## Universitatea Tehnică “Gheorghe Asachi” din Iași Facultatea de Automatică și Calculatoare Domeniul: Calculatoare și Tehnologia Informației Anul 3

Logical Scheme Interpreter

-documentație-

**Realizat de:** Vlad Batalan

# Cîrstea Cătălin Lucian – Mircea Mihai Adrian - Pașcu Alexandru Nicolae

**Grupa:** 1308A

**Software requirements specifcation**

## Introduction

* 1. **Purpose**

Scheme programming language has the characteristics of both a compiled and an interpreted language. Also, it is a very nice language for implementing languages, or for transformational programming in general which can be translated as: writing programs that write programs.

## Scope

The reason behind this project is to create an application that makes logical decisions based on our choices when it comes to us creating a program without a wide variety of programming knowledge.

## Definitions, Acronyms and Abbreviations

**Scheme Programming language:**  is typically used to write to write text editing programs, analyse and optimise application-based compilers, write drivers for graphics cards, develop operating systems, and many more.

**SRS:** is a document that describes a software to be developed.

## References

[Programează cu Blockly | www.pbinfo.ro](https://www.pbinfo.ro/?pagina=blockly)

## Overview

Section 2 presents in more depth and completeness the requirements of this application.

This section is intended for the customer or others who are unfamiliar with software engineering. It contains very little technical data and targets people who do not have a technical training in this field.

Section 3 will contain all technical requirements for the application including a list of functional and non-functional requirements.

This section is intended for software developers to create this program. Most of the information presented there is very technical and so it is not advisable to be read or understood by someone outside the field.

## Overall Description

* 1. **Product Perspective**

Logical Scheme Interpretor was created entirely by us to understand how programming works. It is not the only or the first product of this kind on the market but it can bring an easier and new aspect to use.

## Product Functions

The application allows the following actions that can be performed by the user in the graphical interface:

* + - Choosing which command to add to the program
    - Entering variables from the keyboard
    - Entering text from the keyboard
    - Entering values ​​for variables from the keyboard
    - Running the created program

## User Characteristics

The application can be used by anyone in any field without the need for advance technical training. The primary users will be the IT people who will use the application for a better understanding of programming.

## General Constraints

A first constraint or limitation is that once you press enter you can not return to the textbox to change the added value.

Also it is limited to Windows as an operating system, and both mouse and keyboard will be used as hardware.

## Assumptions and Dependencies

Let’s suppose that this application is used by a programmer who is just starting out and wants to understand the logic behind programming. He will quickly notice the ease with which he can create a runnable program, and will also have free rein to the imagination to create any problem in day-by-day life. After learning the mechanism of programming, it will have a formed judgment and will understand much more easily the programming languages and the correct way of solving problems due to the very well structured thinking.

1. **Specific Requirements**
   1. **External Interface Requirements**
      1. **User interfaces**
         * Front-end software: Windows Forms App (.NET Framework)
         * Back-end software: C#

## The starting program

## 

*Figure 1*

* + - * 1. **Button Asign**

|  |  |
| --- | --- |
| **Item** | Asign |
| **Purpose** | Allows the user to assign a value to a variable |
| **Input** | Click on mouse |
| **Output** | The drag and drop mode command appears |
| **Error Handling** | No errors |
| **Reference** | Figura 1, button with the name: Asign |

**3.1.1.1.1.1 TextBox Assign**



*Figure 2*

|  |  |
| --- | --- |
| **Item** | TextBox |
| **Purpose** | Allows the user to assign a value to a variable |
| **Input** | Enter |
| **Output** | Assign the value |
| **Error Handling** | If input doesn’t have a correct format an error message will display |
| **Reference** | The textbox inside |

* + - * 1. **Button Test**

|  |  |
| --- | --- |
| **Item** | Test |
| **Purpose** | Allows the user to choose the true option when there are several possible variants for the compiler |
| **Input** | Click on mouse |
| **Output** | The drag and drop mode command appears |
| **Error Handling** | No errors |
| **Reference** | Figura 1, button with the name: Test |

**3.1.1.1.2.1 TextBox Test**



*Figure 3*

|  |  |
| --- | --- |
| **Item** | TextBox |
| **Purpose** | Allows the user to put a condition for a ambiguous situation |
| **Input** | Enter |
| **Output** | Choose the way to follow the program |
| **Error Handling** | If input doesn’t have a correct format an error message will display |
| **Reference** | The textbox inside |

* + - * 1. **Button Read**

|  |  |  |
| --- | --- | --- |
| **Item** | Read | |
| **Purpose** | allows the user to read a variable from the keyboard | |
| **Input** | | Click on mouse |
| **Output** | | The drag and drop mode command appears |
| **Error Handling** | | No errors |
| **Reference** | | Figura 1, button with the name: Read |

**3.1.1.1.3.1 TextBox Read**



*Figure 4*

|  |  |
| --- | --- |
| **Item** | TextBox |
| **Purpose** | Allows the user to read a variable from the keyboard |
| **Input** | Enter |
| **Output** | Add a new variable |
| **Error Handling** | If input doesn’t have a correct format an error message will display |
| **Reference** | The textbox inside |

* + - * 1. **Button Write**

|  |  |
| --- | --- |
| **Item** | Write |
| **Purpose** | Allow the user to display a value of a variable on the terminal |
| **Input** | Click on mouse |
| **Output** | The drag and drop mode command appears |
| **Error Handling** | No errors |
| **Reference** | Figura 1, button with the name: Write |

**3.1.1.1.5 TextBox Write**



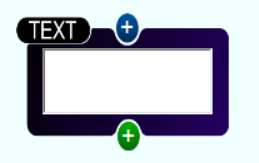
*Figure 5*

|  |  |
| --- | --- |
| **Item** | TextBox |
| **Purpose** | Allow the user to display a value of a variable on the terminal |
| **Input** | Enter |
| **Output** | Display a variable’s value |
| **Error Handling** | If input doesn’t have a correct format an error message will display |
| **Reference** | The textbox inside |

**3.1.1.1.6 Button Text**

|  |  |
| --- | --- |
| **Item** | Text |
| **Purpose** | Allow the user to display a string, a message or a value on the terminal |
| **Input** | Click on mouse |
| **Output** | The drag and drop mode command appears |
| **Error Handling** | No errors |
| **Reference** | Figura 1, button with the name: Text |

**3.1.1.1.6.1 TextBox Text**



*Figure 6*

|  |  |
| --- | --- |
| **Item** | TextBox |
| **Purpose** | Allow the user to display a string, a message or a value on the terminal |
| **Input** | Enter |
| **Output** | Display a string |
| **Error Handling** | If input doesn’t have a correct format an error message will display |
| **Reference** | The textbox inside |

**3.1.1.1.7 Button Delete**

|  |  |
| --- | --- |
| **Item** | Delete |
| **Purpose** | Allow the user to delete a progress and to restart the project |
| **Input** | Click on mouse |
| **Output** | Delete every item |
| **Error Handling** | No errors |
| **Reference** | Figura 1, red button with the name: Delete |

**3.1.1.1.8 Button Run**

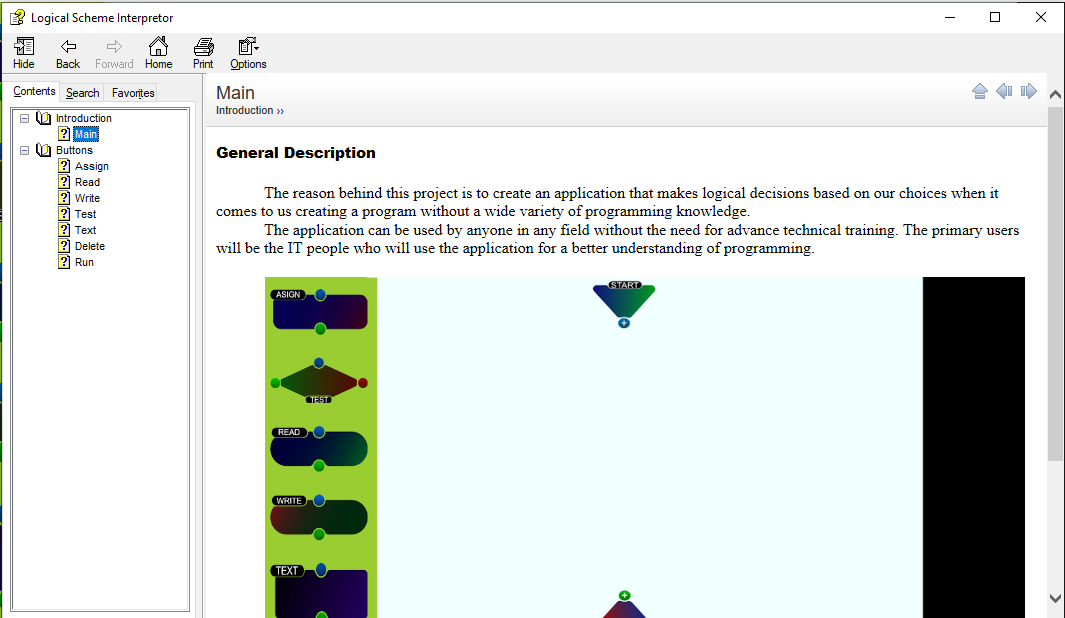
|  |  |
| --- | --- |
| **Item** | Run |
| **Purpose** | Allow the user to run the created program |
| **Input** | Click on mouse |
| **Output** | Run on terminal |
| **Error Handling** | If the structure is incorrect or on a textbox was not pressed the Enter to save the content, the program will display an error message |
| **Reference** | Figure 1, green button with the name: Run |

## 3.1.1.1.9 About  Help



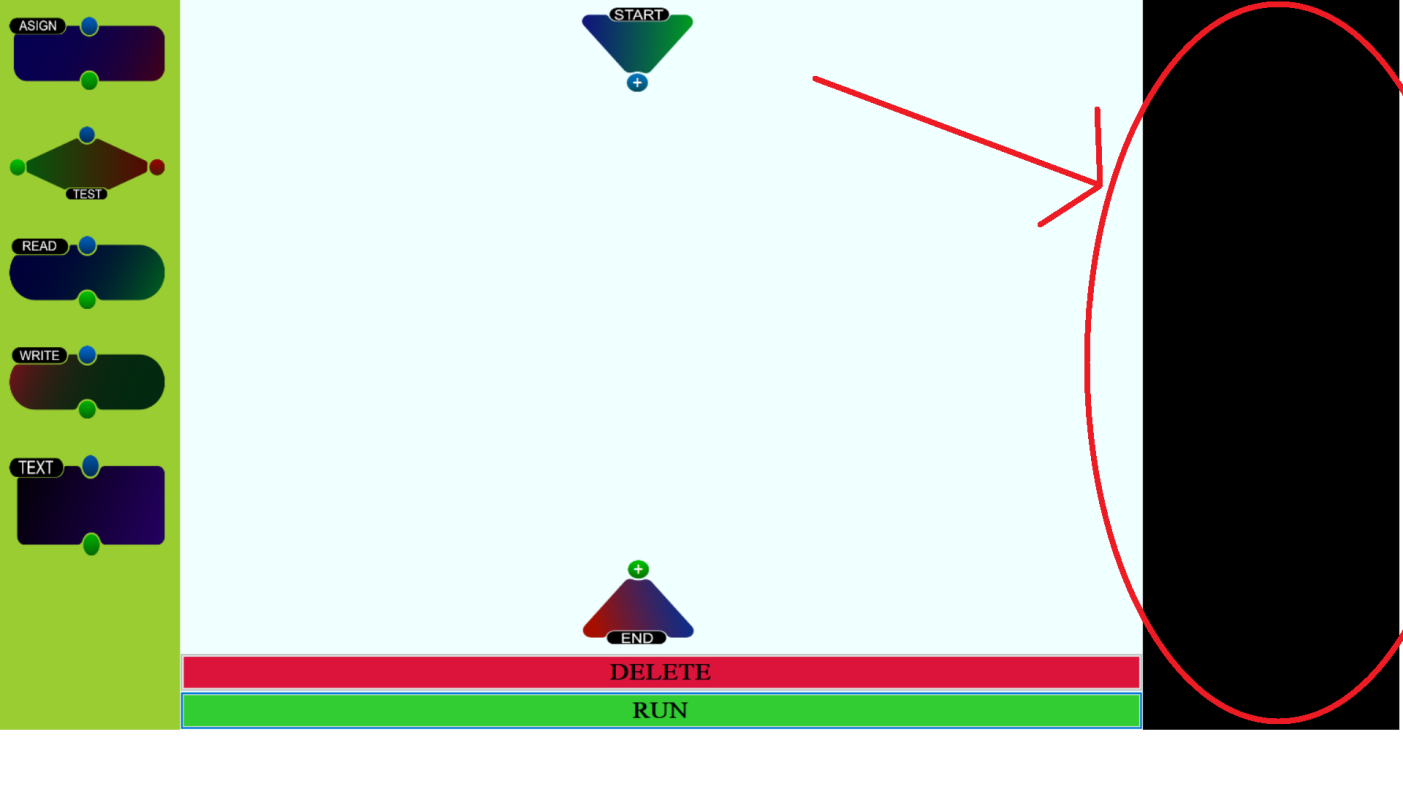
*Figure 7*

|  |  |
| --- | --- |
| **Item** | Help |
| **Purpose** | Allows the user to open a help that will help them understand and use the program |
| **Input** | Click on mouse |
| **Output** | Opens a chm file |
| **Error Handling** | No errors |
| **Reference** | Figure 7 |



## *Figure 8*

***3.1.1.1.10 Terminal***



*Figure 9*

|  |  |
| --- | --- |
| **Item** | Terminal |
| **Purpose** | Displays the output |
| **Input** | None |
| **Output** | Itself |
| **Error Handling** | No errors |
| **Reference** | The black list from the right |

* + 1. **Hardware interfaces**
* Windows

## Software interfaces

As an operating system we used Windows due to its mass use and ease of use.

We worked in Microsoft Visual Studio using the Windows Forms App in the C # language because the user interface is easy to create and implement.

## Structure

* LogicalSchemeInterpretor
* LogicalSchemeManager
* TestUnit

## Logical Scheme Interpretor

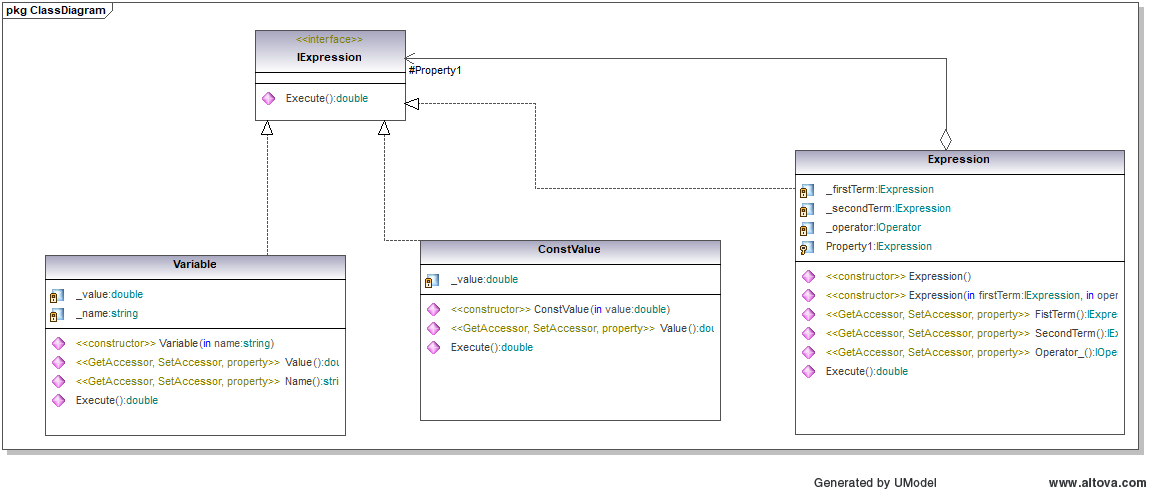
## This module represents the frontend of the application. It manages all the interactions with the client.

## Logical Scheme Manager

## This module is the backend of the application. It manages all the interactions between commands and the execution of the final created program.

**3.2.2.1 Composite**

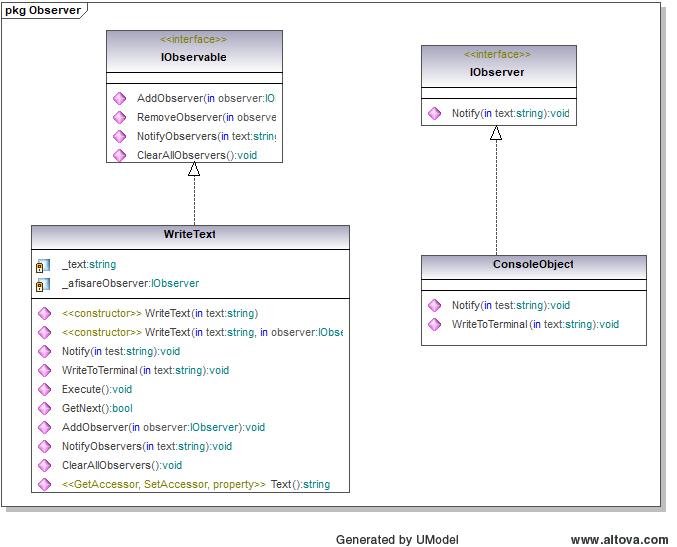
Composite design pattern was used to create a tree of complex expressions, each having the possibility to integrate another complex expression. It has the advantage to execute all the expressions without considering the order of the instructions.



*Figure 10*

**3.2.2.2 Observer**

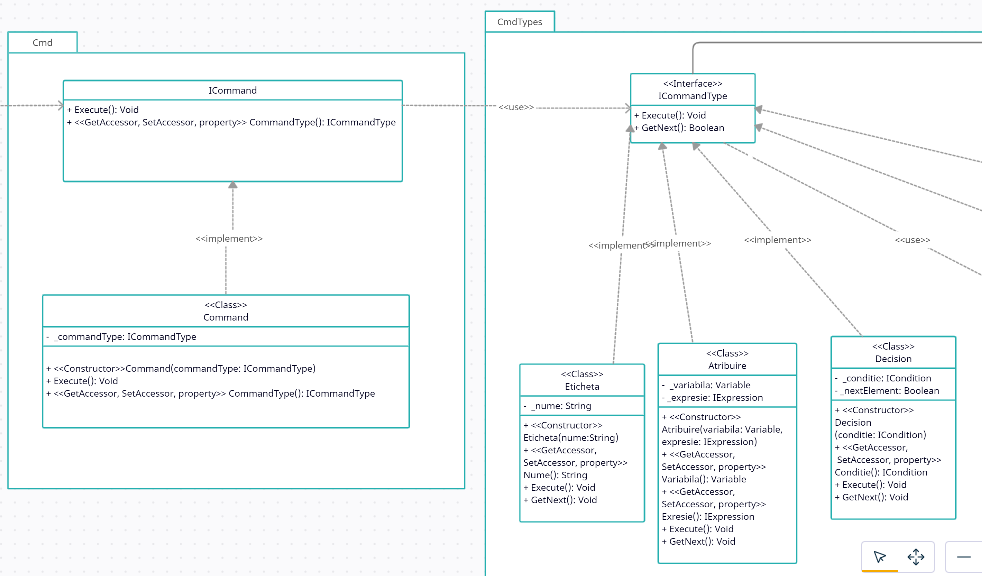
The Observer design pattern was used expecially for objects that can write to an instance of a class that implements IObserver. When a write command needs to write to a terminal, it sends a Notify to the specific terminal. All Write command types are implementing IObservable.



*Figure 11*

**3.2.2.3 Bridge**

Bridge pattern was used to split a command class into smaller related classes, separating the abstraction, which is ICommand, from the implementation, ICommandType.

*Figure 12*

## Test Unit

## In this module all the units of the program are tested.

## 

*Figure 12*

## Performance Requirements

The application must work on any computer running Windows.

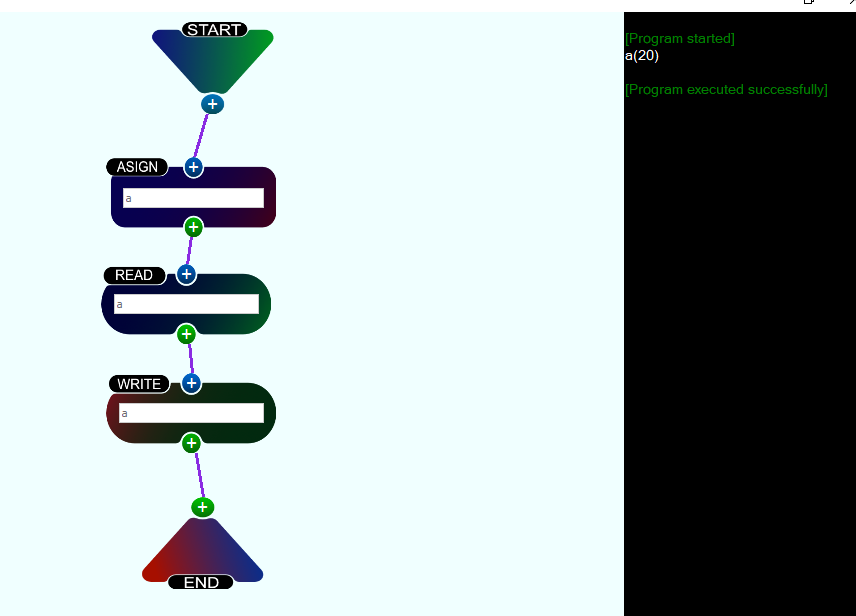
## Attributes

* **Usability:** The program can be used by a large amount of users.
* **Educationally:** The purpose of the application is pure educational, offering a visual way to learn programming concepts.

## 4. Verification

## For the verification of the application, we used some algorithms that assure the good functionality of the application.

**4.1**   **Read and write to the terminal verification**



*Figure 13*

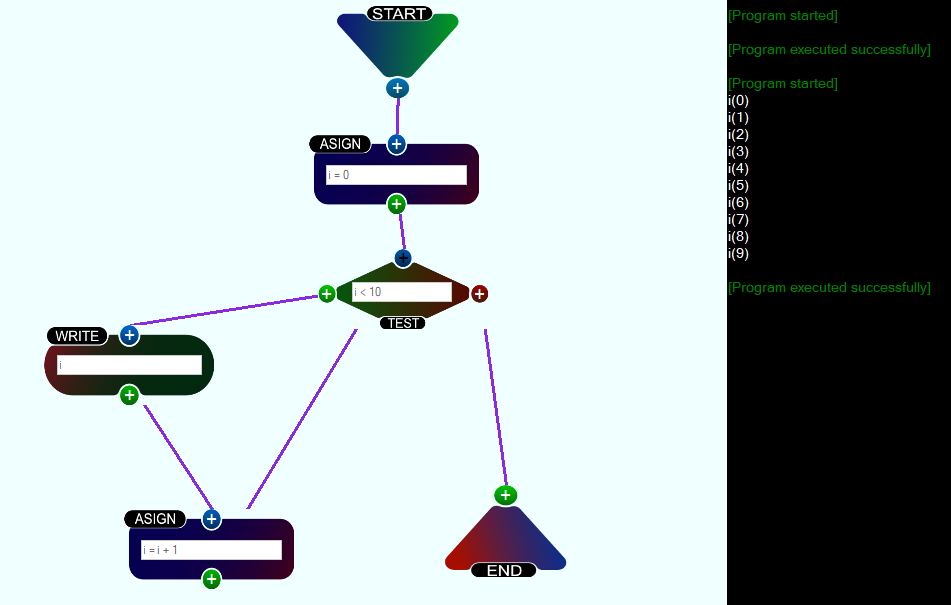
In the presented example at the figure 13, there was considered the following program translated into Control objects on the interface:

ASSIGN: Let there be a variable called “a”

READ: Read the variable from the terminal (in the example, it was introduced the value 20)

WRITE: Write the variable and it’s value to the terminal (in the example, the result cand be observed in the right part of the image)

**4.2**   **Simple loop verification**



*Figure 14*

In the presented example at the figure 14, there was considered the following program translated into Control objects on the interface:

ASSIGN1: Let there be a variable called “i” with the initial value of 0.

TEST: If the condition (i < 10) is true, the next intructions will be those on the left side, else, those on the right side.

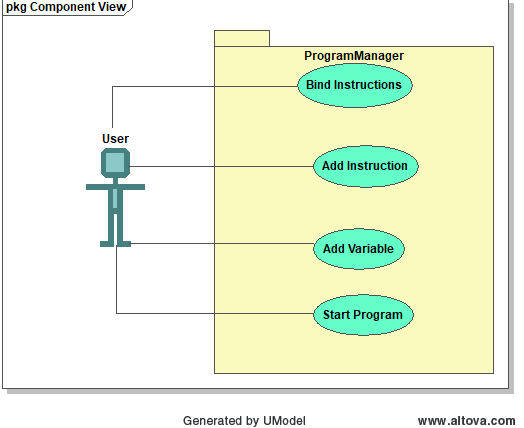
WRITE: Write the value of i.

ASSIGN2: Increment the value of i.

## 5 UML diagrams

## This section contains some the diagrams that proved useful to the process of creating the application.

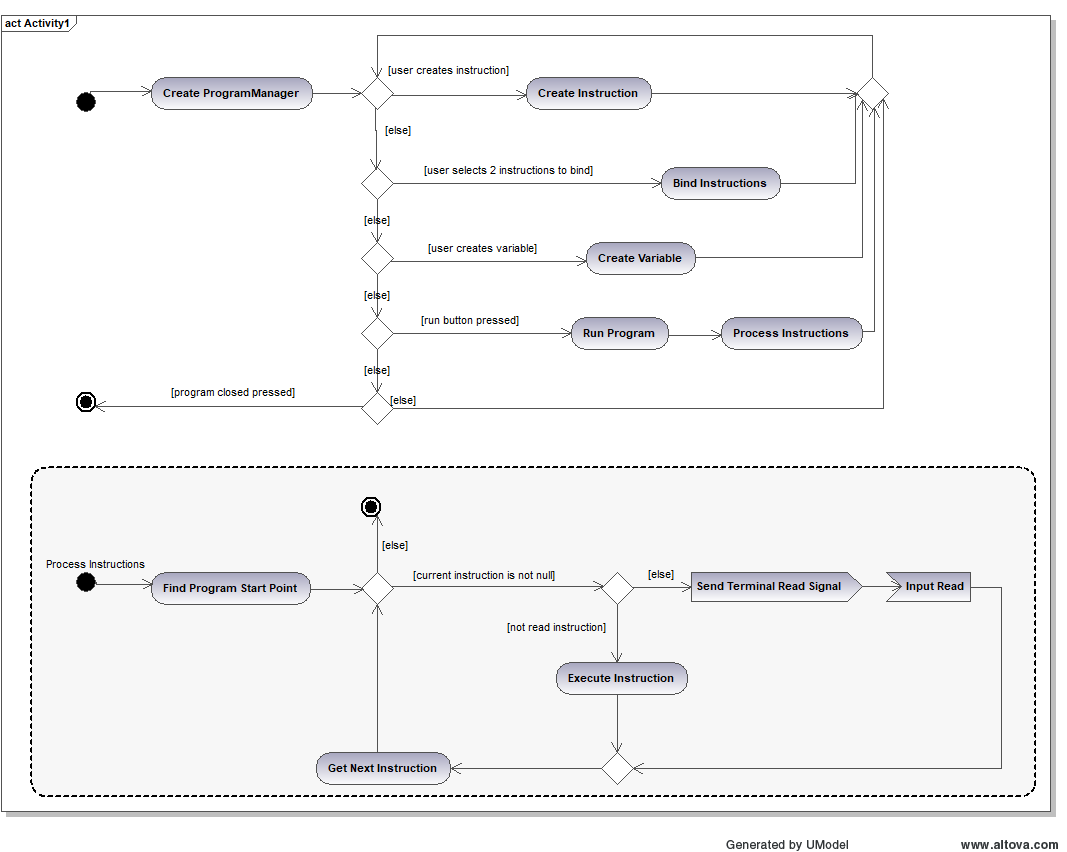
## 5.1 Use case diagram



*Figure 15*

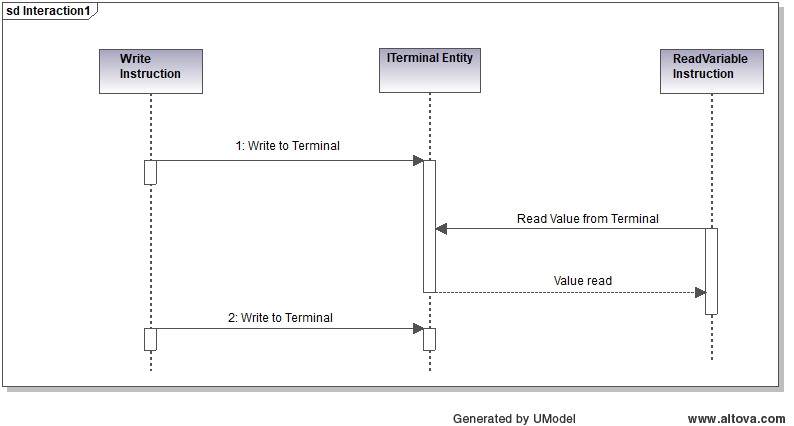
\

## 5.2 Activity diagram



*Figure 16*

**5.3** **Sequence diagram**



*Figure 17*

This sqcuence diagram describes the communication of the Read and Write commands with an abstract ITerminalEntity.