2016

TP-Mini Editor Project

Salsabil Amri

Manel

Mukrram Ur Rahman

Introduction: **MiniTextEditor** is a type of program used for editing plain text. It is developed using different tools and technologies. We followed different conventions also applied various design patterns during the development of this software. **MiniTextEditor** provides all the functionalities, which were required for this project, and it also offers different features, they were not requested in the subject but are just added to make **MiniTextEditor** smarter. All of these components are explained below one by one.

Tools: We used various tools for the development of **MiniTextEditor** project. They are listed below.

- **Eclipse:** Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used as Java IDE[1]. It is a very nice tool for development. We used it for the development of **MiniTextEditor.**
- **Java:** We used Java as a programming language to develop **MiniTextEditor.** We used it because it is easy, object-oriented and also platform-independent.
- **Junit:** We used JUnit for performing Unit testing. We used JUnit for various reasons, mainly because of its simplicity.
- **draw.io:** We draw different UML diagrams for each version of the **MiniTextEditor** project. We used it mainly because of the following reasons.
  - **Simple**
  - **Shareable :** Diagrams drawn in draw.io are shareable. We are a team of three members. We used it mainly because of its shareable feature.
  - **Online**

Versions: We divided the project into three versions. Versioning is based on the features. We added different features in each version. All of these versions, with their artifacts and design decisions are listed below.

First Version: In the first version, we supported following features.

- **Insert Text**
- **Select Text**
- **Copy Text**
- **Cut Text**
- **Delete Text**
- **Paste Text**

For first version, we used command line interface. It enables users to enter different commands to perform different operations. For this version we supported commands as show below.

Operation	Command
Insert	[Ii][:]<<Text to insert.>>
Select	[Ss][:]<<Start>>,<<End>>
Copy	[Cc]

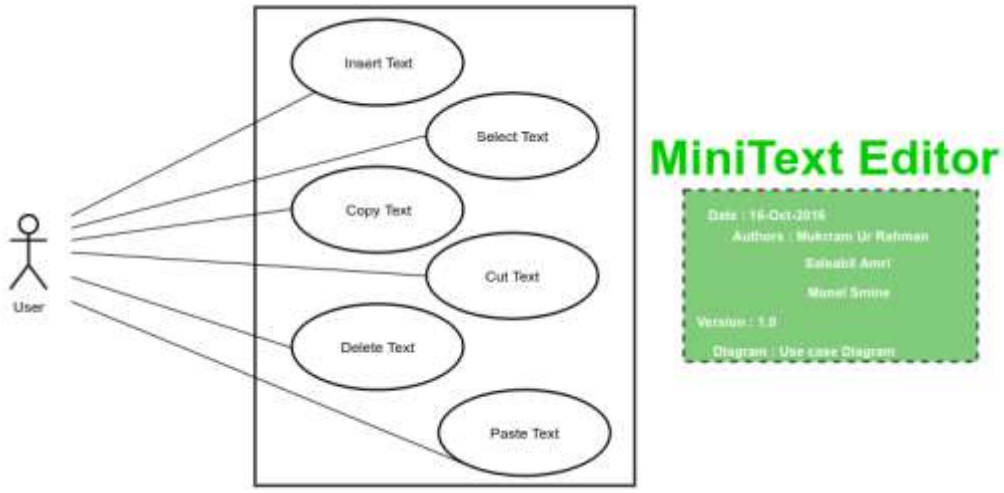
Delete	[Dd]
Cut	[Xx]
Paste	[Pp]

Design Decisions: For the first version, we only used command design pattern.

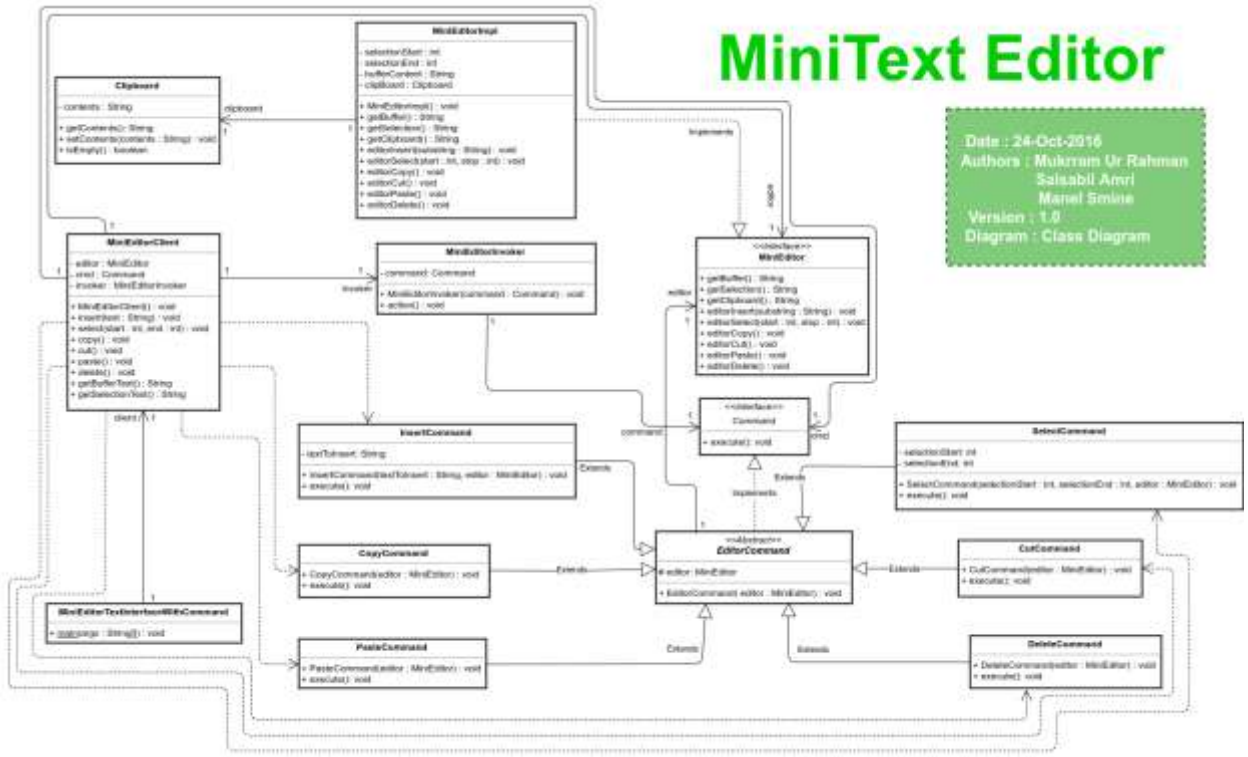
Command Design Pattern: We explained below the components of the design pattern mapped to **MiniTextEditor.**

- **Client:** We have the following client.
  - **MiniEditorClient**
- **Invoker:** We have the following invoker.
  - **MiniEditorInvoker**
- **Command:** It is an interface and we also provide an abstract class named **EditorCommand,** which implements this interface and each concrete command extends this abstract class. We have the following command interface.
  - **Command**
- **Concrete Command:** We have the following concrete commands in our solution.
  - **InsertCommand**
  - **CopyCommand**
  - **CutCommand**
  - **DeleteCommand**
  - **PasteCommand**
  - **SelectCommand**
- **Receiver:** We have the following receiver interface.
  - **MiniEditor**
- **Concrete Receiver:** We have the following concrete receiver in our **MiniTextEditor.**
  - **MiniEditorImpl**

Use case diagram for the first version is attached below.



Class Diagram for the first version is attached below.



Second Version: In the second version, we supported following features.

- **Start Recording**
- **Stop Recording**
- **Replay Recording**

For second version, we also used command line interface. It enables users to enter different commands to perform different operations. For this version we supported additional commands as show below.

Operation	Command
Start Recording	[Ff]
Stop Recording	[Ll]
Replay Recording	[Gg]

Design Decisions: For the second version, we used command design pattern, observer design pattern and also memento design pattern.

Command Design Pattern: We explained below the components of this design pattern mapped to **MiniTextEditor.**

- **Client:** We have the following client.
  - **MiniEditorClient**
- **Invoker:** We have the following invoker.
  - **MiniEditorInvoker**
- **Command:** It is an interface and we also provide an abstract class named **EditorCommand,** which implements this interface and each concrete command extends this abstract class. We have the following command interface.
  - **Command**
- **Concrete Command:** We have the following concrete commands in our solution.
  - **InsertCommand**
  - **CopyCommand**
  - **CutCommand**
  - **DeleteCommand**
  - **PasteCommand**
  - **SelectCommand**
  - **StartRecordCommand**
  - **StopRecordCommand**
  - **ReplayRecordCommand**
- **Receiver:** We have the following receiver interface.
  - **MiniEditor**
- **Concrete Receiver:** We have the following concrete receiver in our **MiniTextEditor.**
  - **MiniEditorImpl**

Memento Design Pattern: We explained below the components of this design pattern mapped to **MiniTextEditor.**

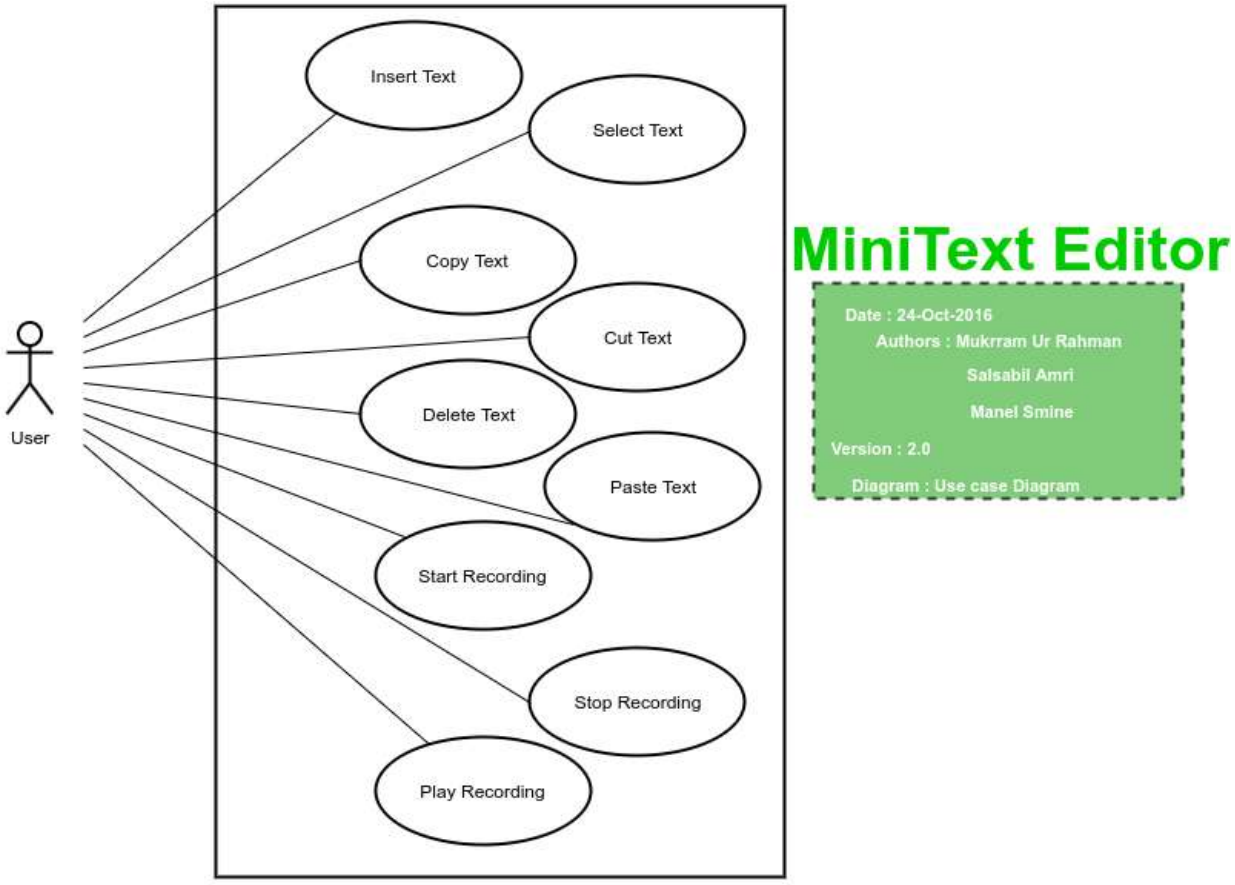
- **Originator:** We have the following originator class.
  - **State**
- **Caretaker:** We have the following caretaker class.
  - **CareTaker**
- **Memento:** We have the following memento class.
  - **Memento**

Observer Design Pattern: We explained below the components of this design pattern mapped to **MiniTextEditor.**

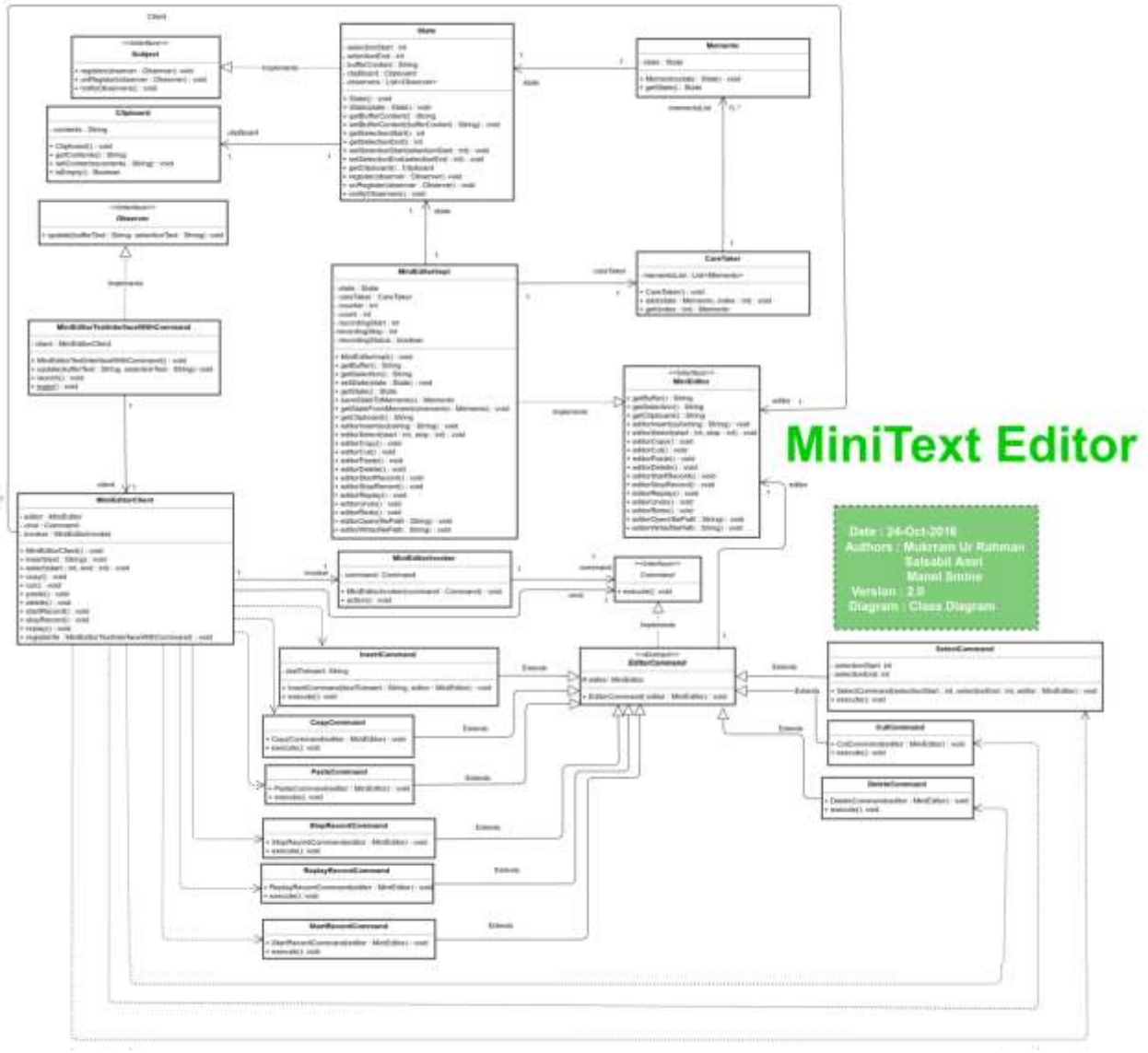
- **Subject:** We have the following subject interface.
  - **Subject**

- **Concrete Subject: State** class has dual role in our system. Concrete observer is listed below.
  - **State**
- **Observer:** We have the following observer interface.
  - **Observer**
- **Concrete Observer:** We have the following concrete observer :
  - **MiniEditorTextInterfaceWithCommand**

Use case diagram for the second version is attached below.



Class Diagram for the second version is attached below.



Third Version: In the third version, we supported following features.

- **Undo**
- **Redo**
- **Open File**
- **Write to File**

For third version, we also used command line interface. It enables users to enter different commands to perform different operations. For this version we supported additional commands as show below.

Operation	Command
Undo	[Uu]
Redo	[Rr]

Open	**[Oo][:]<<File path>>**
Write	**[Ww][:]<<File path>>**

Design Decisions: For the third version, we used command design pattern, observer design pattern and also memento design pattern.

Command Design Pattern: We explained below the components of this design pattern mapped to **MiniTextEditor.**

- **Client:** We have the following client.
  - **MiniEditorClient**
- **Invoker:** We have the following invoker.
  - **MiniEditorInvoker**
- **Command:** It is an interface and we also provide an abstract class named **EditorCommand,** which implements this interface and each concrete command extends this abstract class. We have the following command interface.
  - **Command**
- **Concrete Command:** We have the following concrete commands in our solution.
  - **InsertCommand**
  - **CopyCommand**
  - **CutCommand**
  - **DeleteCommand**
  - **PasteCommand**
  - **SelectCommand**
  - **StartRecordCommand**
  - **StopRecordCommand**
  - **ReplayRecordCommand**
  - **UndoCommand**
  - **RedoCommand**
  - **OpenCommand**
  - **WriteCommand**
- **Receiver:** We have the following receiver interface.
  - **MiniEditor**
- **Concrete Receiver:** We have the following concrete receiver in our **MiniTextEditor.**
  - **MiniEditorImpl**

Memento Design Pattern: We explained below the components of this design pattern mapped to **MiniTextEditor.**

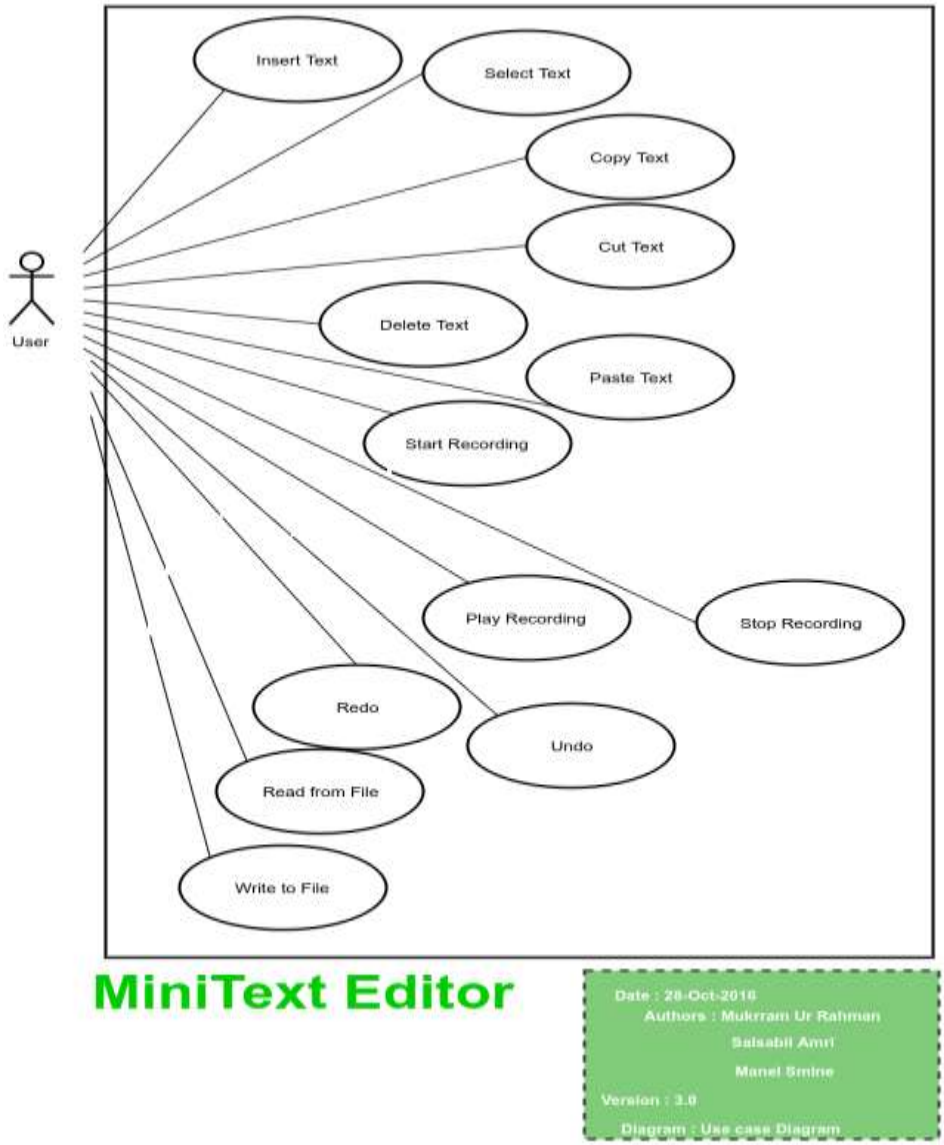
- **Originator:** We have the following originator class.
  - **State**
- **Caretaker:** We have the following caretaker class.
  - **CareTaker**

- **Memento:** We have the following memento class.
    - **Memento**

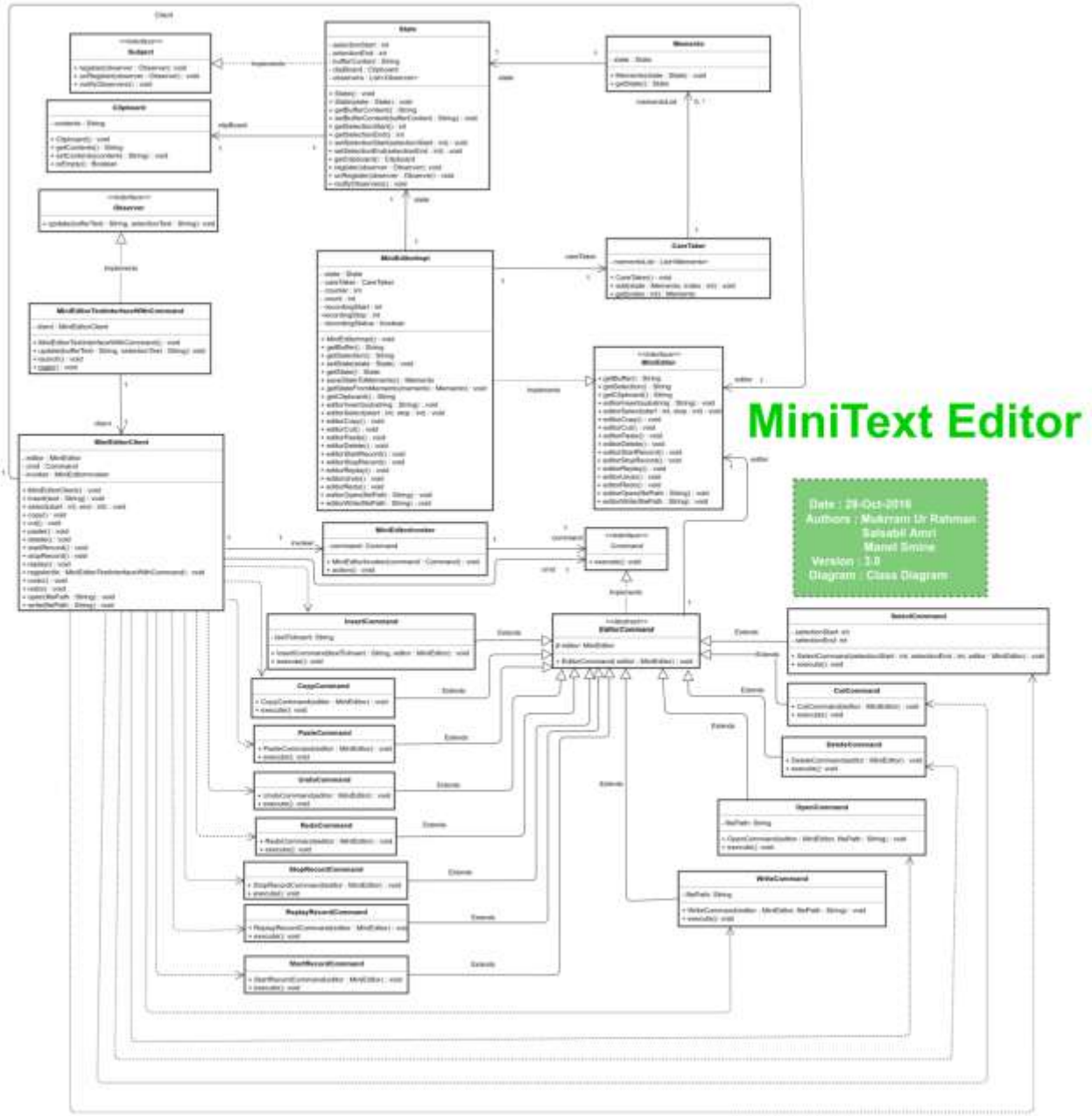
<u>Observer Design Pattern</u>: We explained below the components of this design pattern mapped to **MiniTextEditor.**

- **Subject:** We have the following subject interface.
    - **Subject**
- **Concrete Subject: State** class has dual role in our system. Concrete observer is listed below.
    - **State**
- **Observer:** We have the following observer interface.
    - **Observer**
- **Concrete Observer:** We have the following concrete observer :
    - **MiniEditorTextInterfaceWithCommand**

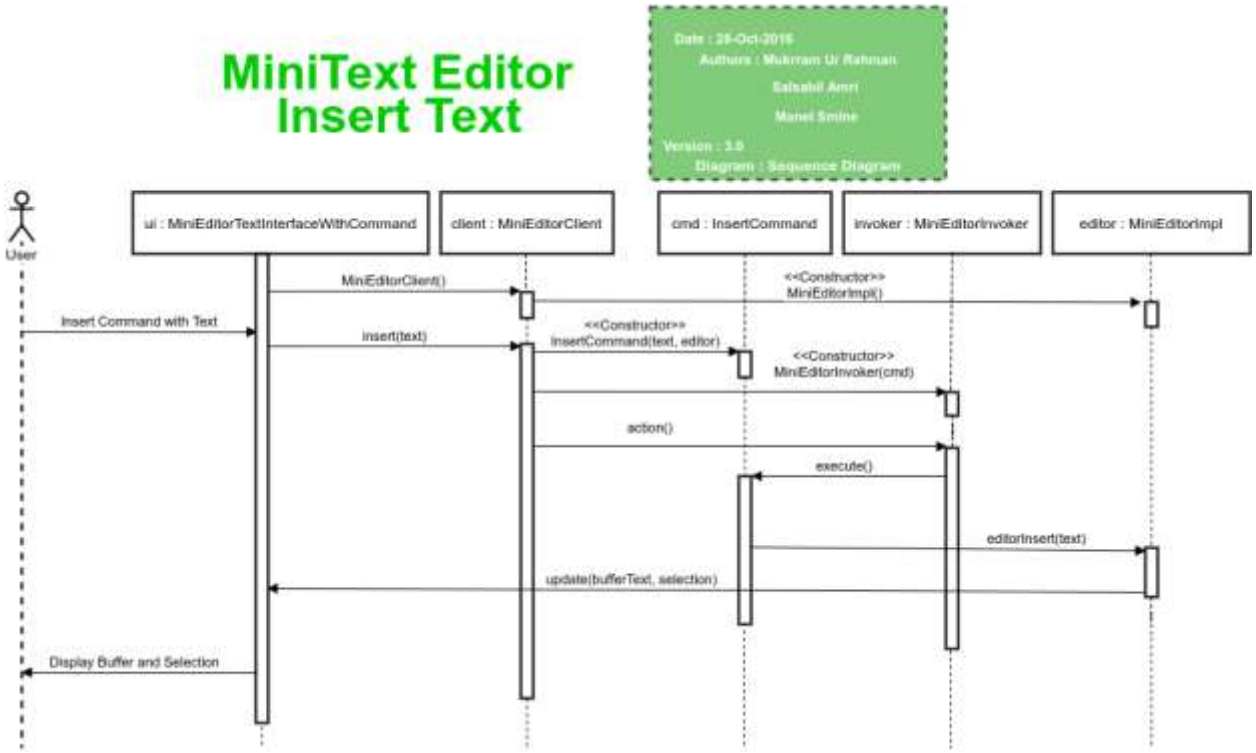
Use case diagram for the third version is attached below.

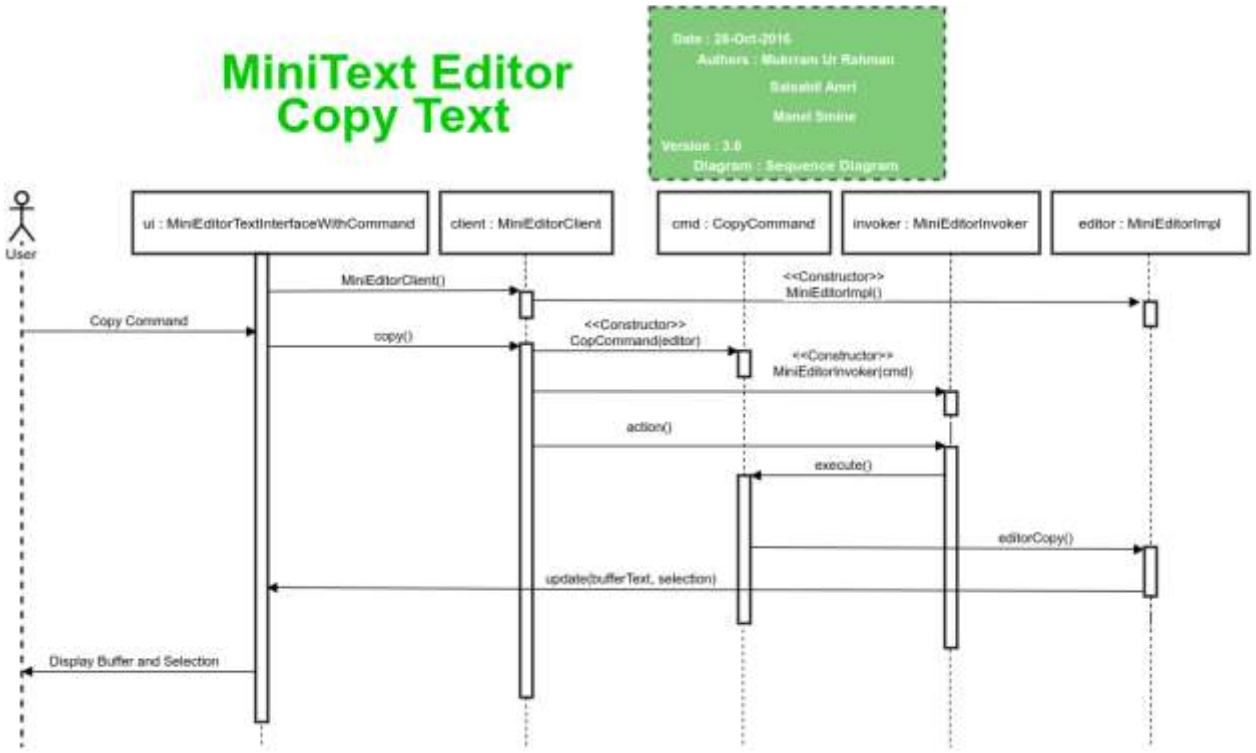


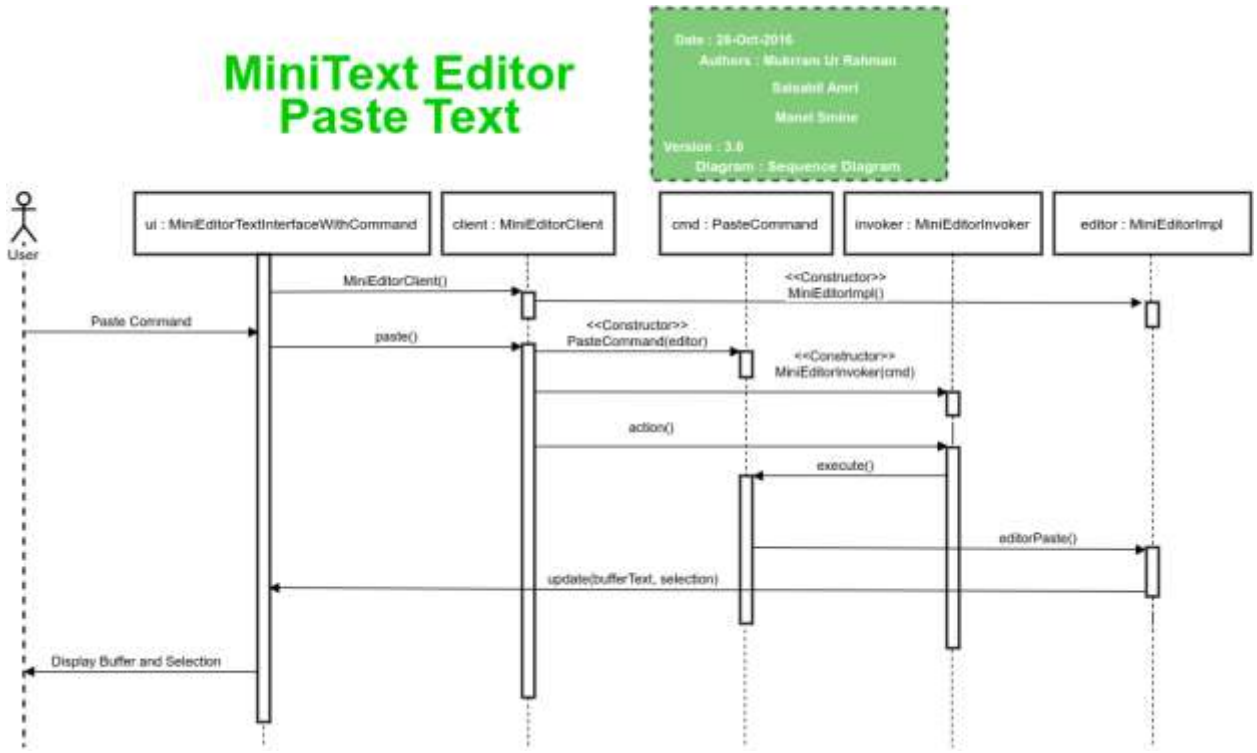
Class Diagram for the third version is attached below.

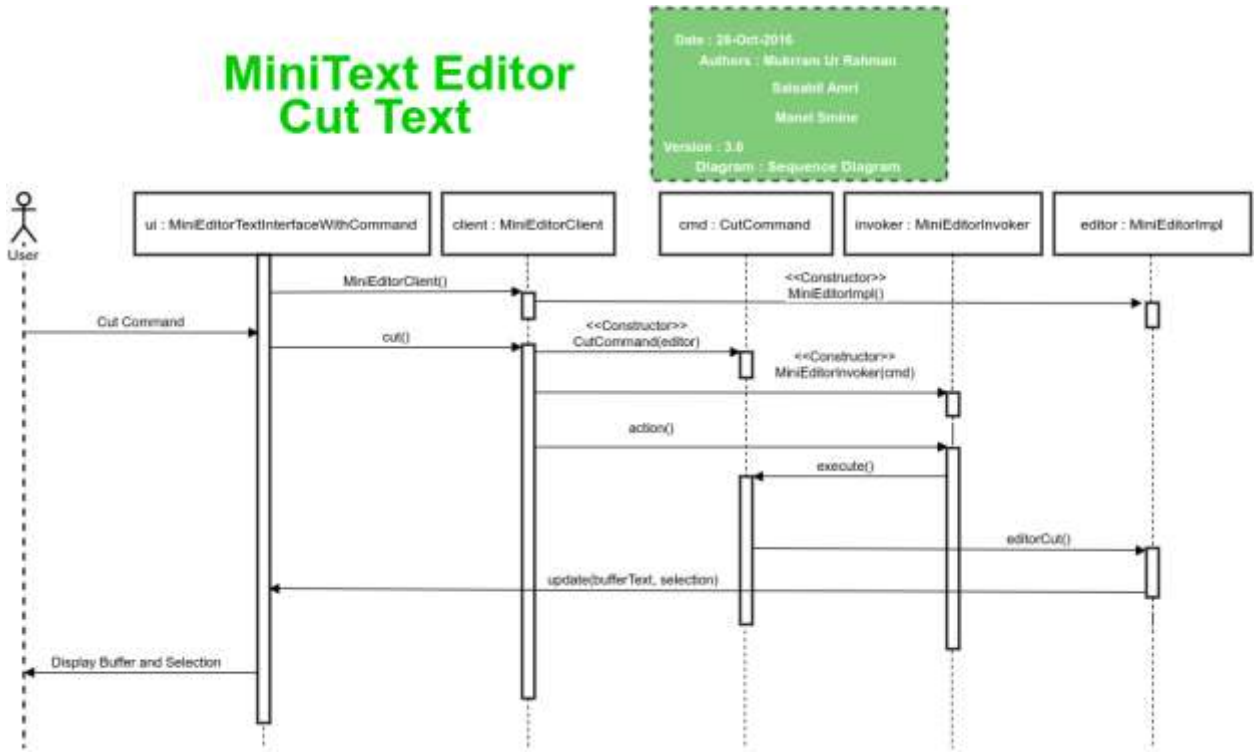


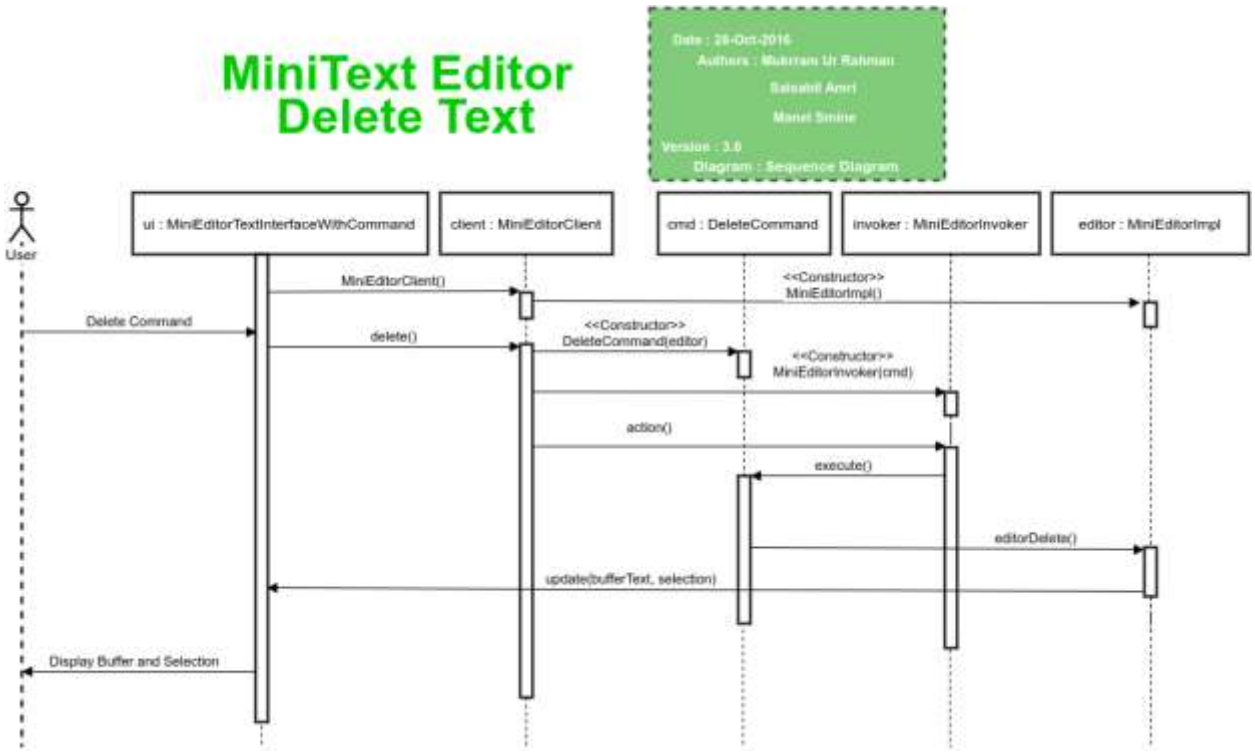
<u>Sequence Diagrams</u>: Sequence diagrams for the **MiniTextEditor** are given below.

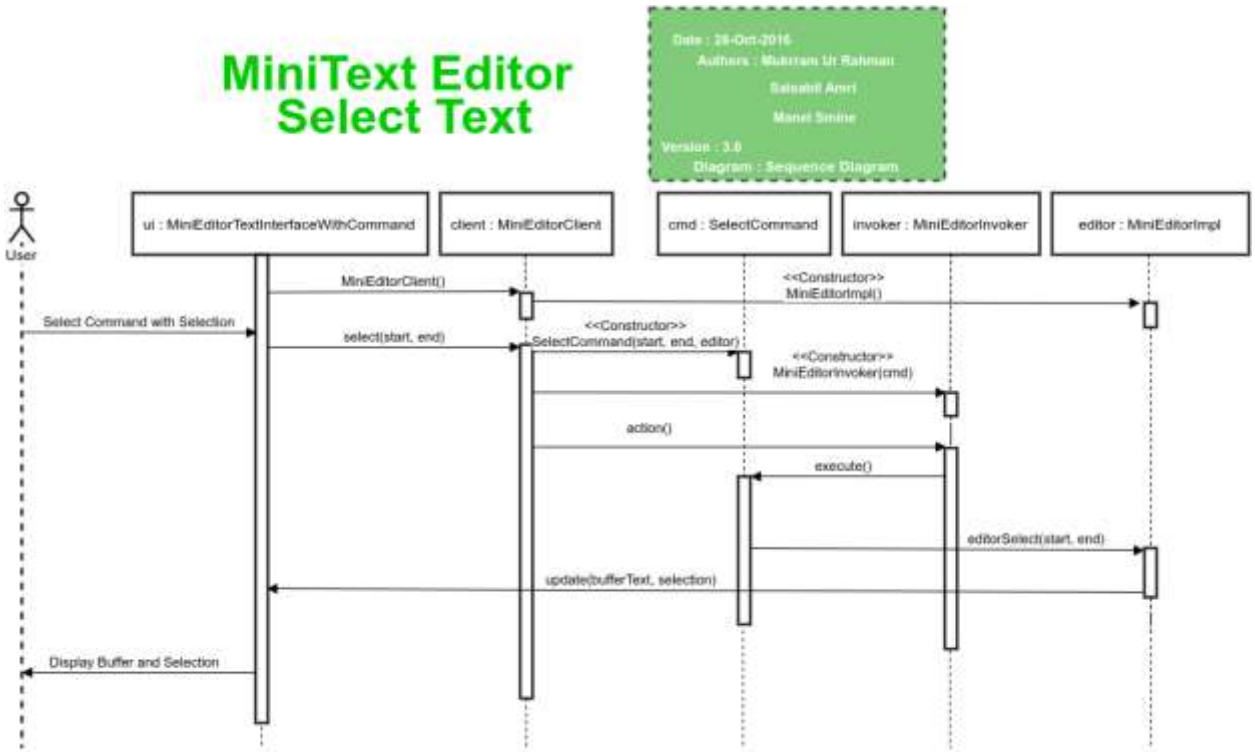

MiniText Editor
Insert Text
Date : 28-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : InsertCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Insert Command with Text
insert(text)
<<Constructor>>
InsertCommand(text, editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorInsert(text)
update(bufferText, selection)
Display Buffer and Selection

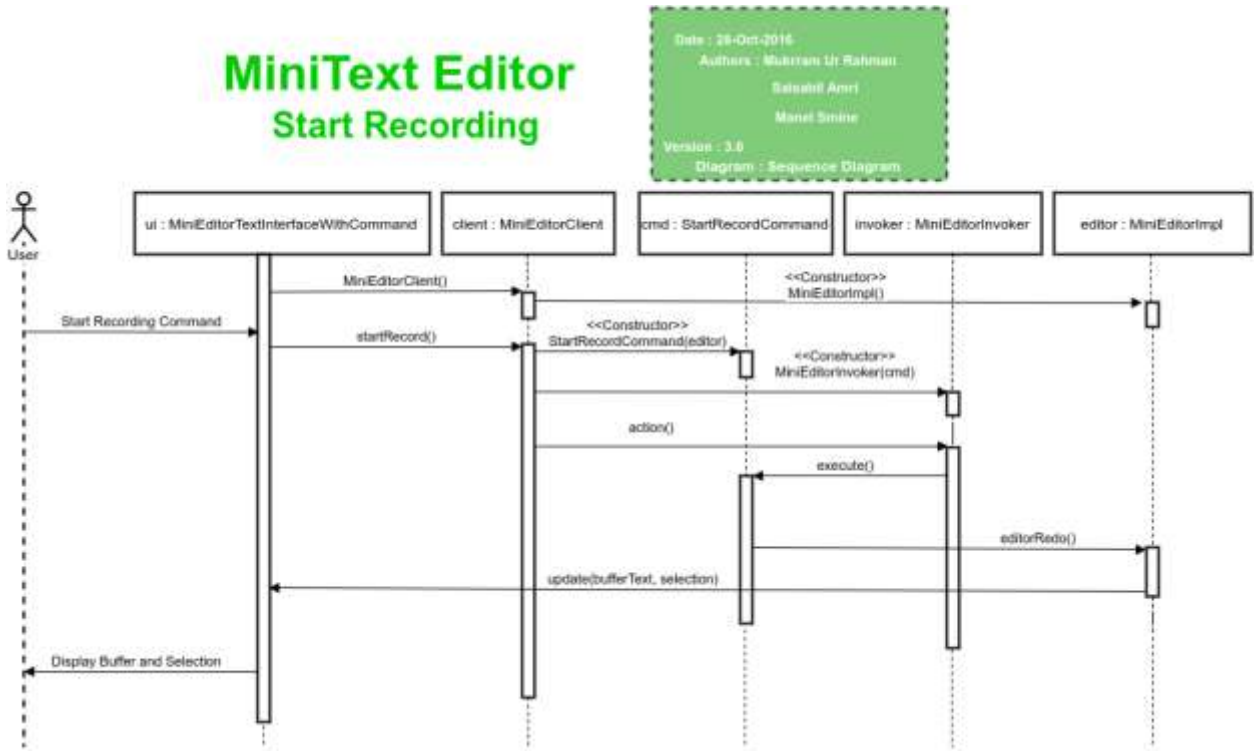

MiniText Editor
Copy Text
Date : 28-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : CopyCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Copy Command
copy()
<<Constructor>>
CopCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorCopy()
update(bufferText, selection)
Display Buffer and Selection

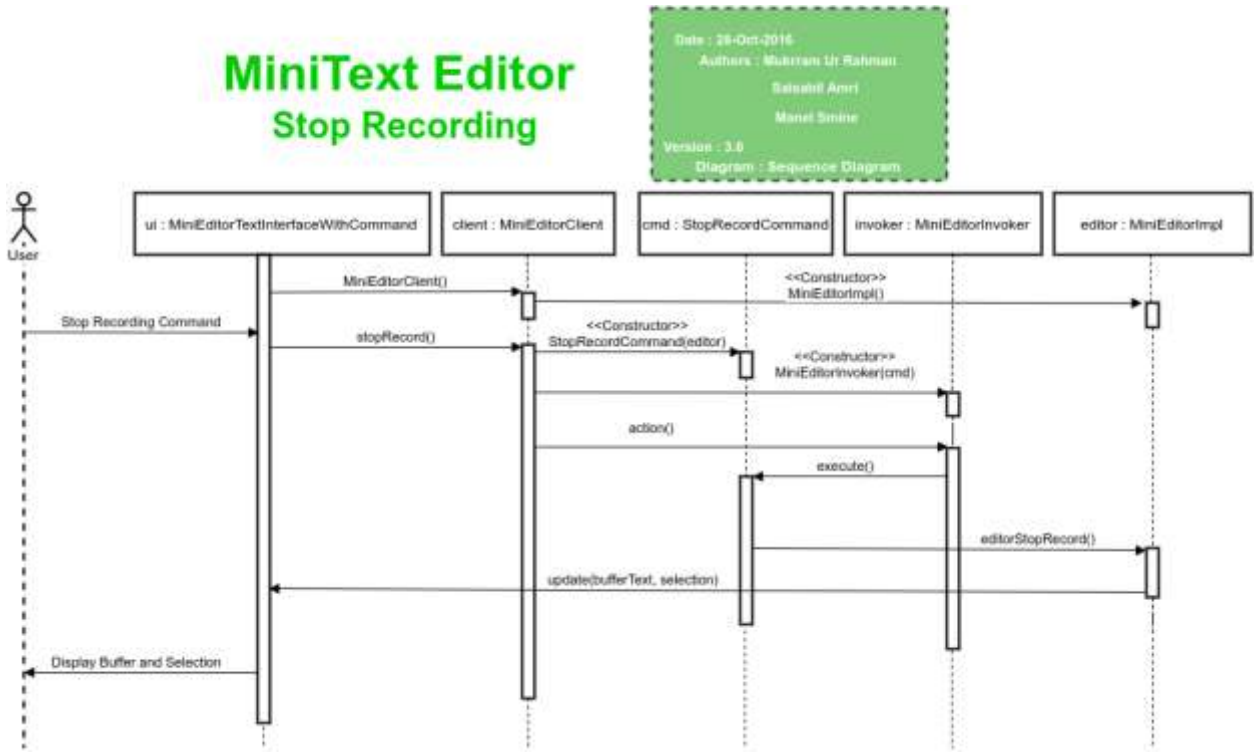

MiniText Editor
Paste Text
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : PasteCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Paste Command
paste()
<<Constructor>>
PasteCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorPaste()
update(bufferText, selection)
Display Buffer and Selection

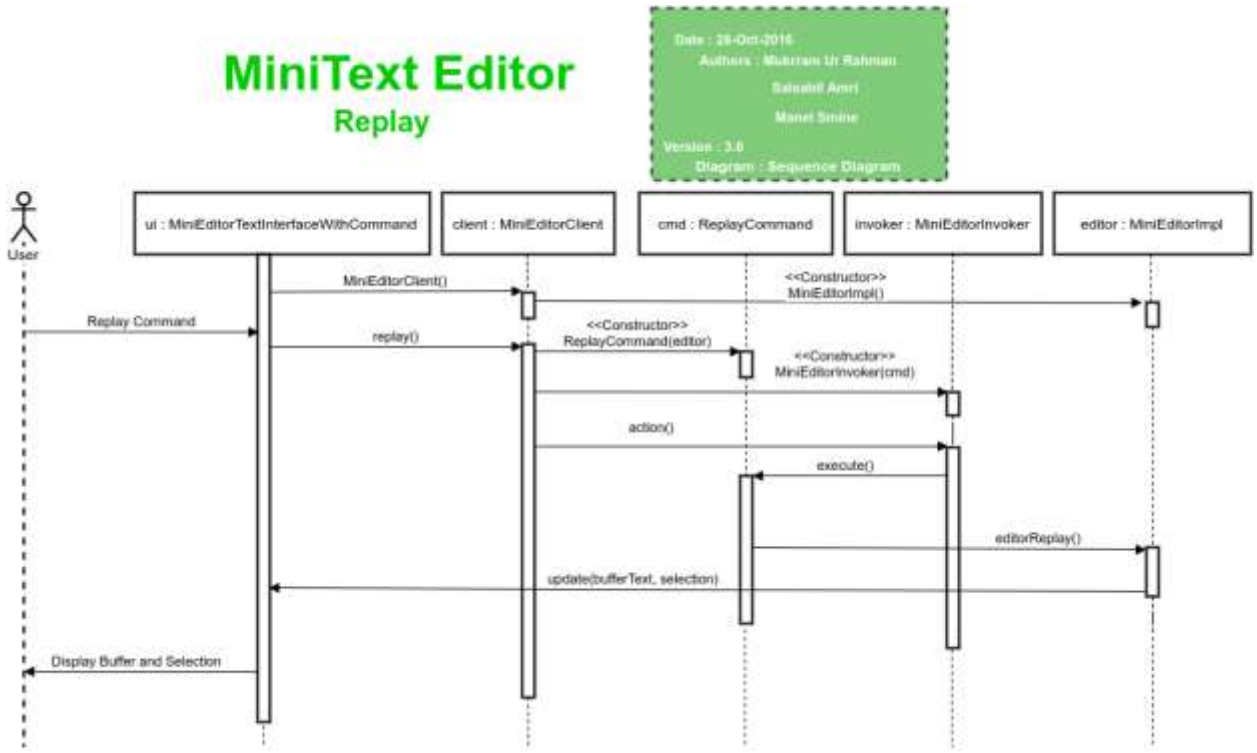

MiniText Editor
Cut Text
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : CutCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Cut Command
cut()
<<Constructor>>
CutCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorCut()
update(bufferText, selection)
Display Buffer and Selection

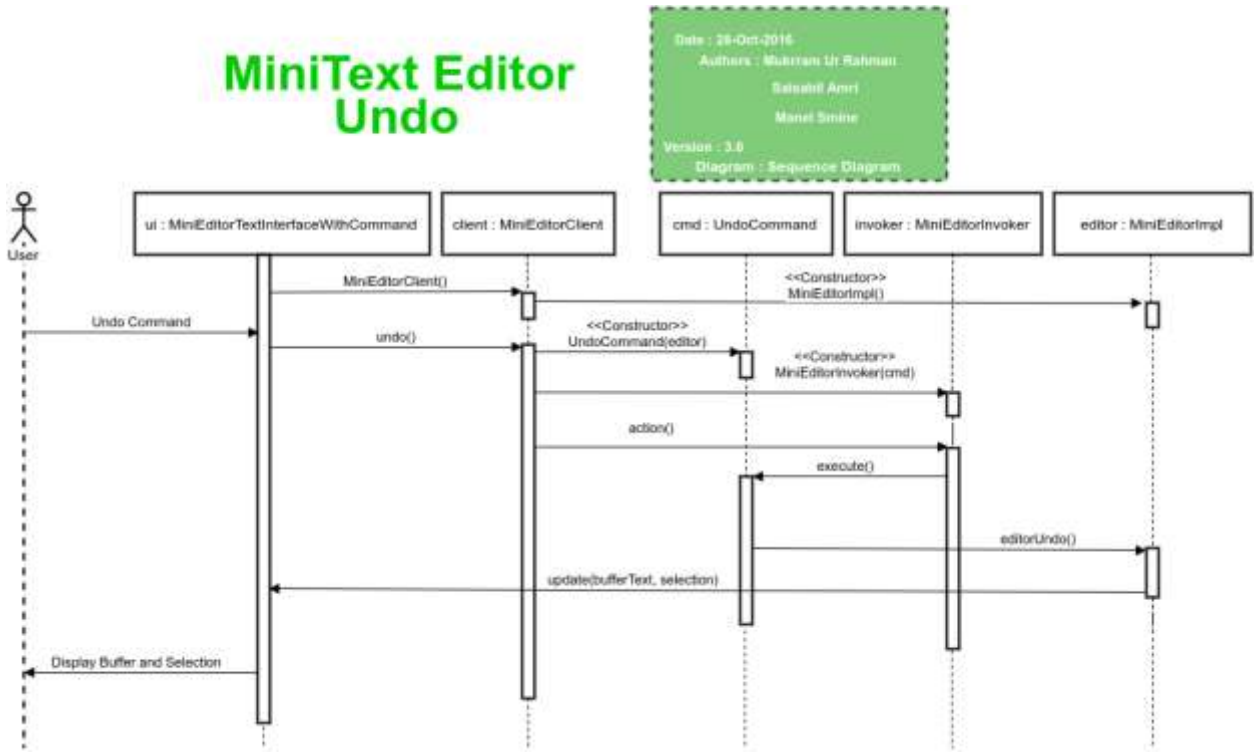

MiniText Editor
Delete Text
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : DeleteCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Delete Command
delete()
<<Constructor>>
DeleteCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorDelete()
update(bufferText, selection)
Display Buffer and Selection

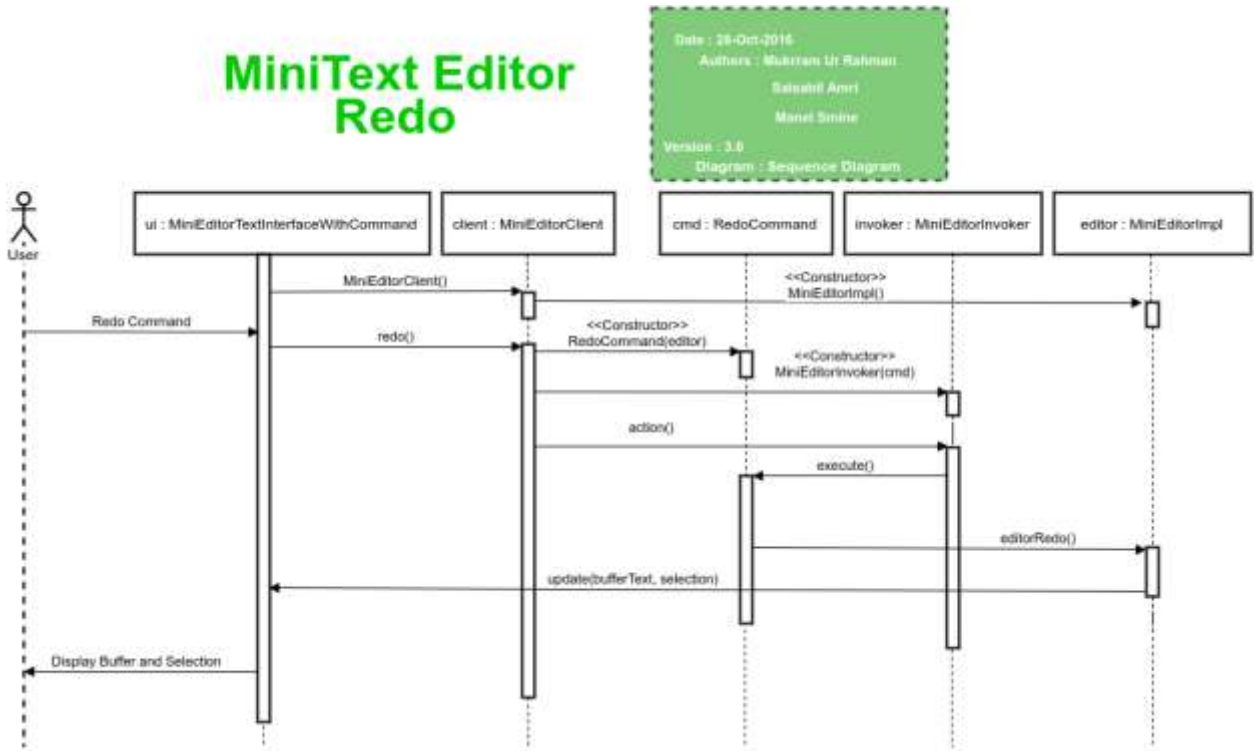

MiniText Editor
Select Text
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : SelectCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Select Command with Selection
select(start, end)
<<Constructor>>
SelectCommand(start, end, editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorSelect(start, end)
update(bufferText, selection)
Display Buffer and Selection

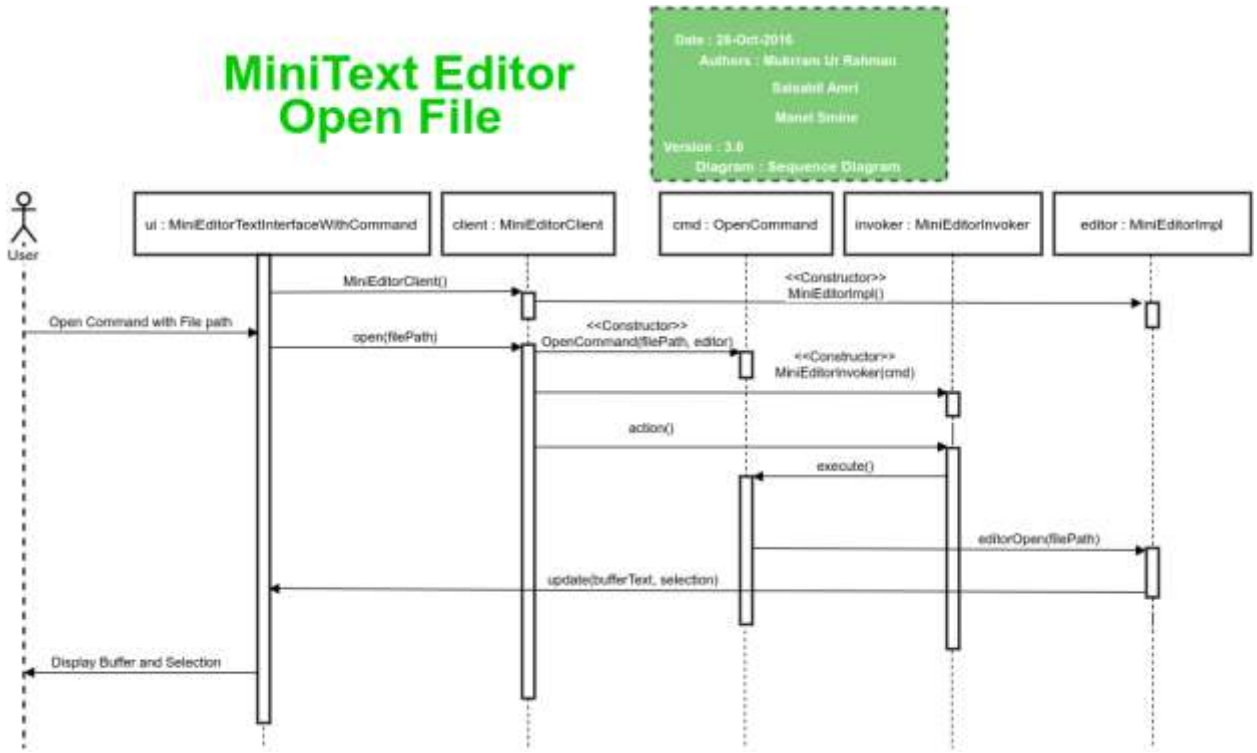

MiniText Editor
Start Recording
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : StartRecordCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Start Recording Command
startRecord()
<<Constructor>>
StartRecordCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorRedo()
update(bufferText, selection)
Display Buffer and Selection

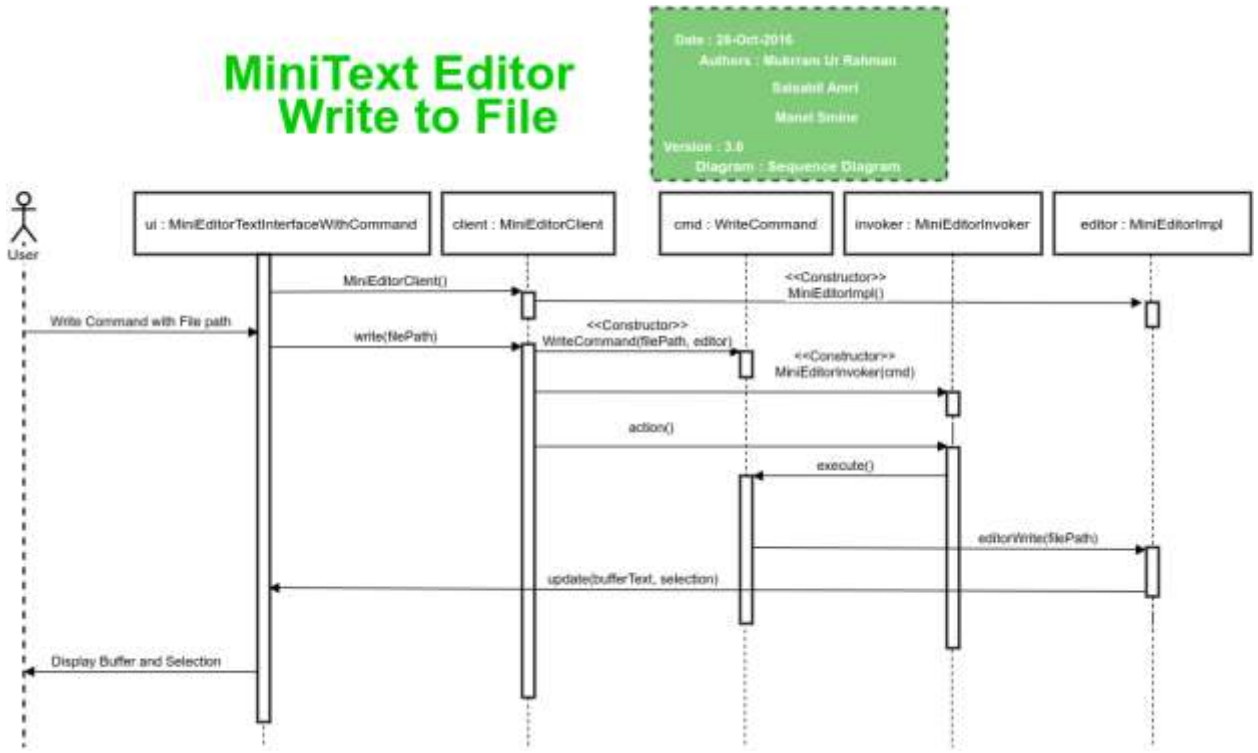

MiniText Editor
Stop Recording
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : StopRecordCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Stop Recording Command
stopRecord()
<<Constructor>>
StopRecordCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorStopRecord()
update(bufferText, selection)
Display Buffer and Selection


MiniText Editor
Replay
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : ReplayCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Replay Command
replay()
<<Constructor>>
ReplayCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorReplay()
update(bufferText, selection)
Display Buffer and Selection


MiniText Editor
Undo
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : UndoCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Undo Command
undo()
<<Constructor>>
UndoCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorUndo()
update(bufferText, selection)
Display Buffer and Selection


MiniText Editor
Redo
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : RedoCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Redo Command
redo()
<<Constructor>>
RedoCommand(editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorRedo()
update(bufferText, selection)
Display Buffer and Selection


MiniText Editor
Open File
Date : 26-Oct-2016
Authors : Mukrram Ur Rahman
Salsabil Amri
Manel Smine
Version : 3.0
Diagram : Sequence Diagram
User
ui : MiniEditorTextInterfaceWithCommand
client : MiniEditorClient
cmd : OpenCommand
invoker : MiniEditorInvoker
editor : MiniEditorImpl
MiniEditorClient()
<<Constructor>>
MiniEditorImpl()
Open Command with File path
open(filePath)
<<Constructor>>
OpenCommand(filePath, editor)
<<Constructor>>
MiniEditorInvoker(cmd)
action()
execute()
editorOpen(filePath)
update(bufferText, selection)
Display Buffer and Selection



Testing: Unit tests were performed using JUnit to compare the actual functioning of the system against those expected functionalities specified in the Technical Specifications. The following test cases were used for the tests:

- **MiniEditorStroageTest**

No.	Test	Pass/Fail
1	Test for empty clip boad.	**Pass**
2	Content change test.	**Pass**
3	Buffer change test.	**Pass**

- **SimpleCommandTest**

No.	Test	Pass/Fail
1	Test insert text functionality.	**Pass**
2	Test copy text functionality.	**Pass**
3	Test paste text functionality.	**Pass**
4	Test cut text functionality.	**Pass**
5	Test select text functionality.	**Pass**
6	Test delete text functionality.	**Pass**

- **CompoundCommandTest**

No.	Test	Pass/Fail
1	Test combination of insert and paste operation.	**Pass**
2	Test combination of insert and copy operation.	**Pass**
3	Test combination of insert, copy and paste operation.	**Pass**

4	Test N(1000) times copy operation.	**Pass**
5	Test N(1000) times paste operation.	**Pass**
6	Test combination of cut and select operation.	**Pass**
7	Test combination of select and copy operation.	**Pass**
8	Test combination of delete and copy operation.	**Pass**
9	Test combination of delete and paste operation.	**Pass**
10	Test N(1000) times undo operation.	**Pass**
11	Test N(1000) times redo operation.	**Pass**
12	Test for empty buffer.	**Pass**

Code coverage: We used Jenkins to generate run and generate code converge reports for the main functional classes, and it generated 88% code coverage. Above 80 is considered a good figure.

[1] **https://en.wikipedia.org/wiki/Eclipse_(software)**