Technical University "Gheorghe Asachi" Iasi

Faculty of Computer Science

Computers and information technology

Year III

**Software Engineering Project**

**NaNta Editor**

**IDE C / C++**

Prepared by:

Damian Gabriel-Mihai

Tutuianu Robert-Constantin

Miron Alexandru

Florea Alexandru-Daniel

Group: 1308B

Version 1.0 approved

25.05.2022

**Table of Contents**

**1.Introduction**

1.1 Purpose

1.2 Document Conventions

1.3 Intended Audience and Reading Suggestions

1.4 Product Scope

1.5 References

**2.Overall Description**

2.1 Product Perspective

2.2 Product Functions

2.3 User Classes and Characteristics

2.4 Operating Environment

2.5 Design and Implementation Constraints

2.6 User Documentation

**3. External Interface Requirements**

3.1 User Interfaces

3.2 Hardware Interfaces

3.3 Software Interfaces

3.4 Communications Interfaces

**4. System Features**

4.1 Open Project

4.2 New Project

4.3 Close Project

4.4 Add File

4.5 Delete File

4.6 Undo / Redo

4.7 Build

4.8 Run

**5. Other Nonfunctional Requirements**

5.1 Performance Requirements

5.2 Safety Requirements

5.3 Security Requirements

5.4 Software Quality Attributes

5.5 Business Rules

**1.Introduction**

1.1 Purpose

An integrated development environment (IDE) is a [software application](https://en.wikipedia.org/wiki/Application_software) that provides comprehensive facilities to [computer programmers](https://en.wikipedia.org/wiki/Computer_programmer) for [software development](https://en.wikipedia.org/wiki/Software_development). An IDE normally consists of at least a [source code editor](https://en.wikipedia.org/wiki/Source_code_editor), [build automation](https://en.wikipedia.org/wiki/Build_automation) tools and a [debugger](https://en.wikipedia.org/wiki/Debugger).

1.2 Intended Audience and Reading Suggestions

This document is intended for any developer, project manager or tester whose area of interest involves modeling a C / C++ project.

We recommend that this document be read sequentially in the following chapter order: Introduction, Overall Description, External Interface Requirements, System Features, Other Nonfunctional Requirements.

1.3 Product Scope

The purpose of this software product is to facilitate working with C / C++ projects, editing C / C++ source code, and to provide a scalable application structure in terms of adding new functionality.

1.4 References

Tokenizer references:

<https://docs.microsoft.com/en-us/cpp/cpp/character-sets?view=msvc-170>

<https://en.cppreference.com/w/cpp/keyword>

<https://docs.microsoft.com/en-us/cpp/cpp/punctuators-cpp?view=msvc-170>

<https://docs.microsoft.com/en-us/cpp/cpp/identifiers-cpp?view=msvc-170>

<https://en.cppreference.com/w/cpp/language/integer_literal>

<https://en.cppreference.com/w/cpp/language/floating_literal>

Model-View-Presenter References:

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

<https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/>

<https://anshul-vyas380.medium.com/model-view-presenter-b7ece803203c>

Memento Design Pattern:

<https://refactoring.guru/design-patterns/memento>

Command Design Pattern:

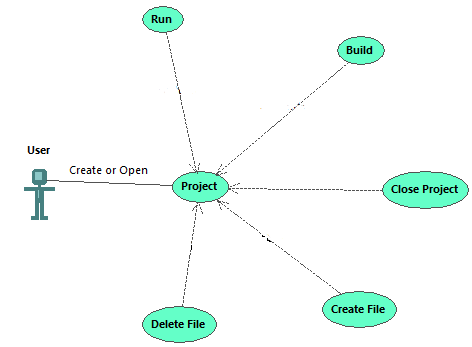
<https://refactoring.guru/design-patterns/command>

**2.Overall Description**

2.1 Product Perspective

This project falls into the category of a self-contained product.

The diagram below illustrates the basic functionalities of the project up to the time of presentation.



2.2 Product Functions

The presented application offers the following functionalities:

* Create a new project
* Open an old project (XML format based configuration)
* Open files in a tab-based layout
* Create a new file
* Undo and redo editor actions
* Auto-saving the project and editor files
* Build & run C / C++ based project
* Syntax highlighting for source code

2.3 User Classes and Characteristics

The project is based on the model-view-presenter template. The classes are separated into the following modules, each of which is encapsulated in a separate DLL or EXE project:

*Model:*

* FormModel
  + Implements the IModel interface

*View:*

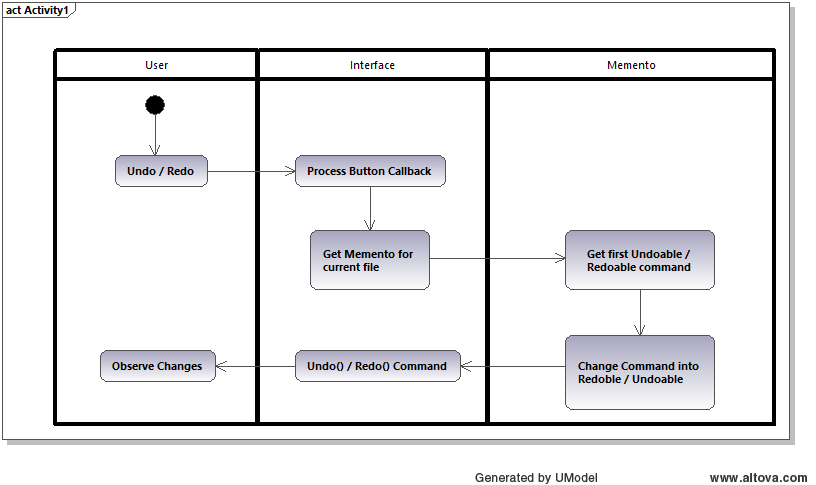
* FormView
  + Implements the IView interface
  + Manages forms and dialogues
  + Connects form events and presenter
* NewFileForm
  + A special dialog that allows the user to create a new project file
* NewProjectForm
  + A special dialog that prompts the user for information about the new project
* SplashScreenForm
  + A special form that displays a startup screen for the application
* WorkspaceForm
  + The main editor interface that allows for editing C / C++ files
* WriteCommand
  + Encapsulates a write action in an object
* DeleteCommand
  + Encapsulates a delete action in an object

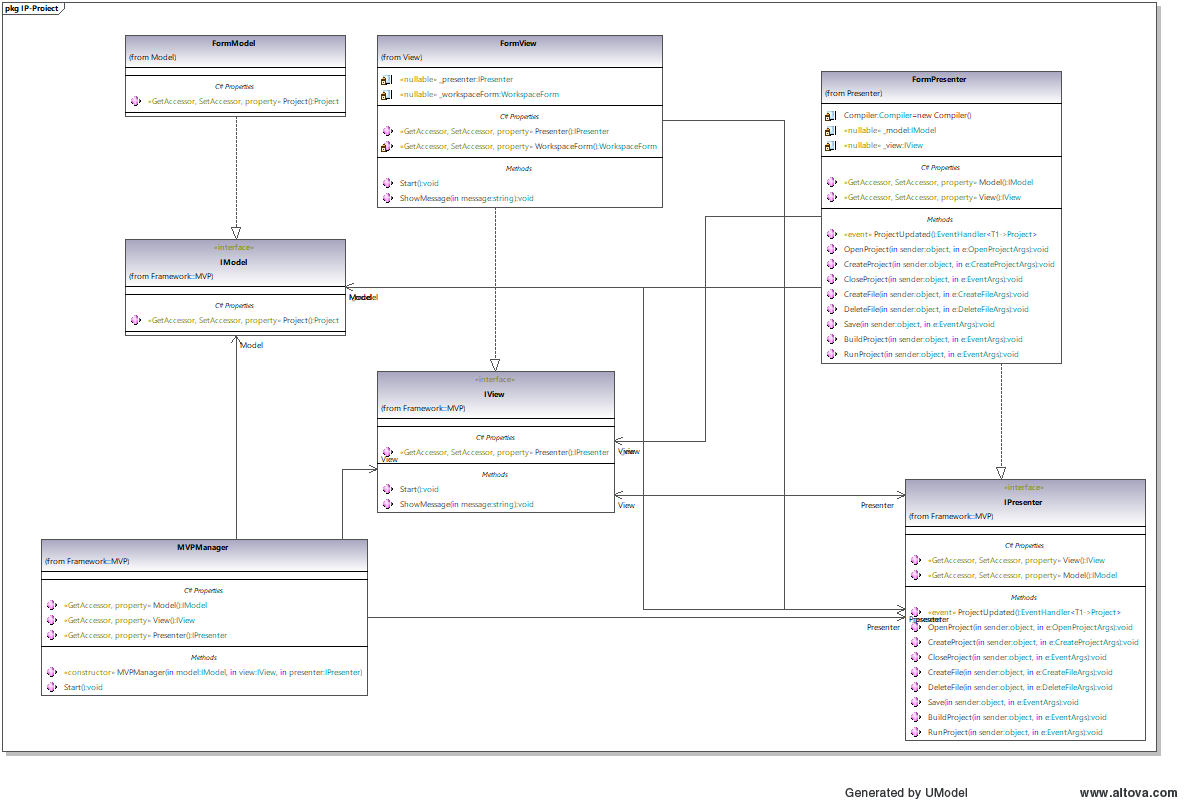
*Presenter:*

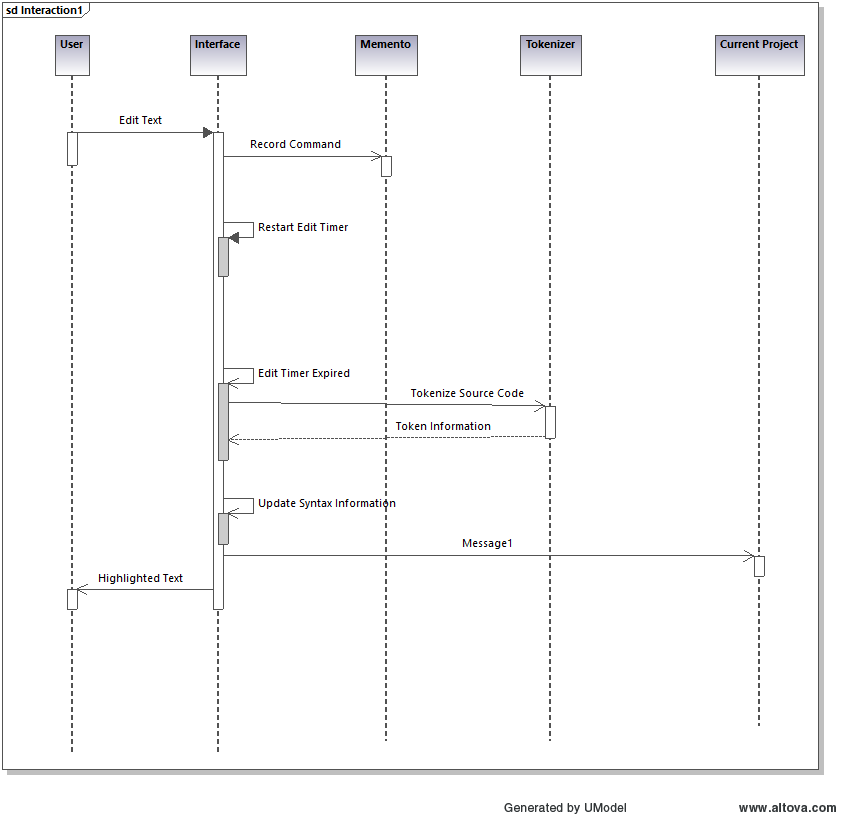
* FormPresenter
  + Contains presentation and business logic

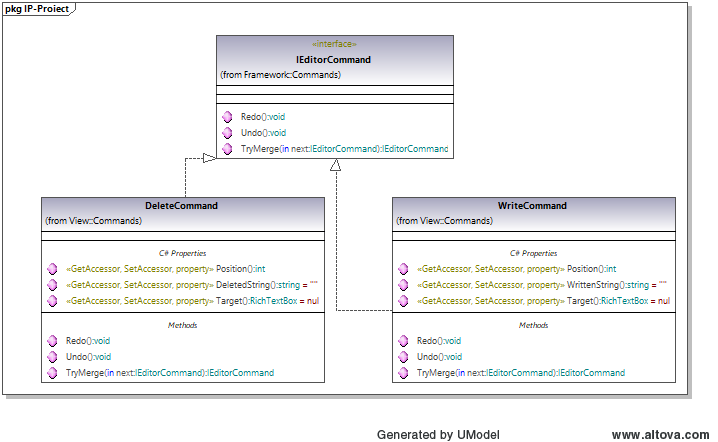
*Framework:*

* IModel
  + The interface for the model in the MVP architecture
* IView
  + The interface for the view in the MVP architecture
* IPresenter
  + The interface for the presenter in the MVP architecture
* Compiler
  + Compiles a C / C++ project
* Tokenizer
  + Parses C / C++ source code into tokens for the purpose of syntax highlighting
* Memento
  + Stores editor commands for a file and allows retrieving command that can be done / undone
* IEditor Command
  + The interface for all commands
  + Defines three methods: Undo, Redo, TryMerge
* Other helper classes, such as:
  + MVPManager
  + Token
  + CppStuff
  + CompileOptions









2.4 Operating Environment

This product is intended to run on the Windows operating system. To run the application, .NET 6 must be installed on the user’s system.

2.5 Design and Implementation Constraints

Being only a demo product, the application is limited in terms of features added. It only aims to demonstrate that the built structure is able to support many more future functionalities.

2.6 User Documentation

The application is provided with a help menu through which the user can learn how to use each feature and what its characteristics are. The menu is available in About > Help.

The help menu covers the next topics in detail (instructions with screen shots):

Introduction:

-Welcome

-What’s new

Getting Started;

-System requirements

-Create New Project

-Open Project

-Create File

-Delete File

-Syntax Highlighting

-Run/Build a program

-Undo & Redo

-Copy & Paste

-Software requirements

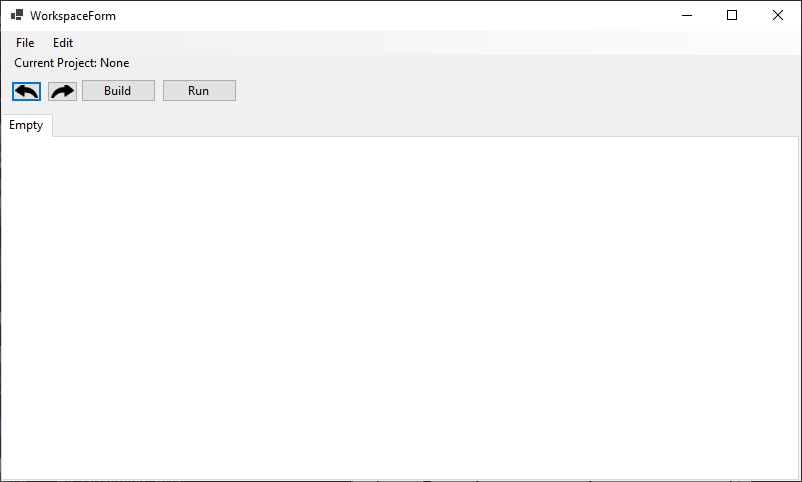
**3. External Interface Requirements**

3.1 User Interfaces

Loading Screen:



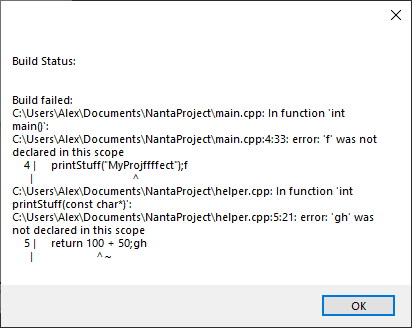
Empty main user interface:



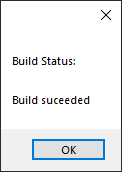
Main user interface with project opened:



Build Failed:



Build succeeded:



3.2 Hardware Interfaces

The application aims to model only software products that will run on the current computing machine, so it will not come into contact with any hardware interface.

3.3 Software Interfaces

This application has no external dependencies like any other software product or database, because it is just a demo that wants to highlight only the basic functionality for such a product.

In the future, you can add features such as login / signup to identify each user, the database to retain any preferences saved by the user or even files in the cloud.

3.4 Communications Interfaces

No communication interface is used in this project, because the functionalities present so far in the application do not require communication with other applications.

**4. System Features**

4.1 Open Project

By pressing the “Open Project” button, a window is shown from which you can select a .xml file.

Using that file a project will open with settings and files set.

4.2 New Project

| public void CreateProject(object? sender, CreateProjectArgs e)  {  Model.Project?.SaveXML();  Model.Project = new Project()  {  ProjectTitle = e.Name,  FilePath = Path.Combine(e.FolderPath, e.Name + ".xml")  };  Model.Project.SaveXML();  Compiler.WorkingDirectory = Path.GetDirectoryName(Model.Project.FilePath) ?? "";  ProjectUpdated?.Invoke(this, Model.Project);  } |
| --- |

By pressing the “New Project” button a new project will be initialized. Project’s state will be saved in a .xml file. This file can be used to open the same project in the same state next time you need it.

4.3 Close Project

| public void CloseProject(object? sender, EventArgs e)  {  Model.Project?.SaveXML();  Model.Project = null;  ProjectUpdated?.Invoke(this, Model.Project);  } |
| --- |

When the user closes the project, the program is saving the project's state in the .xml file, then returns to the initial state where you have no opened project.

4.4 Add File

| public void CreateFile(object? sender, CreateFileArgs e)  {  if (Model.Project == null)  return;  Model.Project.AddFileToProject(e.Name, e.FolderPath);    ProjectUpdated?.Invoke(this, Model.Project);  } |
| --- |

By pressing the “Add File” button a new file will be created in the target directory and then added to the project.

4.5 Delete File

| public void DeleteFile(object? sender, DeleteFileArgs e)  {  if (Model.Project == null || e.Instance == null)  return;  Model.Project.RemoveFileFromProject(e.Instance);  ProjectUpdated?.Invoke(this, Model.Project);  } |
| --- |

Pressing “Delete File”, the current opened file will be removed from the project and then deleted from disk.

4.6 Undo / Redo

| public void Redo()  {  if (Target == null)  return;  Target.Select(Position, 0);  Target.SelectedText = WrittenString;  }  public void Undo()  {  if (Target == null)  return;  Target.Select(Position, WrittenString.Length);  Target.SelectedText = "";  } |
| --- |

When the user edits a file, the file state is added in a stack-like structure used to store all actions. When pressing “Undo” the last state is popped from the stack, is added to the redo stack and then the file is updated with that state. When “Redo” is pressed, the same story applies but the popped value is added to the undo stack.

4.7 Build

| public bool BuildFiles(CompileOptions options)  {  string filesToCompile = string.Join(" " ,options.AbsoluteFilePaths);  string compileArguments = $"{options.CommandLineArguments} -o {options.OutputName}";  var process = new Process  {  StartInfo = new ProcessStartInfo  { |
| --- |

| WorkingDirectory = WorkingDirectory != "" ? WorkingDirectory : Directory.GetCurrentDirectory(),  FileName = COMPILER\_NAME,  Arguments = $"{filesToCompile} {compileArguments}",  RedirectStandardError = true,  CreateNoWindow = true  }  };  try  {  process.Start();  }  catch (Exception e)  {  StatusText = "Failed to run compiler. Exception:\n" + e;  return Success = false;  }  process.WaitForExit();  string errors = process.StandardError.ReadToEnd();  StatusText = errors != "" ? "Build failed: \n" + errors : "Build succeeded";  return Success = !errors.Contains("error");  } |
| --- |

Pressing the “Build” button, all files are compiled and linked together.An executable will be created that can be run directly. Finally a message box is shown with the state of the operation.

4.8 Run

| public bool RunFileInTerminal(CompileOptions options)  {  string comandaRulare = $"{options.OutputName}.exe";  if (!File.Exists(comandaRulare) && !BuildFiles(options))  {  return Success = false;  }  var process = new Process  {  StartInfo = new ProcessStartInfo  {  WorkingDirectory = WorkingDirectory != "" ? WorkingDirectory : Directory.GetCurrentDirectory(),  FileName = "CMD.exe",  Arguments = $"/C {comandaRulare} & {WAIT\_TERMINAL}",  }  };  try  {  process.Start();  }  catch (Exception e)  {  StatusText = "Failed to run program. Exception:\n" + e;  return Success = false;  }  StatusText = "Run succeeded";  return Success = true;  } |
| --- |

Pressing the “Run” button, the project will be builded if it is not already and then run.

**5. Other Nonfunctional Requirements**

5.1 Performance Requirements

The application aims for decent performance metrics so that the user experience is smooth. This means avoiding the use of time-consuming operations and optimizing program algorithms where possible. Given that the program demo is fairly simple, this is easily achievable.

5.2 Safety Requirements

The program should be tolerant to user mistakes and errors, as well as being resistant to potential programmer mistakes. This is achieved by the use of exception-free program logic where possible, the handling of exceptions generated by system modules and third party APIs, as well as the use of development options such as turning on all compiler warnings and treating warnings as errors.

5.3 Security Requirements

Since the program is fairly simple in design, it does not deal with sensitive user data, therefore the security requirements are low.

5.4 Software Quality Attributes

The main qualities sought after by the user are ease of use and editing. The application is also open to feature extension due to its careful design.

5.5 Business Rules

Any user that uses the application has free access to each module of the application. There are no constraints in regards to use, and our target is for there to not be any either.