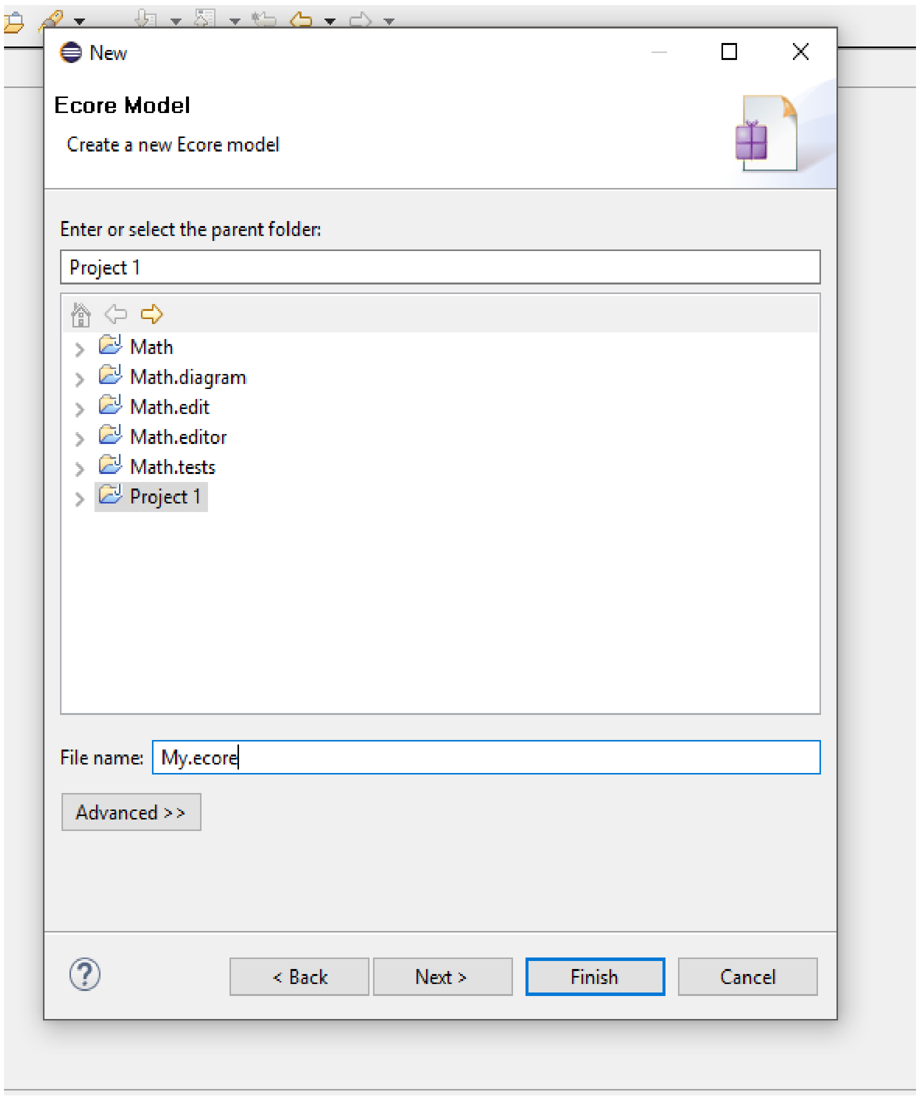
☐ Add project to working sets

Working sets:

- In package explorer right click on your project and GO to New → Others →Eclipse Modeling Framework → Ecore Model



- Select Ecore Model and click next.
- On next window select your current project and give name to your ecore model and click finish.


New

Ecore Model

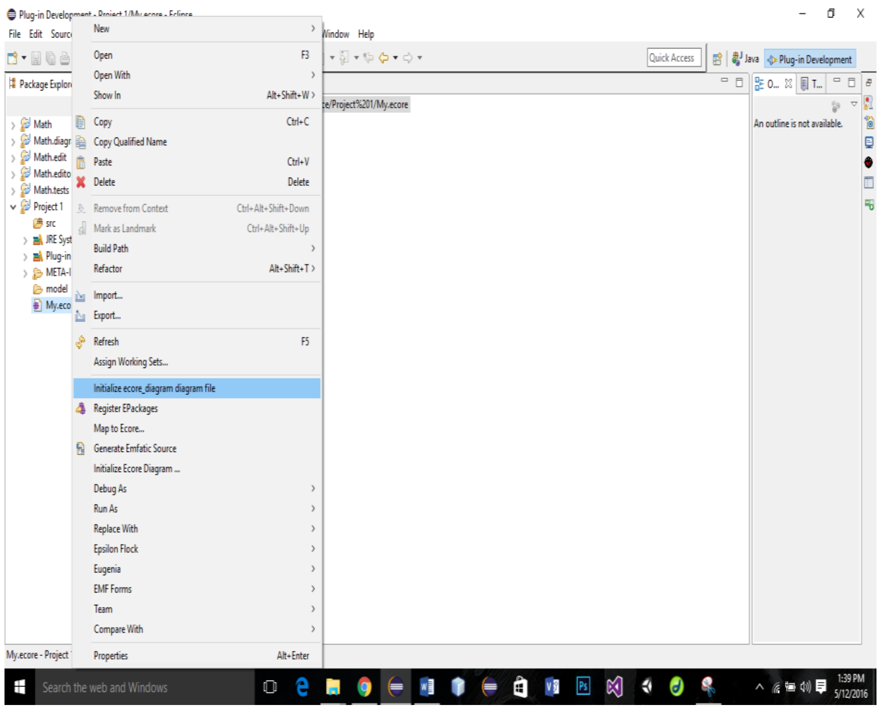
Create a new Ecore model

Enter or select the parent folder: Project 1

- Math
- Math.diagram
- Math.edit
- Math.editor
- Math.tests
- Project 1

File name: My.ecore

- Now in you Project Explorer right click on your Ecore model which is located in the model folder and click on "**initialize ecore_diagram digram file**".



- Then select the name of your file and click finish.

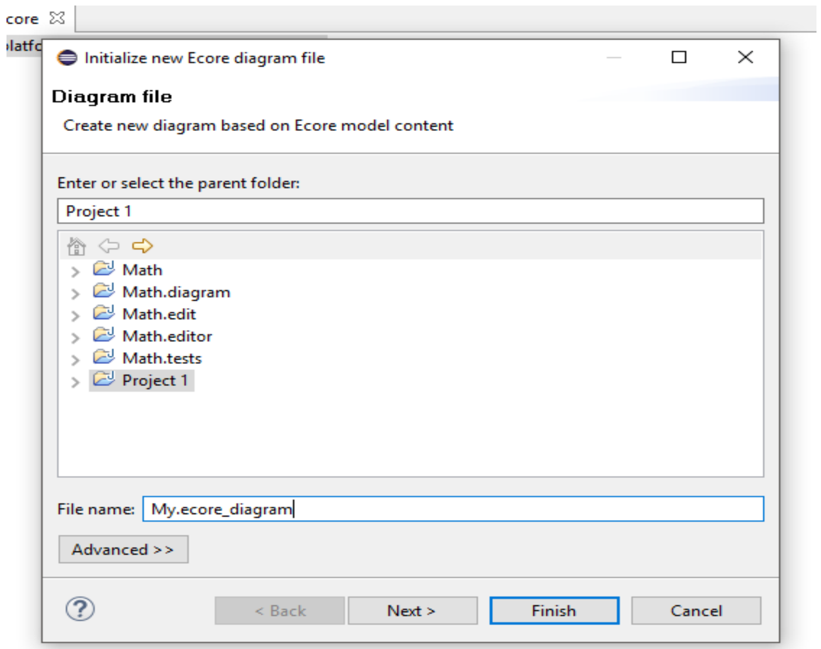

Initialize new Ecore diagram file

Diagram file

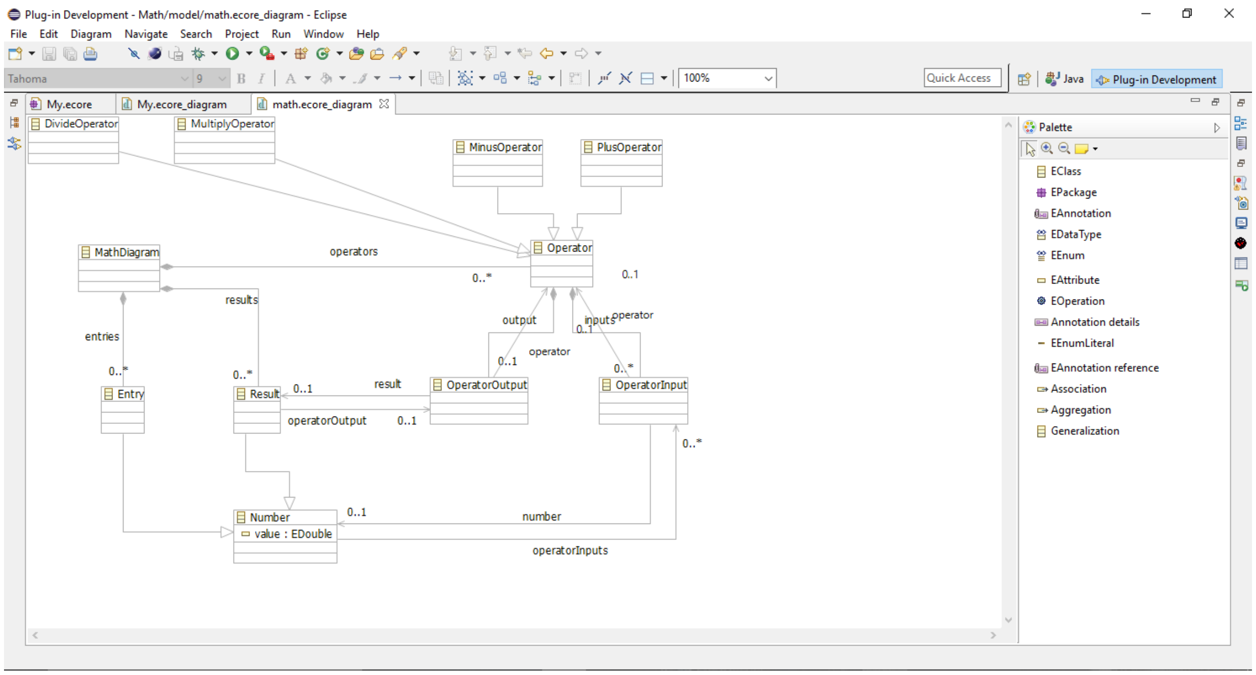
Create new diagram based on Ecore model content

Enter or select the parent folder: Project 1

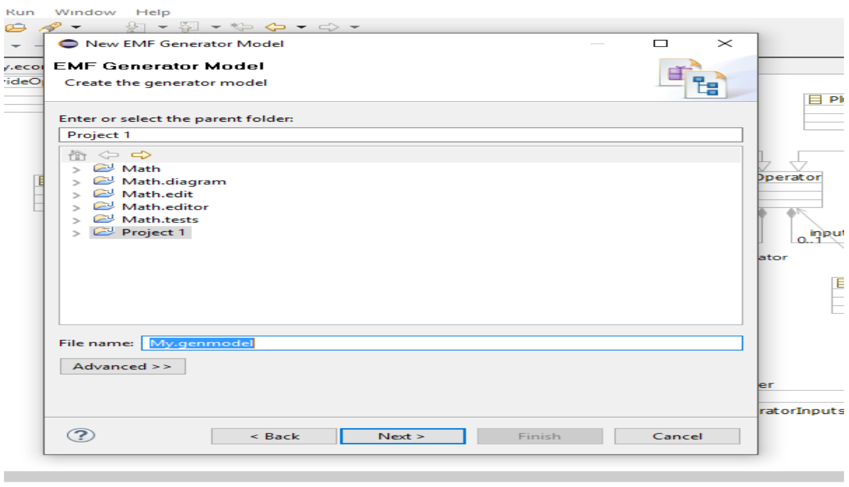
- Math
- Math.diagram
- Math.edit
- Math.editor
- Math.tests
- Project 1

File name: My.ecore_diagram

- Now ecore_diagram file will be opened in a new tab.
- In this ecore_diagram file you will create the model of your program.



- After creating the model again right click on your ecore file and
  Go to : New → Others →Eclipse Modeling Framework → EMF Generator Model and click next.
- Now select the name of your genmodel and click next.



New EMF Generator Model

EMF Generator Model

Create the generator model

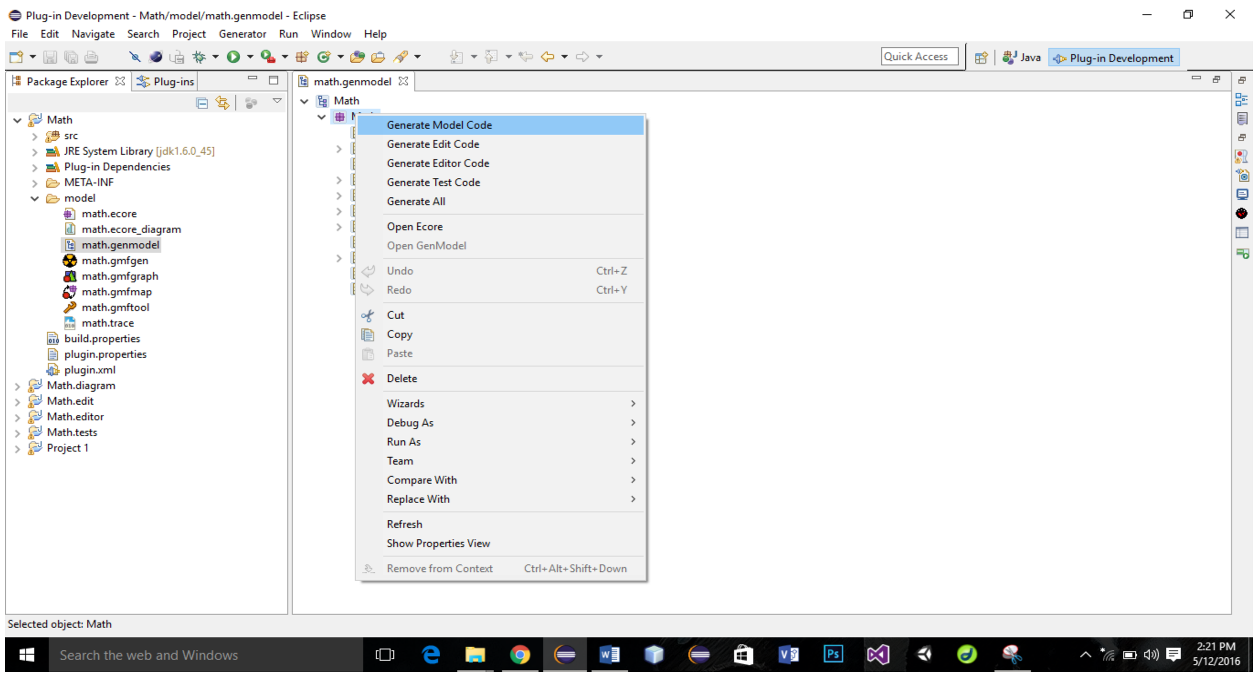
Enter or select the parent folder:

Project 1

File name: My.genmodel

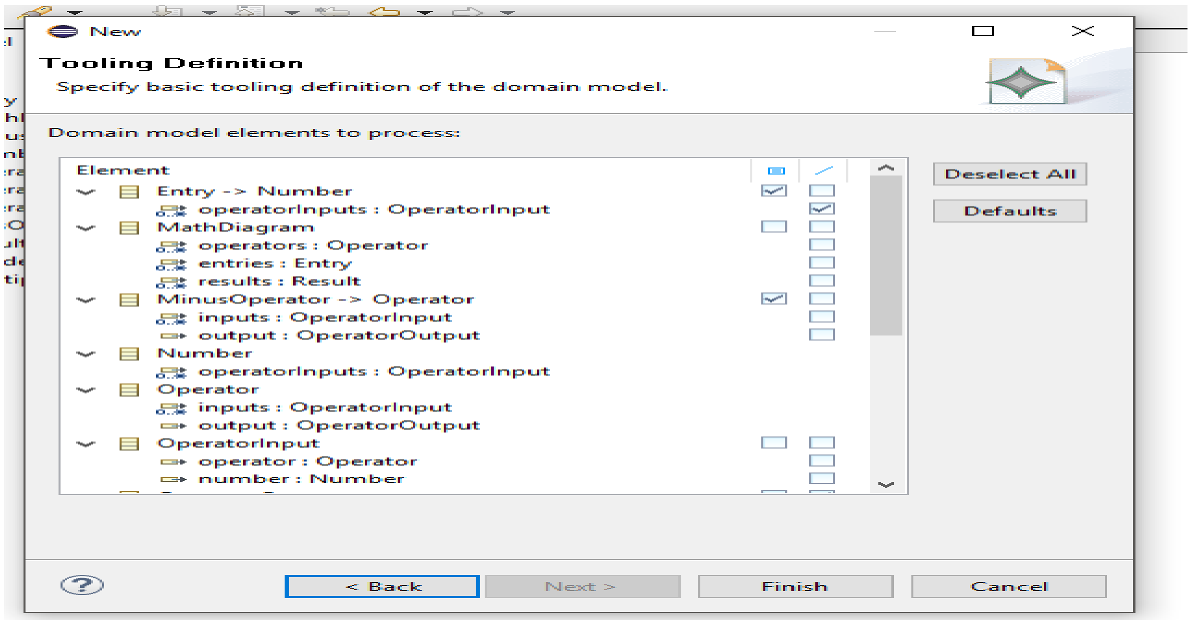
- Now select Ecore model on next window and click finish.

- Now open your genmodel file and right click on the name of genmodel and select the following one by one:
    1. Generate model code
    2. Generate edit code.
    3. Generate editor code.



- This will create two new folders edit folder and editor folder.
- Now once again go to your ecore file.
- Right click on it and go to:

  New → Others → Graphical Modeling Framework →Simple Tooling Definition Model and click next.
- On tooling definition screen select your nodes and connections and click finish.


New

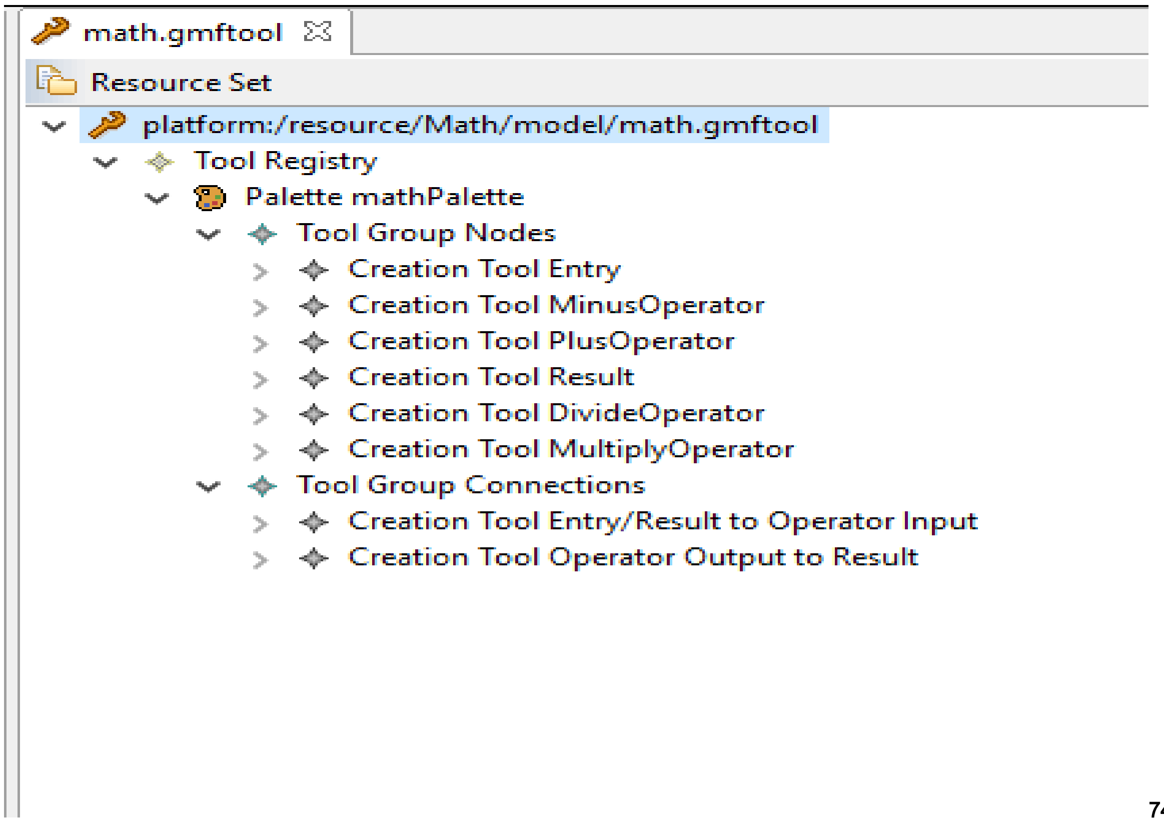
Tooling Definition

Specify basic tooling definition of the domain model.

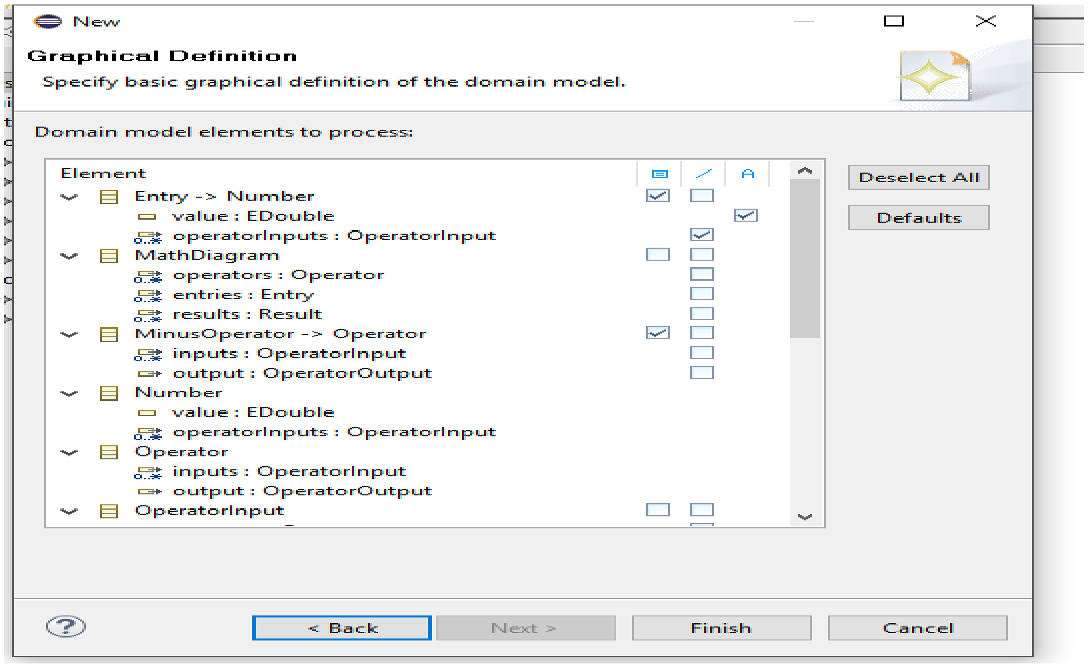
Domain model elements to process:

```
Element
  Entry -> Number                                   [✓] [ ]
    operatorInputs : OperatorInput                      [✓]
  MathDiagram                                       [ ] [ ]
    operators : Operator                                [ ]
    entries : Entry                                     [ ]
    results : Result                                    [ ]
  MinusOperator -> Operator                         [✓] [ ]
    inputs : OperatorInput                              [ ]
    output : OperatorOutput                             [ ]
  Number
    operatorInputs : OperatorInput
  Operator
    inputs : OperatorInput
    output : OperatorOutput
  OperatorInput                                     [ ] [ ]
    operator : Operator                                 [ ]
    number : Number                                     [ ]
```

- Now open gmftool file and create two groups, one for nodes and one for connectors.


```
math.gmftool
Resource Set
  platform:/resource/Math/model/math.gmftool
    Tool Registry
      Palette mathPalette
        Tool Group Nodes
          Creation Tool Entry
          Creation Tool MinusOperator
          Creation Tool PlusOperator
          Creation Tool Result
          Creation Tool DivideOperator
          Creation Tool MultiplyOperator
        Tool Group Connections
          Creation Tool Entry/Result to Operator Input
          Creation Tool Operator Output to Result
```

- Now again go to:
- New → Others → Graphical Modeling Framework →Simple Graphical Definition Model and click next.
- On the Graphical definition screen select your nodes,connectors and labels. Then click finish.



New

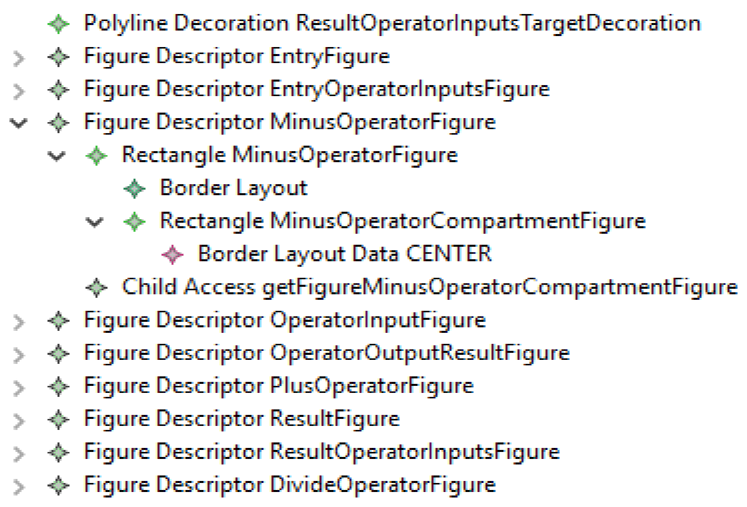
Graphical Definition

Specify basic graphical definition of the domain model.

Domain model elements to process:

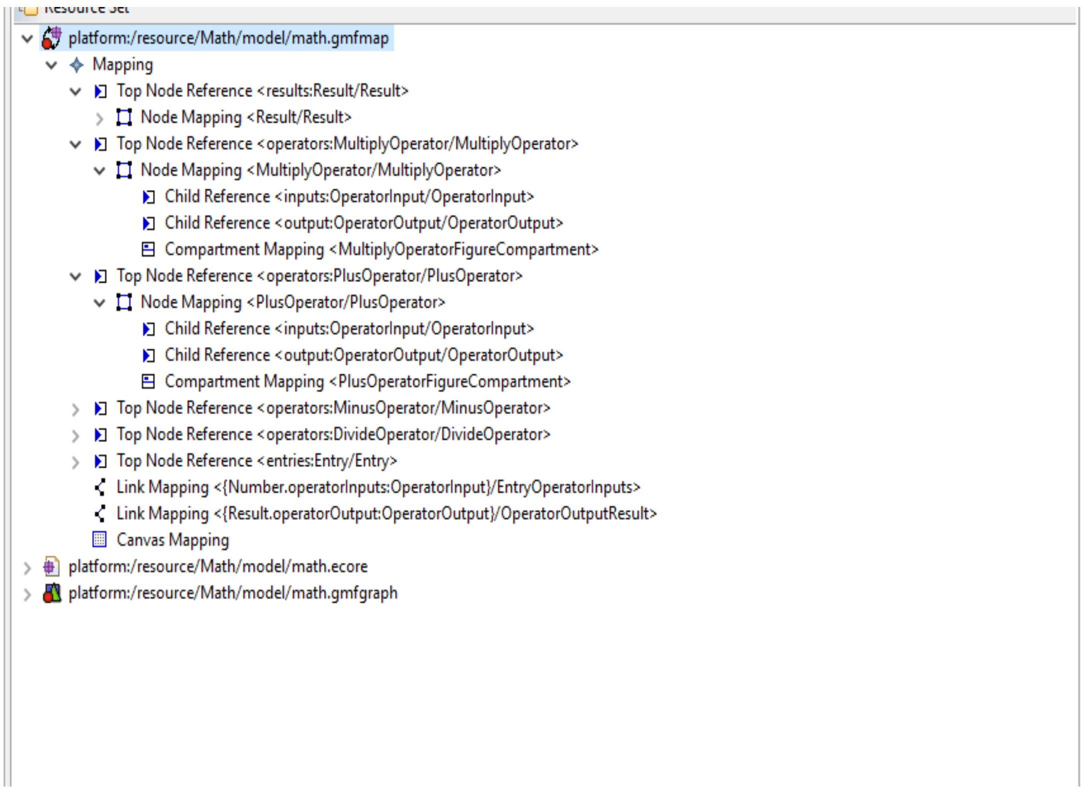
Element	Node	Link	Label
Entry -> Number	✓	☐	
value : EDouble			✓
operatorInputs : OperatorInput		✓	
MathDiagram	☐	☐	
operators : Operator		☐	
entries : Entry		☐	
results : Result		☐	
MinusOperator -> Operator	✓	☐	
inputs : OperatorInput		☐	
output : OperatorOutput		☐	
Number			
value : EDouble			
operatorInputs : OperatorInput			
Operator			
inputs : OperatorInput			
output : OperatorOutput			
OperatorInput	☐	☐	

- Open gmfgraph file define how your nodes will appear in the end of project to user.



Polyline Decoration ResultOperatorInputsTargetDecoration
Figure Descriptor EntryFigure
Figure Descriptor EntryOperatorInputsFigure
Figure Descriptor MinusOperatorFigure
    Rectangle MinusOperatorFigure
        Border Layout
        Rectangle MinusOperatorCompartmentFigure
            Border Layout Data CENTER
    Child Access getFigureMinusOperatorCompartmentFigure
Figure Descriptor OperatorInputFigure
Figure Descriptor OperatorOutputResultFigure
Figure Descriptor PlusOperatorFigure
Figure Descriptor ResultFigure
Figure Descriptor ResultOperatorInputsFigure
Figure Descriptor DivideOperatorFigure

- Now again select your ecore file and right click on it and Go to: New → Others → Graphical Modeling Framework →Guide Mapping Model Creation and click next.
- Select name for your file and click finish.
- Open gmfmap file and link gmfgraph with gmftool

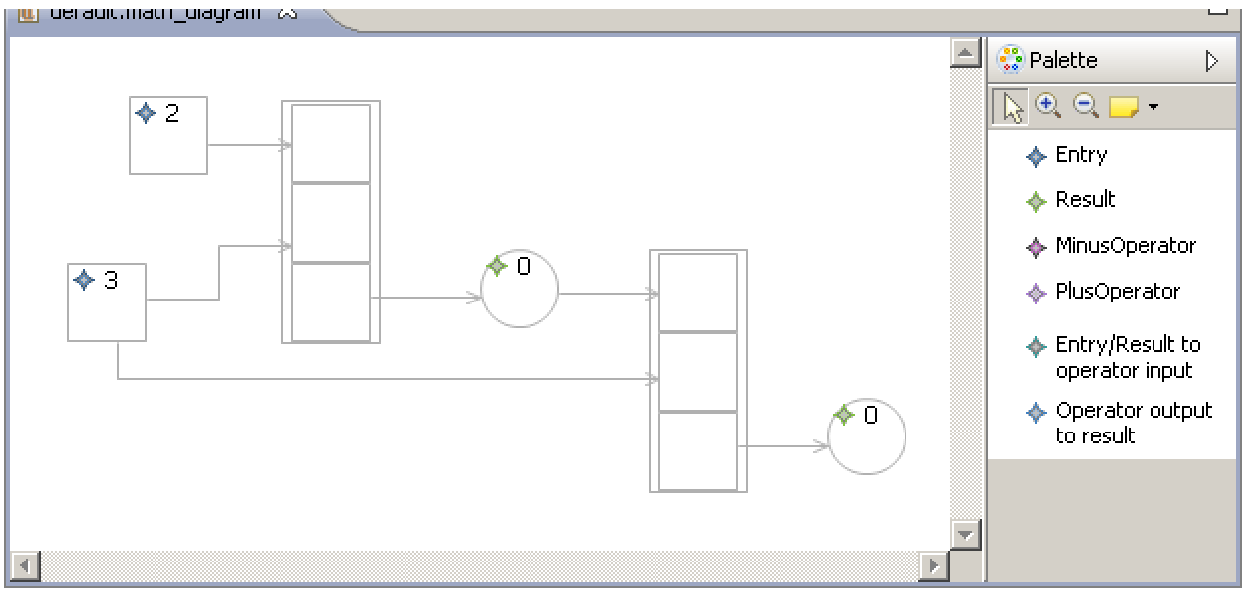


Resource Set
- platform:/resource/Math/model/math.gmfmap
  - Mapping
    - Top Node Reference <results:Result/Result>
      - Node Mapping <Result/Result>
    - Top Node Reference <operators:MultiplyOperator/MultiplyOperator>
      - Node Mapping <MultiplyOperator/MultiplyOperator>
        - Child Reference <inputs:OperatorInput/OperatorInput>
        - Child Reference <output:OperatorOutput/OperatorOutput>
        - Compartment Mapping <MultiplyOperatorFigureCompartment>
    - Top Node Reference <operators:PlusOperator/PlusOperator>
      - Node Mapping <PlusOperator/PlusOperator>
        - Child Reference <inputs:OperatorInput/OperatorInput>
        - Child Reference <output:OperatorOutput/OperatorOutput>
        - Compartment Mapping <PlusOperatorFigureCompartment>
    - Top Node Reference <operators:MinusOperator/MinusOperator>
    - Top Node Reference <operators:DivideOperator/DivideOperator>
    - Top Node Reference <entries:Entry/Entry>
    - Link Mapping <{Number.operatorInputs:OperatorInput}/EntryOperatorInputs>
    - Link Mapping <{Result.operatorOutput:OperatorOutput}/OperatorOutputResult>
    - Canvas Mapping
- platform:/resource/Math/model/math.ecore
- platform:/resource/Math/model/math.gmfgraph

- Click on the transform label of the dashboard.
- Abc.gmfgen file is created
- Change the properties of this file as per requirement.
- Click on the Generate diagram editor label of the dashboard.
- Now to have two inputs and one output in every operator. Following changes in the code have to be made OperatorCreateCommand file.

```
        // Adds both operator inputs

            newElement.getInputs().add(MathFactory.eINSTANCE.createOperatorInput());

            newElement.getInputs().add(MathFactory.eINSTANCE.createOperatorInput());

                    // Adds the operator output

            newElement.setOutput(MathFactory.eINSTANCE.createOperatorOutput());
```

Starting Eclipse at this point, we get somewhat this interface.



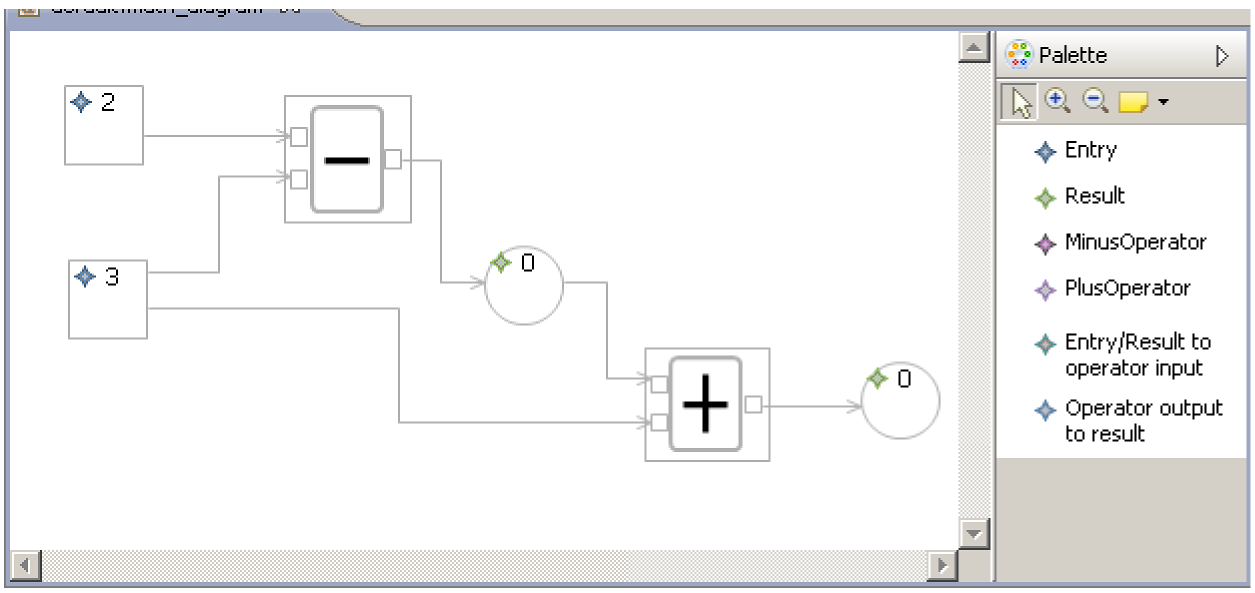
- Next we need to set shapes of our operators.
- We need to make changes in OperatorRoundedRectangle classes.
- Following code has to be written down in classes of all operators.

```
// vertical line
                    graphics.drawLine(
                                        new Point(r.x + r.width / 2, r.y + r.height * 0.2),
                                        new Point(r.x + r.width / 2, r.y + r.height * 0.8));
                    // horizontal line
                    graphics.drawLine(
                                        new Point(r.x + r.width * 0.2, r.y + r.height / 2),
                                        new Point(r.x + r.width * 0.8, r.y + r.height / 2));
```

- Now we need to add our rounded rectangles to the graphical representations of our operators.
- To do that we need to edit (Minus/Plus)Operator(Minus/Plus)OperatorFigureCompartmentEditPart.java file and modify the 'createFigure' method.
- We need to do it for all Operators.
- Following code is to be added.

```
public IFigure createFigure() {
                    ResizableCompartmentFigure result = (ResizableCompartmentFigure)
super.createFigure();
                    result.setTitleVisibility(false);

                    // Setup for a XYLayout
                    IFigure contentPane = result.getContentPane();
                    contentPane.setLayoutManager(new XYLayout());

                    // Delete content pane insets
                    Insets is = contentPane.getInsets();
                    is.top = 0;
                    is.bottom = 0;
                    is.left = 0;
                    is.right = 0;

                    // Setup graphical elements
          MinusRoundedRectangle roundedRectangle = new MinusRoundedRectangle();
                    contentPane.add(roundedRectangle);

                    // Add the resize events listener
                    result.addFigureListener(new
OperatorCompartmentFigureListener(this, roundedRectangle));

                    return result;
          }
```

- Now we will cater for figure resizeability.
- For that we need to make changes in OperatorCompartmentFigureListener class.
- Starting Eclipse at this point we should get a diagram editor like this.

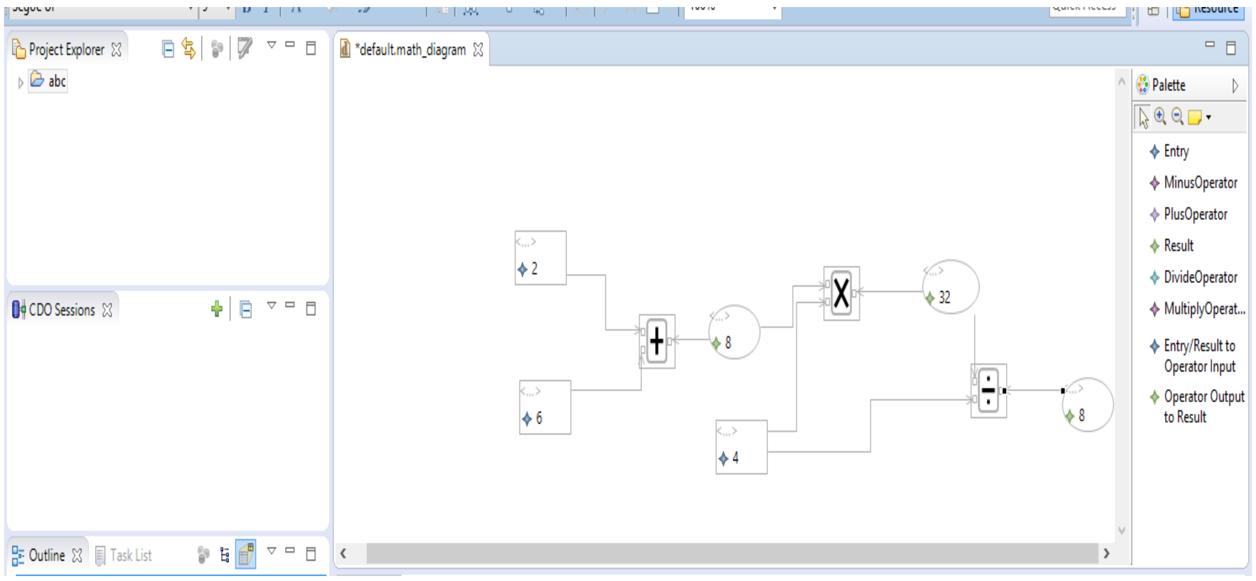


- Now we are going to add features that will automatically compute operations.
- Following code is to be present in AutomaticComputationHelper class to get operators work.

```
public static void updateOperatorResult(Operator operator) {
            Result result = operator.getOutput().getResult();
            if (result!=null) {
                                    // If there is a cycle...
            if (CycleDetectionHelper.cycleDetected(result)) {
                                                result.setValue(0);
                                    }
                                    else {
Number in1 = operator.getInputs().get(0).getNumber();
Number in2 = operator.getInputs().get(1).getNumber();
double _in1 = in1 != null ? in1.getValue() : 0;
double _in2 = in2 != null ? in2.getValue() : 0;
result.setValue(operator instanceof PlusOperator ? _in1 + _in2 : _in1 - _in2);
                                    }
```

```
                    }
          }
```

- We need to do the same thing for all operators.
- Finally we are finished with two operators.
- We have following final output.



Project Explorer
abc
CDO Sessions
Outline
Task List
*default.math_diagram
Palette
Entry
MinusOperator
PlusOperator
Result
DivideOperator
MultiplyOperat...
Entry/Result to Operator Input
Operator Output to Result
2
6
8
32
4
8