AJAX DEVELOPMENT ENVIRONMENT

FINAL DESIGN REPORT

karınca teknoloji	FINAL DESIGN REPORT	Date: 18.01.2006

1. INTRODUCTION

1.1. Purpose of This Document

This document is prepared to demonstrate the initial design of our product "kajax". Our main purpose, while preparing this document, was to show our design process and what kind of solutions we developed and our point of view to the Ajax IDE Project. We explained our design process via various diagrams to clarify every single detail of the reader's mind. These diagrams include Data Flow Diagrams, Sequence Diagrams, Use Case Diagrams and Class Diagrams.

Actually since this is the initial design report of the product, we cannot say the report is covering every aspect of the design. There are probably some incomplete parts about the design process at the report. However, Karınca team will work extremely concentrated and hard to form a whole design report.

1.2. Problem Redefinition

For the last few years Internet and Web-Applications have undergone a very sharp shift. What caused this situation is emerging the new Ajax technology. Ajax promised bandwidth utilization, speed and high interactivity which is the indispensable feature of the desktop applications.

AJAX refers to a set of techniques centered on background browser to server communication. Instead of always submitting a full page of data to the server and receiving a full page back, by using AJAX techniques, an application can send an individual field value and receive information to update a portion of the page. The result is that, with AJAX, web applications can be much more responsive and intuitive.

Since AJAX is a new technology, new development environments should be implemented to support this technology. As a result of our researches, the number of development environment is not enough and most of available ones are integrated to the general development environment such as Eclipse, Net Beans etc. In other words they are not a stand alone product for the web developers. To meet the needs, we have decided to develop an AJAX IDE (Integrated Development Environment) named as **kajax**.

1.3 Project Scope and Goals

Our product **kajax** is simply an Integrated Development Environment for Ajax. **kajax** is designed for developers to develop web applications faster and easier with in a user friendly environment with the following goals:

- To develop an independent desktop application/IDE with a setup, supporting operating systems Windows and Linux.
- To develop a modular product using a component based approach.
- To provide a Database Module with database connection, sql query, viewing tables features.
- To supply an Editor Module with HTML, CSS, XML and JavaScript text editors.
- To provide a Debugger Module with a JavaScript Debugger.
- To supply Remote Connection Module with FTP.
- To provide Panel Module with File Explorer, Solution Explorer, Toolbox, Properties and Ajax Action panels.
- To supply Versioning Manager Module with CVS.

1.4 Design Goals and Objectives

1.4.1 Modularity

We have introduced a modular approach to the project, so we resolved kajax into modules. At the result of this approach, the design emerged in a very modular way. We think that this modularity will be useful during coding phase. The modularity feature of our design also helped us during making the design by providing us to grasp all the aspects of the project in a complete way. This feature will especially help us during the coding phase through implementing the features of our modules in a detailed and compact way. High modularity of our project design also showed us the possible difficulties that we will face and made us to concentrate on those parts. We think that this feature will be also very useful during the integration process.

1.4.2 Usability & Portability

In our project design one of the most important issues that we gave importance is usability. We tried to design all the modules in a manner that the project will be implemented easily and at the result of implementation, we aim to produce software which will be used by developers easily. The design is aimed to be made in a good way that users will not confront with any complexity while using the system. Mnemonics are an important subject in the sense of usability. So we designed things as people accustomed to. Besides these portability is an important issue. Today most developers use different platforms such as Linux, Windows, etc. So we considered this situation as a crucial one and selected Java as the implementation language.

1.4.3 Reliability

We are planning to program our product reliable so that any minor program will not stop the program or corrupt the program. For that aim our testers will make white or black box tests to minimize the bugs of our program. So that our program will be as bug free as possible. All modules should work asynchronously so that any delay of one of our modules should not block others work flow.

1.5 Design Constraints

Our main design constraints are as follows.

1.5.1 Time

Our fixed schedule is determined by our course syllabus. We have approximately six months remaining to finish the project completely. The design should be finished in one month. During the design we will also work on the prototype and it will be finished in one and a half months from the delivery of initial design report.

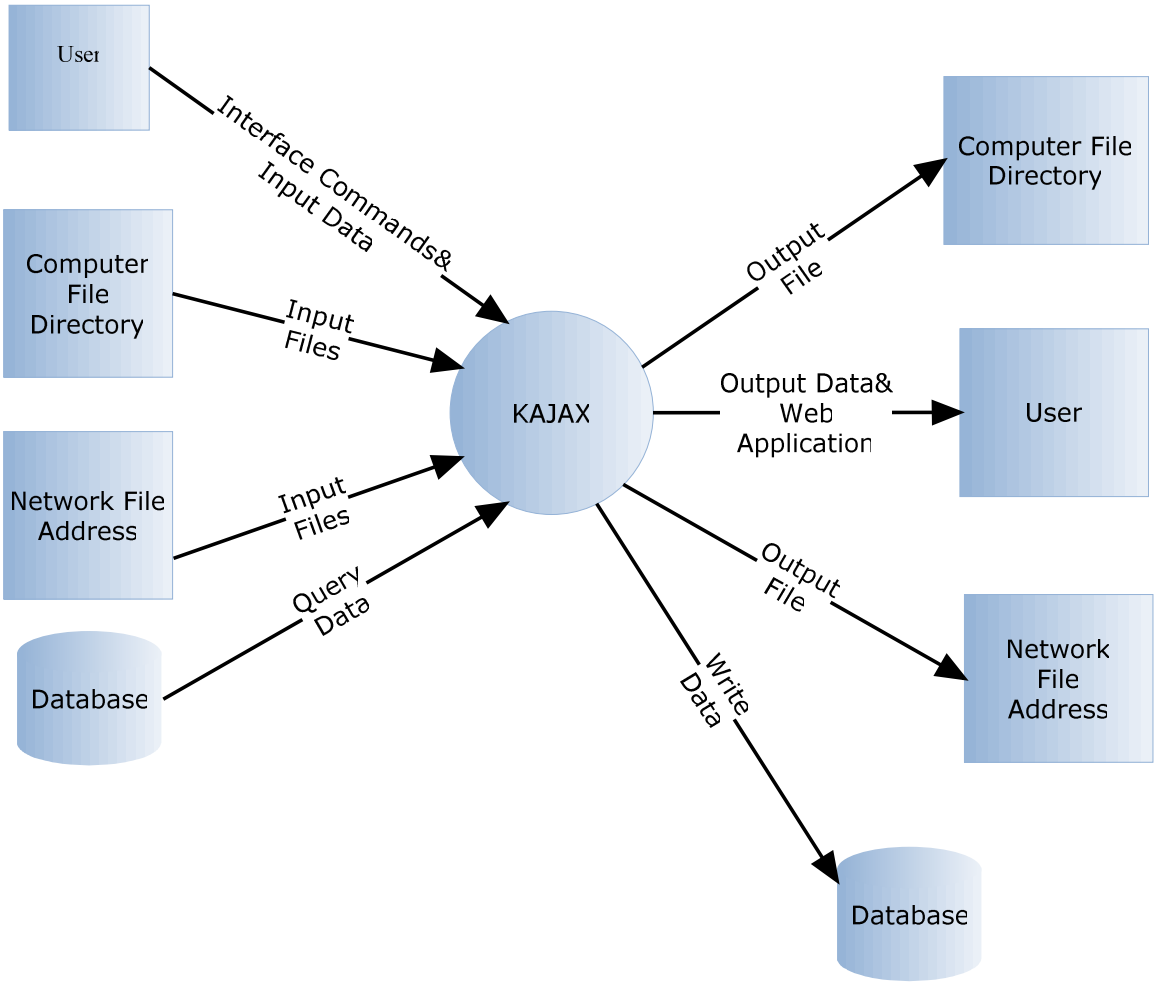
1.5.2 Performance

The performance of our program is very important. A slow development environment will not satisfy the needs of the user. Since we are using Java technologies in our program, we have to be more careful about the performance issue. The system

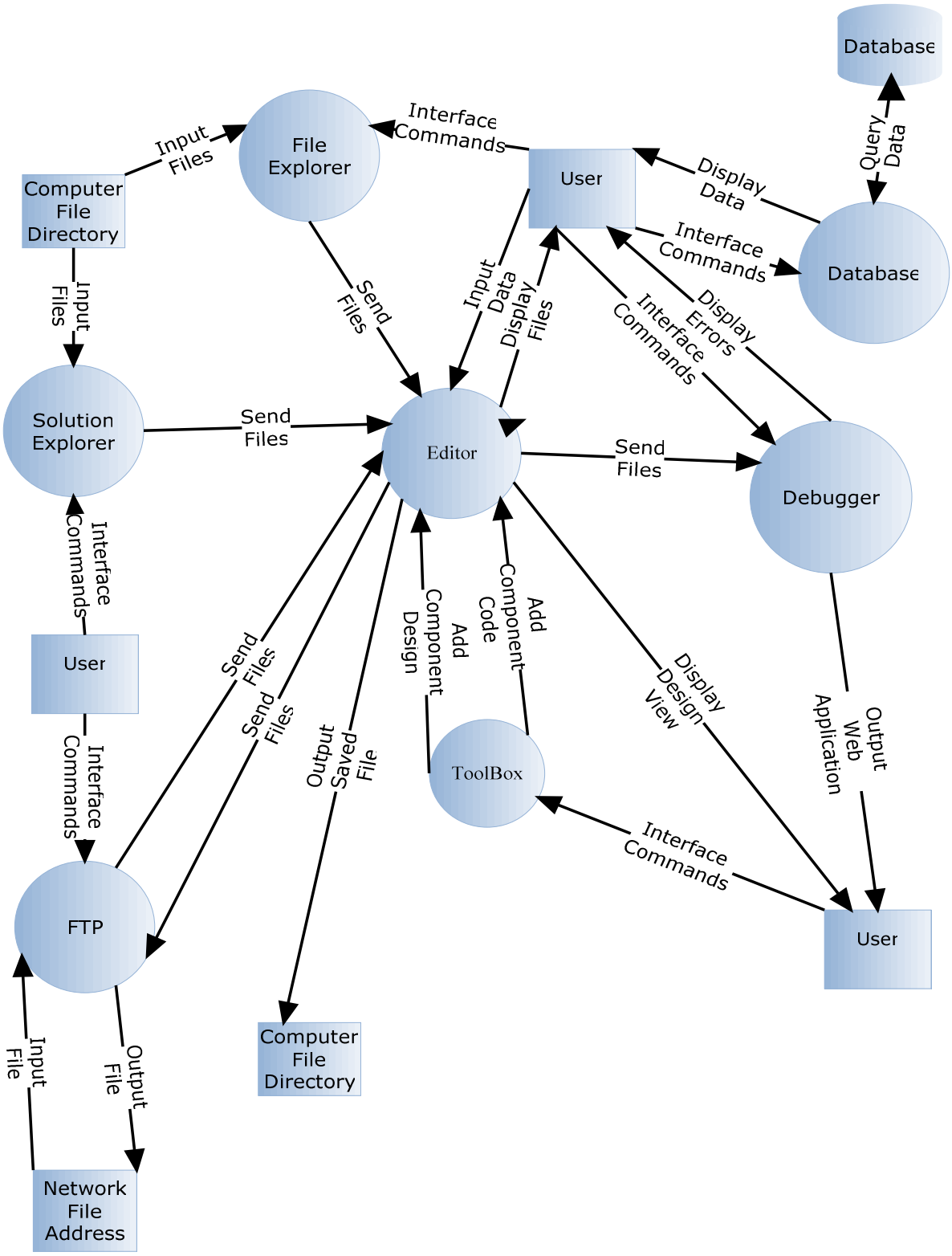
resource usage will be minimized to increase the performance on slower computers. The user should run other applications while using our program.

2. DATA FLOW DIAGRAMS

2.1 DFD LEVEL0



2.2 DFD LEVEL1


Database
Query Data
File Explorer
Interface Commands
User
Display Data
Input Files
Computer File Directory
Interface Commands
Database
Input Files
Send Files
Input Data
Display Files
Display Errors
Interface Commands
Solution Explorer
Send Files
Editor
Send Files
Debugger
Interface Commands
User
Send Files
Send Files
Component Design
Add Component Code
Add
Output Saved File
Display Design View
Output Web Application
ToolBox
Interface Commands
FTP
User
Input File
Output File
Computer File Directory
Network File Address

2.3 DATA DICTIONARY

Name:	Interface Commands & Input Data
Where & How it is used:	*DFD Level0* *DFD Level1(input given by user)*
Description:	"users interaction commands"

Name:	Input Files
Where & How it is used:	*DFD Level 0, DFD Level1.FTP(input),* *Solution Explorer(input files to be loaded by user)*
Description:	"project files"

Name:	Query Data
Where & How it is used:	*DFD Level0* *DFD Level1.Database Module(output info)*
Description:	"results returned after query statements"

Name:	Output File
Where & How it is used:	*DFD Level0* *DFD Level1. File System & FTP(output file for saving)*
Description:	"user files to be saved in local or network file system"

Name:	Output Data & Web Application

Where & How it is used:	*DFD Level0* *DFD Level1.Debugger(output to user)*
Description:	"debug results and viewing data"

Name:	Write Data
Where & How it is used:	*DFD Level0* *DFD Level1.Database Module(input data)*
Description:	"update or insert statements that are queried on DB."

Name:	Send Files
Where & How it is used:	*DFD Level1. FTP(output-input), File Explorer(output) ,* *Editor(input-output), Debugger Modules(input)*
Description:	"writes file data or loads them in editor or debugger"

Name:	Display Data
Where & How it is used:	*DFD Level1.Database Module(output)*
Description:	"outputs, results to the queries"

Name:	Display Errors
Where & How	*DFD Level1. Debugger Module(output)*

it is used:	
Description:	“debugging info and messages to the user”

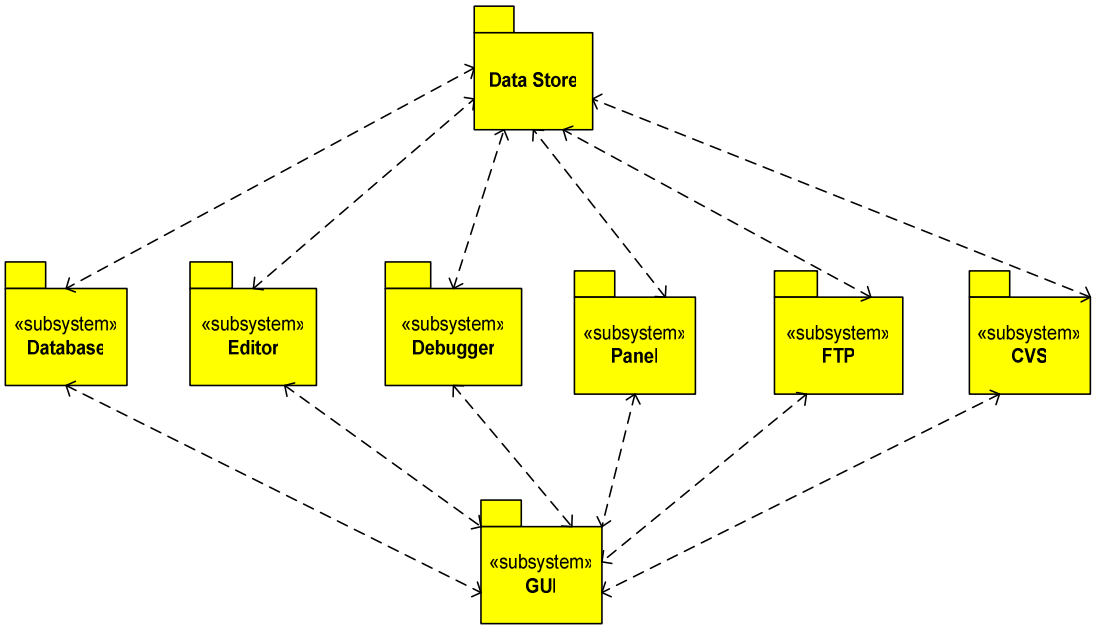
Name:	Display Design View
Where & How it is used:	*DFD Level1. Editor(output)*
Description:	“output to design mode”

Name:	Add Component Code
Where & How it is used:	*DFD Level1. Toolbox(output), Editor(input)*
Description:	“components that are provided by the ide”

3. SYSTEM ARCHITECTURE, MODULES AND FILE FORMATS

3.1 SYSTEM ARCHITECTURE

Every system has to have architecture. As we have mentioned in our Analyses Report in words our program **kajax** has component based architecture. The following diagram shows major modules of the **kajax** with structure and interactions between them.



An overview of these components is below:

Database Module provides database connections, executing SQL statements and viewing database tables.

Editor Module provides HTML, CSS, XML and JavaScript text editors to the **kajax** user.

Debugger Module is a JavaScript debugger.

FTP Module is used for receiving and putting files from/to a remote FTP Server. This module acts as a FTP Client.

CVS Module is used to connect a remote CVS server. This Module acts as a CVS Client.

Panel Module provides File Explorer, Project Explorer, Toolbox, Properties and the AJAX ACTIONS panels. Ajax actions panel is one of the most important feature of the **kajax.**

3.2 MODULES

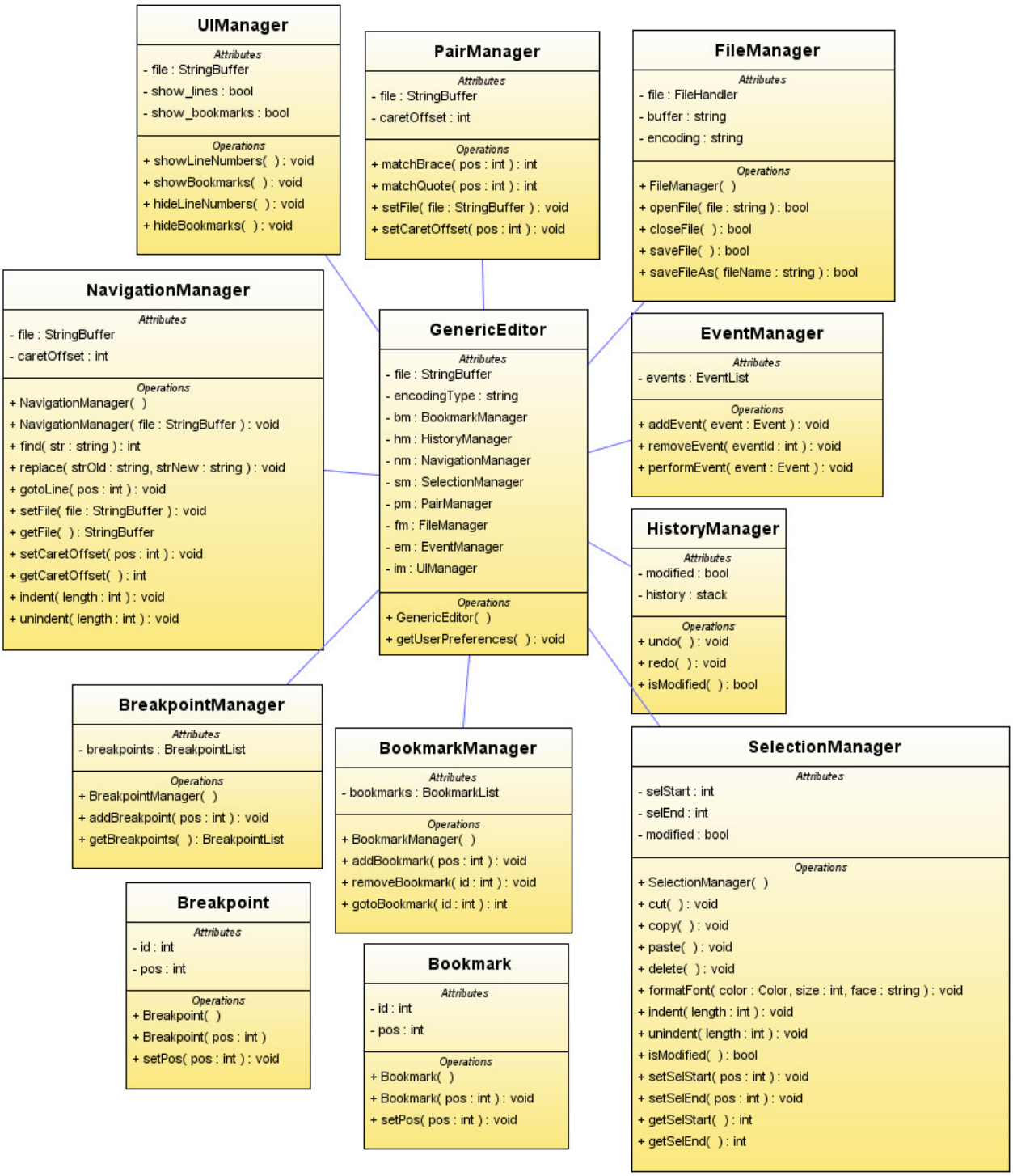
3.2.1 EDITOR MODULE

Text editors are the core components of any integrated development environment. It keeps developer from redundant actions by automating lots of things like indentation, code completion, etc. And also helps the programmer to catch simple errors. **kajax** provides a functional text editor module. Since it is an Ajax IDE it has four specialized editors: Html, JavaScript, Xml and Css editors. All of them support basic functionalities such as pair matching, indenting, highlighting, etc. Besides this colorization, tag completion assistance, validation check are provided. User can bookmark lines and put break points for the debug phase. Also show/hide line numbers, bookmarks and break points.

Our editor module mainly works as follows: Editor Controller is the top controller class. It will have just one instance at the start of the program. It manages new documents, new tabs – new editors. When user wants to open a file or create a new one, editor controller opens a new tab and creates an instance of generic editor. Then generic editor tries to open that file or create new one. According to the type of the file generic editor is transformed to the appropriate editor.

3.2.1.1 CLASS DIAGRAM

• Base editor class is generic editor. It has common functionalities between the specialized ones. It is formed by various managers, so it has a multi-component structure:


UIManager
Attributes
- file : StringBuffer
- show_lines : bool
- show_bookmarks : bool
Operations
+ showLineNumbers( ) : void
+ showBookmarks( ) : void
+ hideLineNumbers( ) : void
+ hideBookmarks( ) : void
PairManager
Attributes
- file : StringBuffer
- caretOffset : int
Operations
+ matchBrace(pos : int) : int
+ matchQuote(pos : int) : int
+ setFile(file : StringBuffer) : void
+ setCaretOffset(pos : int) : void
FileManager
Attributes
- file : FileHandler
- buffer : string
- encoding : string
Operations
+ FileManager( )
+ openFile(file : string) : bool
+ closeFile( ) : bool
+ saveFile( ) : bool
+ saveFileAs(fileName : string) : bool
NavigationManager
Attributes
- file : StringBuffer
- caretOffset : int
Operations
+ NavigationManager( )
+ NavigationManager(file : StringBuffer) : void
+ find(str : string) : int
+ replace(strOld : string, strNew : string) : void
+ gotoLine(pos : int) : void
+ setFile(file : StringBuffer) : void
+ getFile( ) : StringBuffer
+ setCaretOffset(pos : int) : void
+ getCaretOffset( ) : int
+ indent(length : int) : void
+ unindent(length : int) : void
GenericEditor
Attributes
- file : StringBuffer
- encodingType : string
- bm : BookmarkManager
- hm : HistoryManager
- nm : NavigationManager
- sm : SelectionManager
- pm : PairManager
- fm : FileManager
- em : EventManager
- im : UIManager
Operations
+ GenericEditor( )
+ getUserPreferences( ) : void
EventManager
Attributes
- events : EventList
Operations
+ addEvent(event : Event) : void
+ removeEvent(eventId : int) : void
+ performEvent(event : Event) : void
HistoryManager
Attributes
- modified : bool
- history : stack
Operations
+ undo( ) : void
+ redo( ) : void
+ isModified( ) : bool
BreakpointManager
Attributes
- breakpoints : BreakpointList
Operations
+ BreakpointManager( )
+ addBreakpoint(pos : int) : void
+ getBreakpoints( ) : BreakpointList
BookmarkManager
Attributes
- bookmarks : BookmarkList
Operations
+ BookmarkManager( )
+ addBookmark(pos : int) : void
+ removeBookmark(id : int) : void
+ gotoBookmark(id : int) : int
SelectionManager
Attributes
- selStart : int
- selEnd : int
- modified : bool
Operations
+ SelectionManager( )
+ cut( ) : void
+ copy( ) : void
+ paste( ) : void
+ delete( ) : void
+ formatFont(color : Color, size : int, face : string) : void
+ indent(length : int) : void
+ unindent(length : int) : void
+ isModified( ) : bool
+ setSelStart(pos : int) : void
+ setSelEnd(pos : int) : void
+ getSelStart( ) : int
+ getSelEnd( ) : int
Breakpoint
Attributes
- id : int
- pos : int
Operations
+ Breakpoint( )
+ Breakpoint(pos : int)
+ setPos(pos : int) : void
Bookmark
Attributes
- id : int
- pos : int
Operations
+ Bookmark( )
+ Bookmark(pos : int) : void
+ setPos(pos : int) : void

UImanager:

Method Name	Type	Arguments	Description of Method
showLineNumbers()	Public:void	Void	Shows line numbers graphically
showBookmarks()	Public:void	Void	Shows bookmarks graphically near the lines
hideLineNumbers()	Public:void	Void	Hides line numbers graphically
hideBookmarks()	Public:void	Void	Hides bookmarks graphically

Pairmanager:

Method Name	Type	Arguments	Description of Method
matchBrace	Public:int	Pos:int	Highlights the matching braces
matchQuote	Public:int	Pos:int	Highlights the matching braces
setFile()	Public:void	File:StringBuffer	Sets the editors content to the buffer
setCaret()	Public:void	Pos:int	Sets current caret position to this classes caretOffset member

File manager:

Method Name	Type	Arguments	Description of Method
openFile()	Public:bool	File:string	Opens the file specified by the argument
closeFile()	Public:bool	Void	Closes current file
saveFile()	Public:bool	Void	Saves current file, flushes it to the memory

saveFileAs()	Public:bool	Filename:string	Saves current file with the given name

Event manager:

Method Name	Type	Arguments	Description of Method
addEvent()	Public:void	event:Event	Attaches event to the eventList, these will be static events.
removeEvent()	Public:void	eventId:int	Removes the specified event from the eventList.
performEvent()	Public:void	event:Event	Performs the specified event

History manager:

Method Name	Type	Arguments	Description of Method
Undo()	Public:void	Void	Roll backs the last change in the code
Redo()	Public:void	Void	Redoes the last roll backed action in the code
isModified()	Public:bool	Void	Returns whether there happened any change in the code

Breakpoint manager:

Method Name	Type	Arguments	Description of Method
addBreakpoint()	Public:void	Pos:int	Adds breakpoint to the specified position
removeBreakpoint()	Public:void	Id:int	Removes specified breakpoint
getBreakpoints()	Public:BreakpointList	Void	Returns the list of

			breakpoints(i.e. for debugger)

Bookmark manager:

Method Name	Type	Arguments	Description of Method
addBookmark()	Public:void	Pos:int	Adds bookmark to the specified position
removeBookmark()	Public:void	Id:int	Removes the specified bookmark from the bookmarkList
gotoBookmark()	Public:int	Id:int	Sets caret to the specified bookmark position

Selection manager:

Method Name	Type	Arguments	Description of Method
Cut()	Public:void	Void	Cuts the selected code
Copy()	Public:void	Void	Copies the selected code
Paste()	Public:void	Void	Pastes the selection to the current caret position
Delete()	Public:void	Void	Deletes the selection or current character
formatFont()	Public:void	Color:color,size:int,face:string	Sets the format of the font, it applies to all the text in the

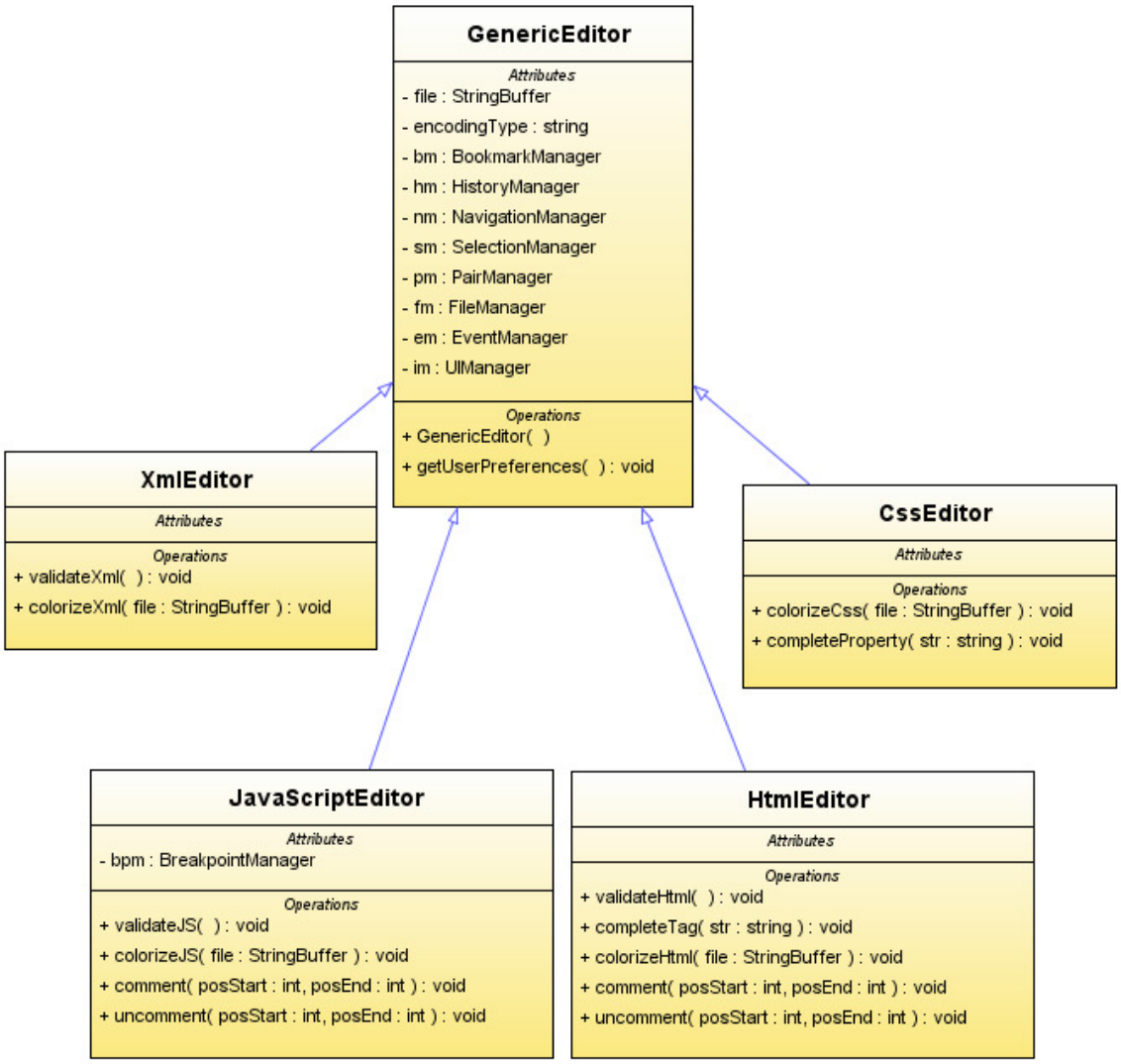
			editor
Indent()	Public:void	Length:int	Indents all the selection by the specified units
Unindent()	Public:void	Length:int	Unindents all the selection by the specified units
isModified()	Public:bool	Void	Returns whether there has been any modification at a time
setSelStart()	Public:void	Pos:int	Sets selection start position
setSelEnd()	Public:void	Pos:int	Sets selection end position
getSelStart()	Public:int	Void	Returns selection start position
getSelEnd()	Public:int	Void	Returns selection end position

Navigation manager:

Method Name	**Type**	**Arguments**	**Description of Method**
Find()	Public:int	Str:string	Finds the given string next by next; it selects the found string
Replace()	Public:void	Strold:string,strnew:string	Replaces the given string with the new one, one

			by one or all at a time
gotoLine()	Public:void	Pos:int	Sets caret position to the given line
setFile()	Public:void	File:stringBuffer	Sets the editors content to the buffer
getFile()	Public:stringBuffer	Void	Returns the buffer reference
setCaretPosition()	Public:void	Pos:int	Sets caret position
getCaretOffset()	Public:int	Void	Returns caret position
Indent()	Public:void	Length:int	Indents current character by specified units
Unindent()	Public:void	Length:int	Unindents current character by specified units

• The derivation classes of generic editor class. We have four different derivations since we have four different editors, html, javascript, xml and css:



Generic editor:

Method Name	Type	Arguments	Description of Method
GenericEditor()	Constructor	Void	Constructs a new generic editor which may be converted to a specific one sooner. When a new object is created, its member managers are also initialized
getUserPreferences()	Public:void	Void	Reads the preferences of the

			user from the file storage, then sets them accordingly

Html editor:

Method Name	Type	Arguments	Description of Method
validateHtml()	Public:void	Void	This is the html validator. It checks the html code for any mistakes. It runs in a thread.
completeTag()	Public:void	Str:string	Completes the given tag accordingly, ex: for a string like "<head>" it completes it with "</head>"
colorizeHtml()	Public:void	File:StringBuffer	Sets different colors to the tags, strings, plain text, etc. according to the predefined schema
Comment()	Public:void	posStart:int,posEnd int	Comments the selected html code
Uncomment()	Public:void	posStart:int,posEnd int	Uncomments the selected html code

Javascript editor:

Method Name	Type	Arguments	Description of Method
validateJS()	Public:void	Void	This is the javascript validator. It checks the javascript code for any syntactic mistakes. It runs in a thread.
colorizeJS()	Public:void	File:StringBuffer	Sets different colors to the

			keywords, strings, plain text, etc. according to the predefined schema
Comment()	Public:void	posStart:int,posEnd int	Comments the selected javascript code
Uncomment()	Public:void	posStart:int,posEnd int	Uncomments the selected javascript code

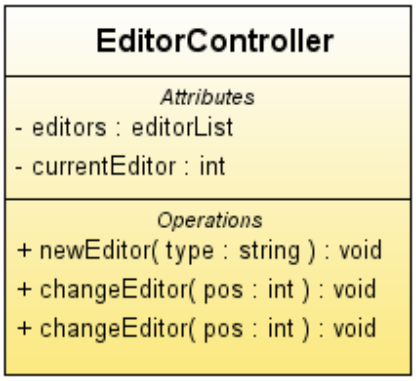
Xml editor:

Method Name	Type	Arguments	Description of Method
validateXml()	Public:void	Void	This is the xml validator. It checks the xml code for any mistakes. It runs in a thread.
colorizeXml()	Public:void	File:StringBuffer	Sets different colors to the tags, strings, plain text, etc. according to the predefined schema

Css editor:

Method Name	Type	Arguments	Description of Method
colorizeCss()	Public:void	File:StringBuffer	Sets different colors to the properties of elements, strings, plain text, etc. according to the predefined schema
completeProperty()	Public:void	Str:string	Presents the user probable element offerings and then completes the property according to the command of user

• Main controller class of the editors which is created once on the start of the program. It handles opening new tabs, closing existing ones and switching between them:

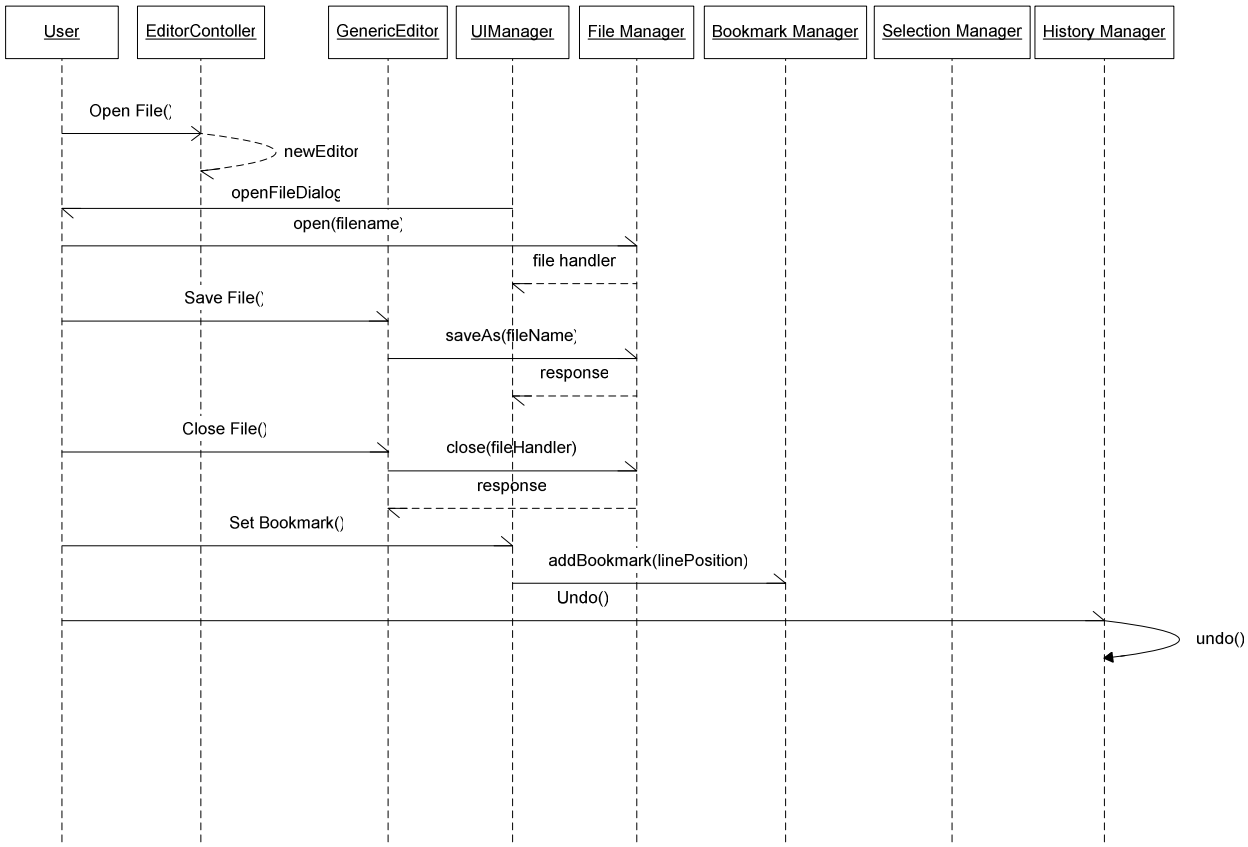


Editor Controller:

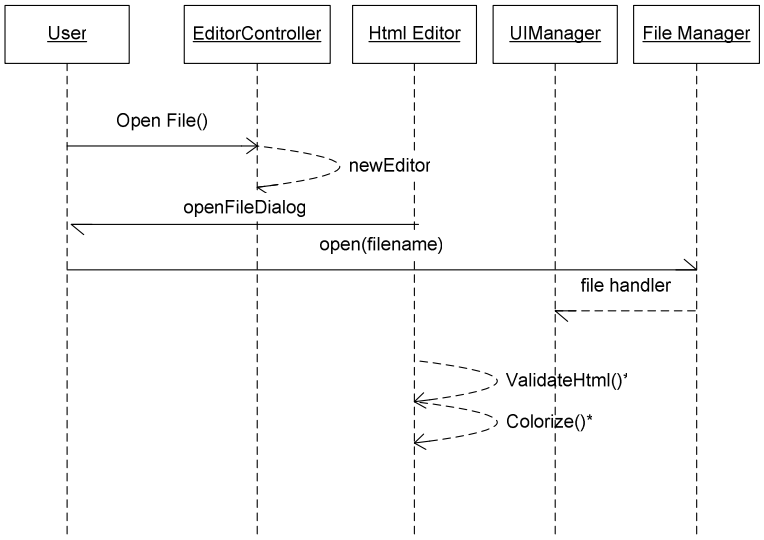
Method Name	Type	Arguments	Description of Method
EditorController	Constructor	Void	Constructs an EditorController object
newEditor()	Public:void	Type:String	Creates a new editor object according to the given type information
changeEditor()	Public:void	Pos:int	This method is used to change the current editor tab
closeEditor()	Public:void	Pos:int	Closes the specified editor tab

3.2.1.2 SEQUENCE DIAGRAM

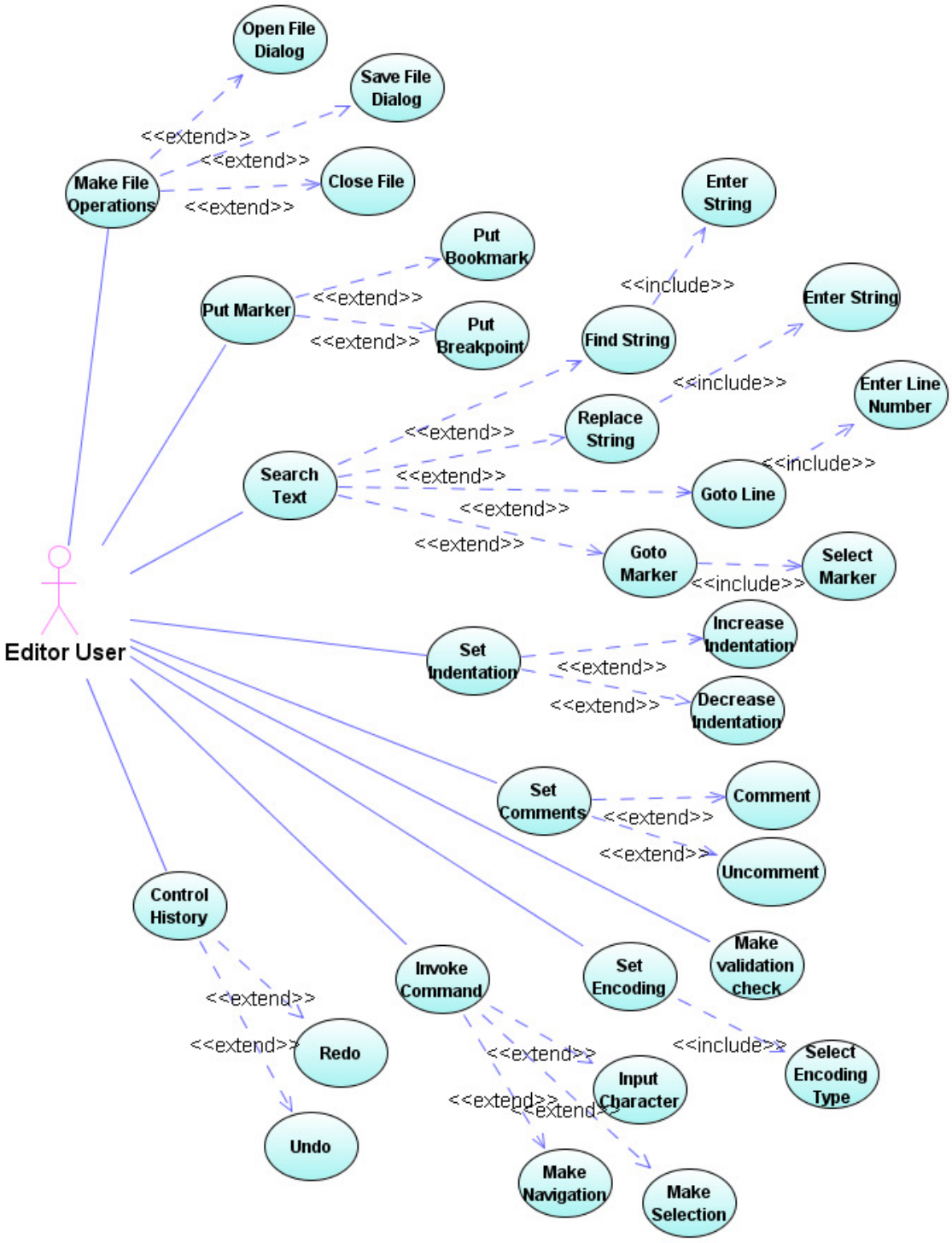
Below is the sequence diagram of the editor module which depicts generic editor. Some of the sequence parts are not showed since they are straightforward and similar to others.



Below is the sequence diagram that belongs to derived html editor. Again some of the parts and other derivations are not showed since they are similar each other.



3.2.1.3 USE CASE DIAGRAM



3.2.1.4 USAGE SCENARIOS

Scenario 1:

To (un)comment a block of code or selection user first determines the block either by mouse controls or by keyboard. Then by using menu or shortcuts he performs the (un)comment operation. By the same way user can set the indentation.

Scenario 2:

To add a bookmark first the line is chosen. Then addBookmark command is given to the BookmarkManager. After a while, if the user wants to go to a location (bookmark) available bookmarks are shown in a frame and gotoBookmark command is passed to the BookmarkManager. From the same frame available bookmarks can be removed.

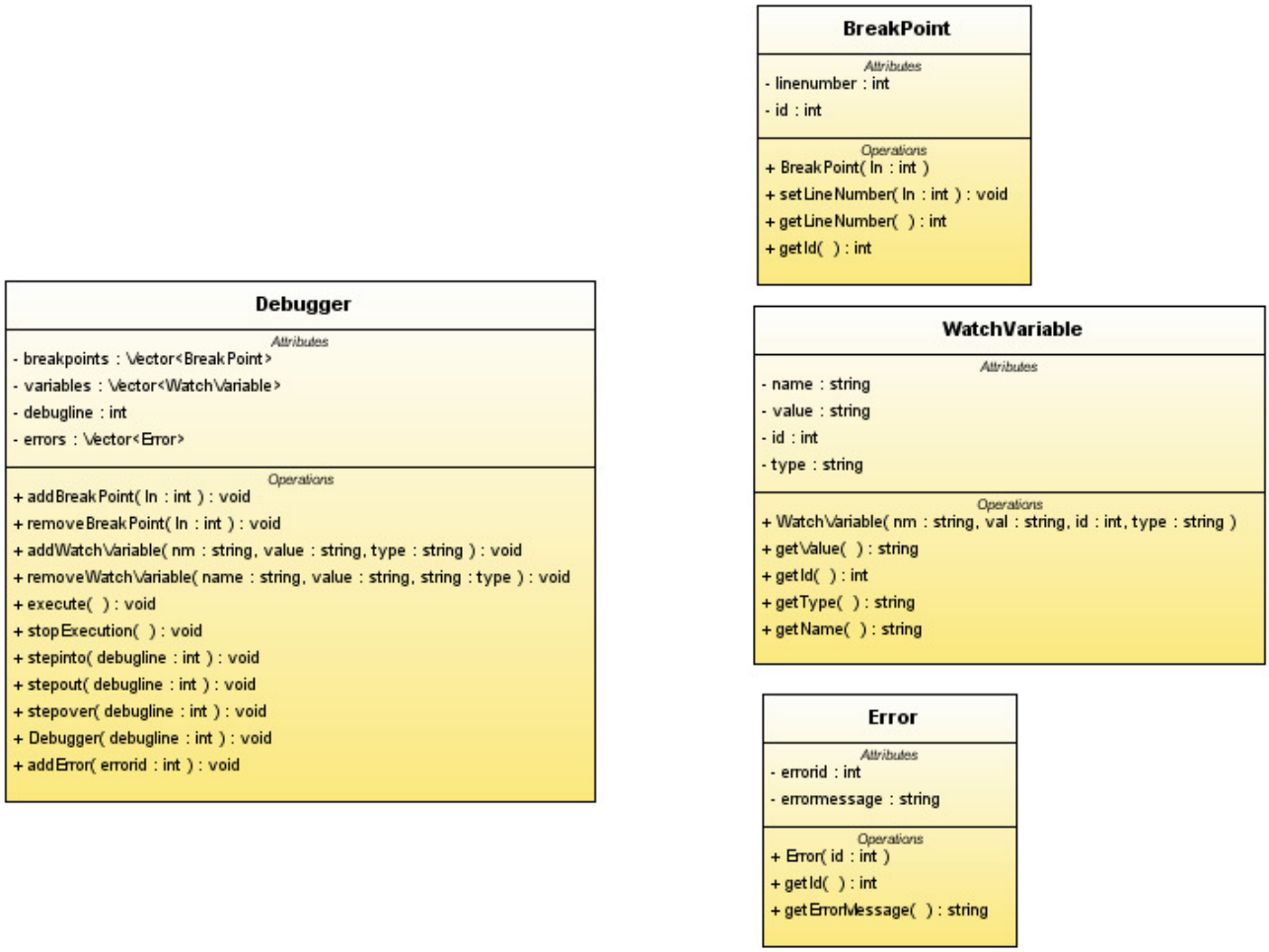
Scenario 3:

To find a string, from an input dialog the string is entered. Then if the string is found it is selected in the interface else a warning is displayed. By the same way any string can be replaced with a specified one.

3.2.2 DEBUGGER MODULE

3.2.2.1 CLASS DIAGRAM

Debugger module is used for debugging the Javascript codes having some additional functionalities such as watch variables, step into/out/over, set breakpoints. It is mainly composed of 4 classes Breakpoint which enables operations in the code, Watch Variables enables choosing and following the values of variables and error class holds the error types and messages for the given code to the debugger. Main Debugger class uses this classes holding a vector data structure for each class which is peculiar to the code debugged on that time.



Here is the definition of Debugger:

BreakPoint Class:

Method Name	Type	Arguments	Description of Method
BreakPoint(int,int)	Constructor	void	Constructs a BreakPoint object with the given line number by the user and assigned id by our system.
setLineNumber(int)	Public:void	ln:string	Sets a line number in the determined line

getLineNumber( )	Public:int	void	Gets the line number of the breakpoint object
getId()	Public:int	void	Gets the id number of the breakpoint object

WatchVariable Class:

Method Name	Type	Arguments	Description of Method
WatchVariable(string,string,int,string)	Constructor	Nm:string, value: string, id:int, type:string	Constructs a WatchVariable object with the given variable name, variable value, id and variable type parameters.
getValue( )	Public:string	void	Gets the value of this variable in the program
getType( )	Public:string	void	Gets the type of this variable in the program
getId( )	Public:int	void	Gets the id of the variable assigned by our system
getName( )	Public:string	void	Gets the name of this variable

Error Class:

Method Name	Type	Arguments	Description of Method

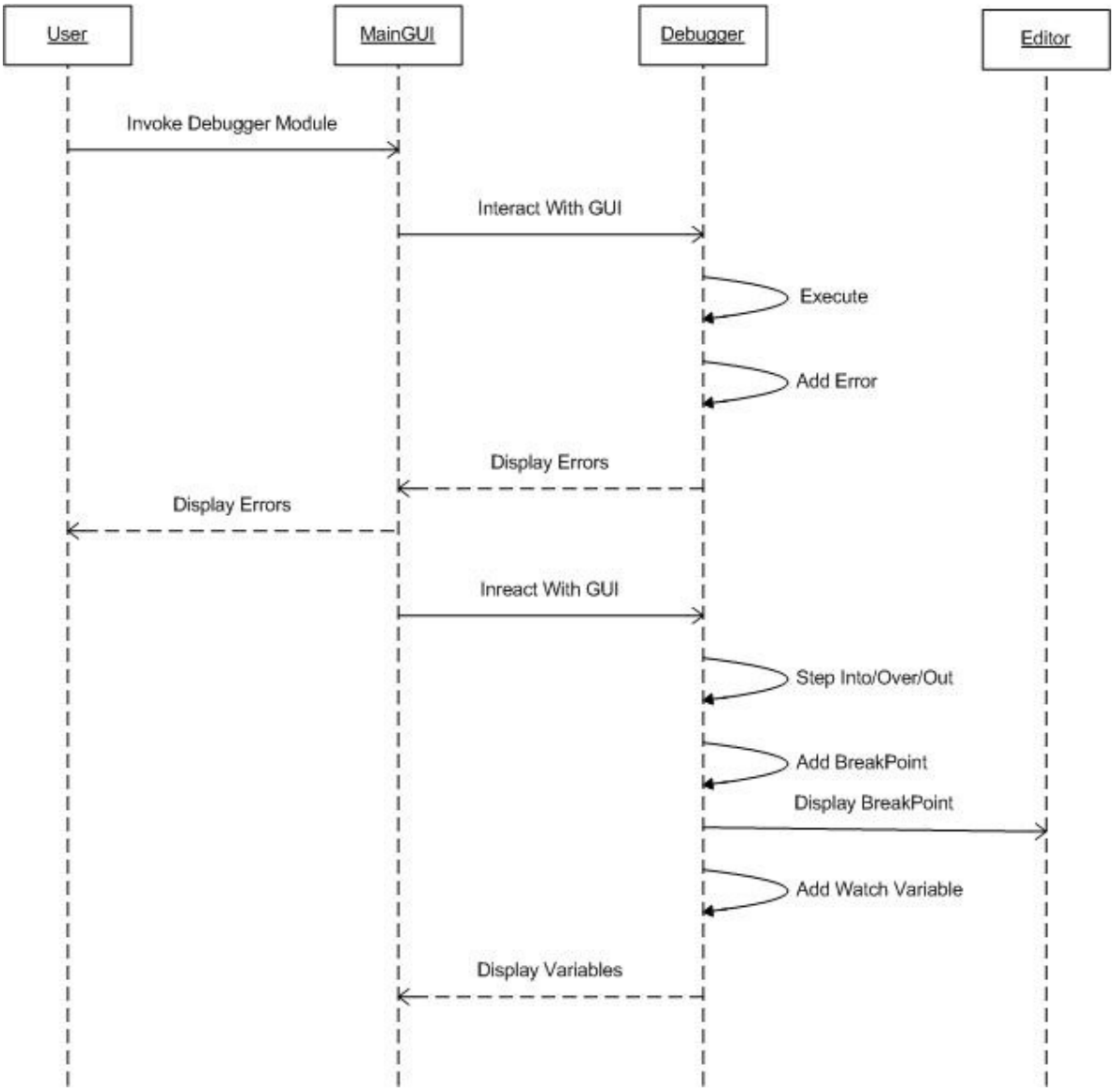
Error(int,string)	Constructor	void	Constructs a Error object with the given string and assigned id by our system.
getErrorMessage( )	Public:string	void	Gets the line number of the breakpoint object
getId()	Public:int	void	Gets the id number of the breakpoint object

Debugger Class:

Method Name	**Type**	**Arguments**	**Description of Method**
Debugger()	Constructor	void	Constructs a Debugger object
StepInto(int)	Public:void	ln:int	Debugger starts to listen user commands in step into model using the given line number
Stepover(int)	Public:void	ln:int	Debugger starts to listen user commands in step over using the given line number

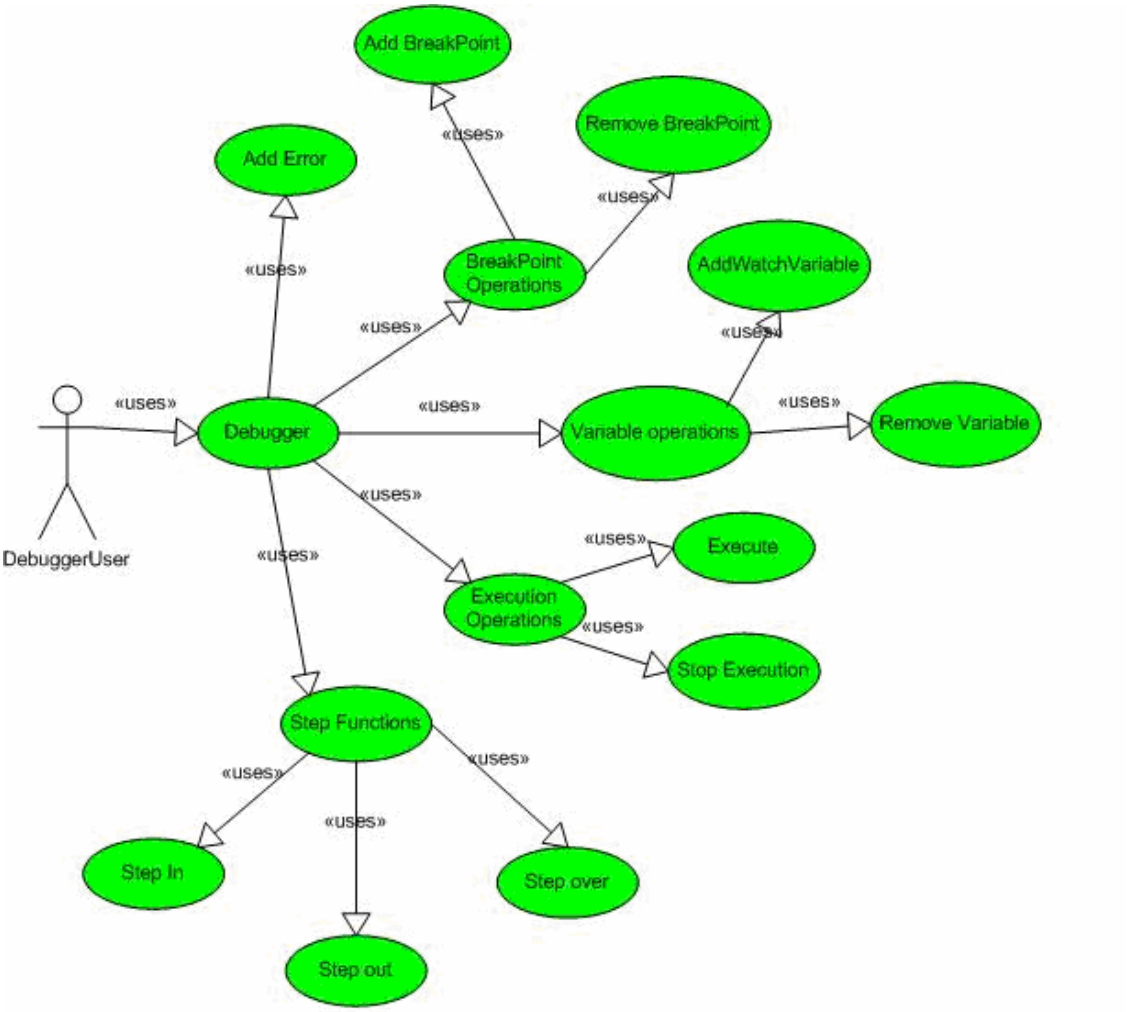
StepOut(int)	Public:void	ln:int	Debugger starts to listen user commands in step out model Using the given line number
Execute()	Public:void	Void	Starts to debug the Javascript code line by line
AddBreakPoint(int)	Public:void	ln:int	Adds a breakpoint to the given line
RemoveBreakPoint(int)	Public:void	ln:int	Removes the breakpoint at the given line
RemoveWatchVariable(string,string,string)	Public:void	nm:string, value: string, type:string	Removes the variable from the vector structure
AddWatchVariable(string,string,string)	Public:void	nm:string, value: string, type:string	Adds a variable to with the given properties in the program

3.2.2.2 SEQUENCE DIAGRAM



Basic functionalities of the debugger module in a sequence are seen above. Firstly the user interacts with the GUI, calls the debugger module and now come to the stage of using the facilities of the debugger module. Then user selects one of the properties and the result of execution is displayed in the Debugger panel of the IDE. If the user selects adding/removing breakpoints simultaneously added/removed breakpoints are displayed in the editor by the editor module of our program.

3.2.2.3 USE CASE DIAGRAM



Flow of Events Debugging Use Case

Objective : To enable the user to debug his file.

Precondition : A selected Javascript file or project should be ready

Main Flow

1. The user interacts with the debugger interface
2. The user selects one of the breakpoint operations BreakPoint, Execute or Step Functions, Variable
3. The user can add new breakpoints or remove breakpoints when selecting breakpoint operation

4. The user can add new variables or remove variables when selecting variable operation
5. User can execute the debugger or stop the debugging.
6. User can apply step into, step over or step out functions to forward line by line

Post-condition The user has done the debugging phase of the code.

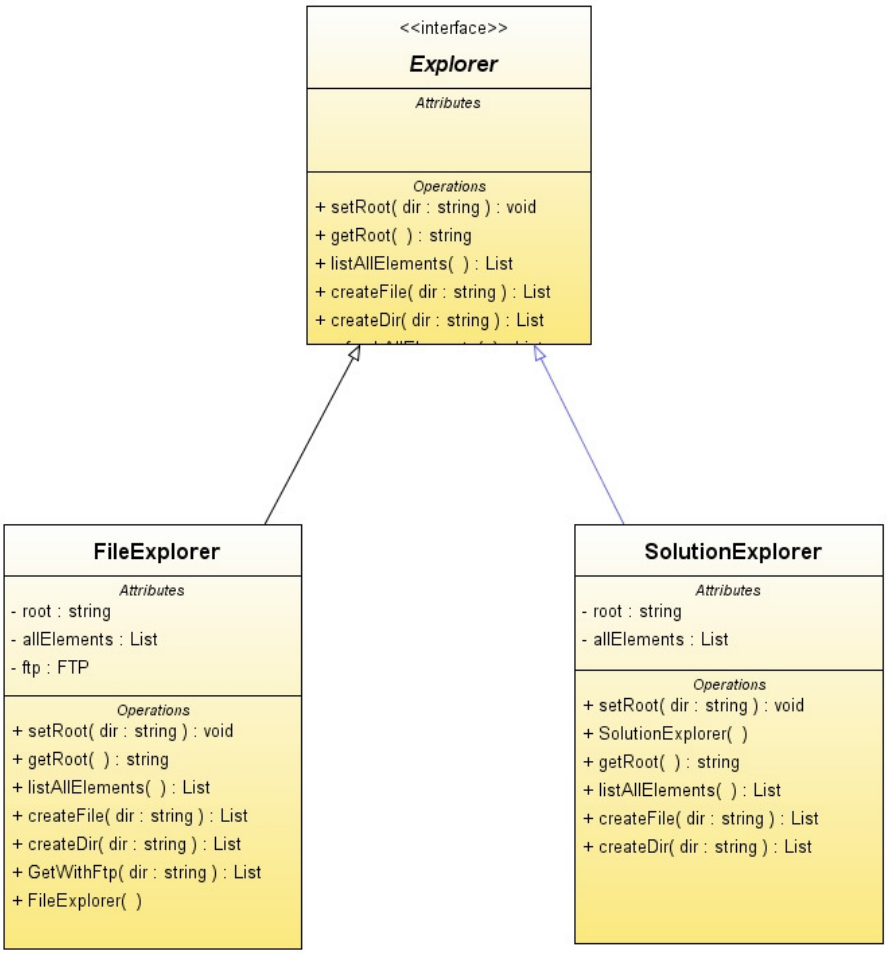
3.2.2.4 USAGE SCENARIO

When the user presses the debug button, active project in the solution explorer is started to be debugged. If the debug process is unsuccessful corresponding errors are seen in the debug window. At this stage user can use step into functions, watch variables and breakpoints properties. If the user chooses the breakpoint property he puts a breakpoint to a specific line in the editor. So the debugger stops at this stage and user can find errors more easily by putting several breakpoints to the editor.

3.2.3 EXPLORER MODULE

3.2.3.1 CLASS DIAGRAM

Our IDE has two Explorer to help the user to bring the files to the editor which they will work on and show the user Project which he works on. To provide these functionalities we have File Explorer and solution Explorer in our program. Below is the class diagram of these explorers. They have very similar functionalities and structure as you can see in the figure. Different from the solution explorer, file explorer imports a ftp object to access the remote files and help the user bring them the environment.



File Explorer Class

Method Name	Type	Arguments	Description of Method
FileExplorer()	Constructor	void	Constructs an FileExplorer object
setRoot(string)	Public:void	Dir:string	Sets a given string as root directory
getRoot( )	Public:string	void	Gets the root directory of the current object
listAllElements()	Public:List	void	Lists all the elements under the root directory
createFile(string)	Public: List	Dir:string	Creates a file under the given

			directory and lists the updated version
createDir(string)	Public: List	Dir:string	Creates a directory under the given directory and lists the updated version
GetWithFtp(string)	Public:List	Dir:string	Retrieves the file via FTP using the address given as a string

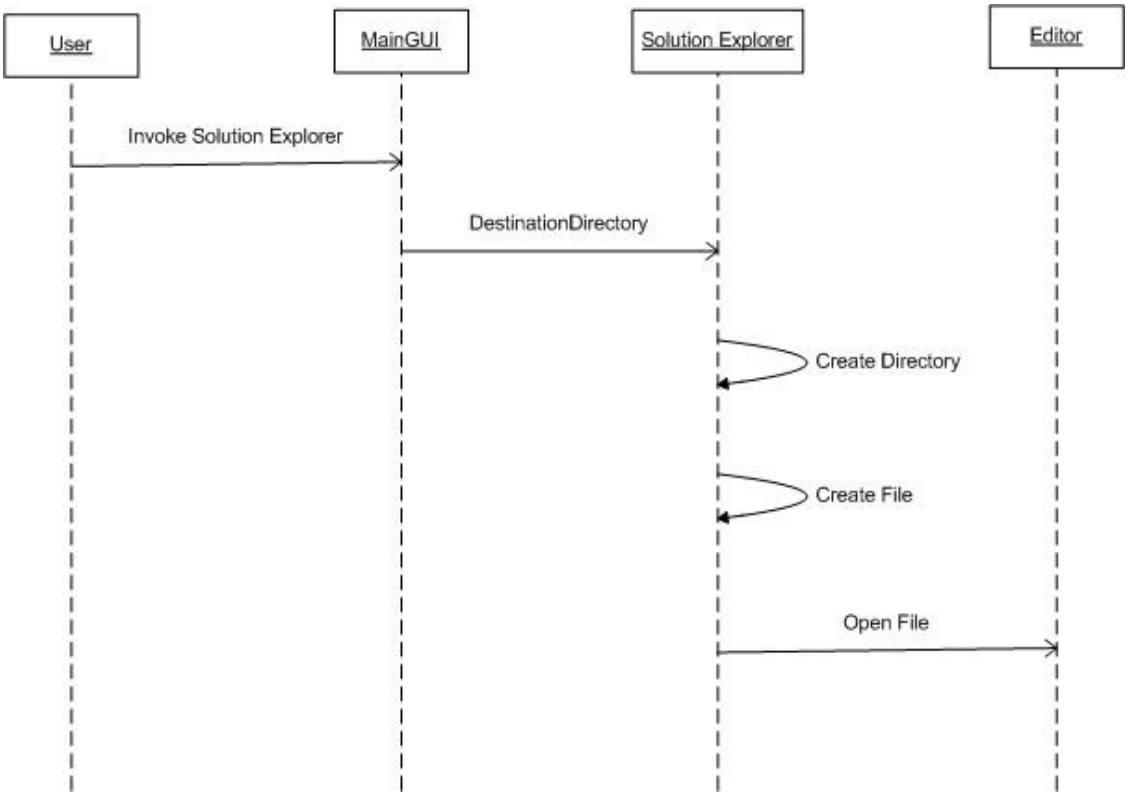
Solution Explorer Class

This Explorer is used to show all files and directories which belong to a Project in the Solution Explorer module.

Method Name	**Type**	**Arguments**	**Description of Method**
SolutionExplorer()	Constructor	void	Constructs an SolutionExplorer object
setRoot(string)	Public:void	Dir:string	Sets a given string as root directory
getRoot( )	Public:string	void	Gets the root directory of the current object
listAllElements()	Public:List	void	Lists all the elements under the root directory
createFile(string)	Public: List	Dir:string	Creates a file under the given directory and lists the updated version
createDir(string)	Public: List	Dir:string	Creates a directory under the given directory and lists the updated version

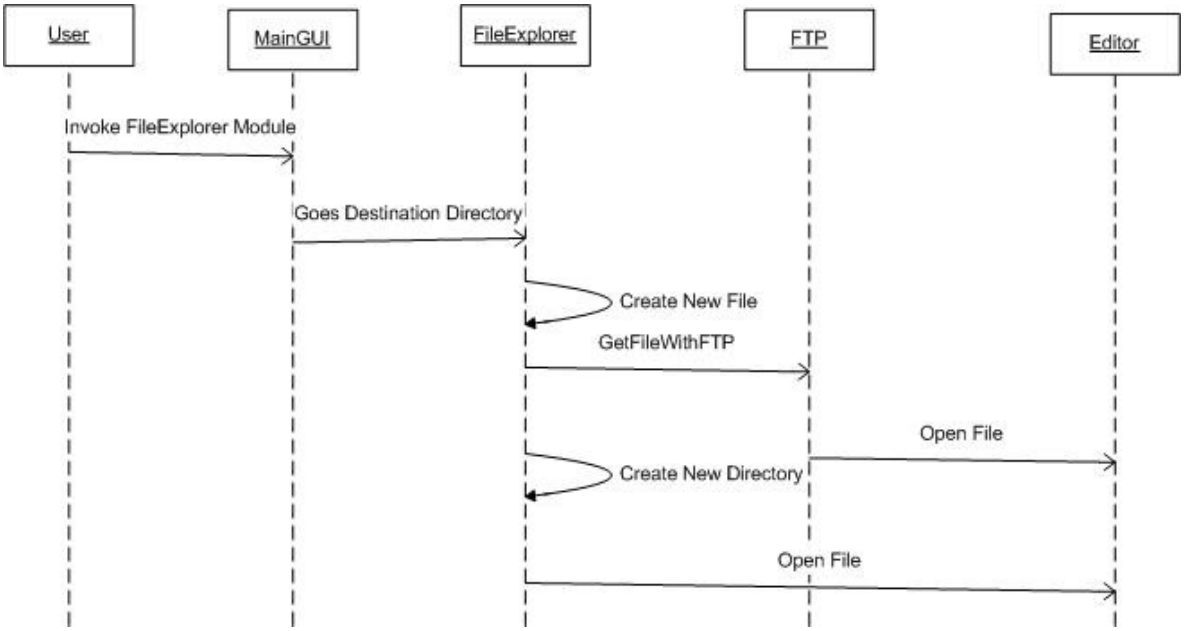
3.2.3.2 SEQUENCE DIAGRAM

Solution Explorer



User goes to a destination directory and then use the functionalities of solution explorer such as create directory, create file. Then user can display the file in the editor.

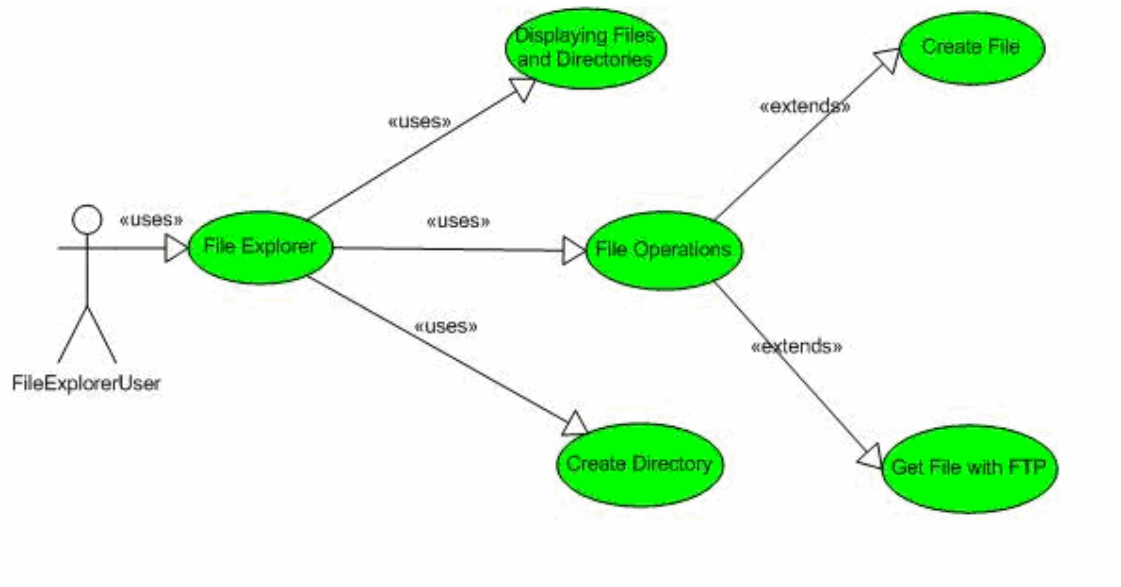
File Explorer



User goes to a destination directory and then use the functionalities of file explorer such as create directory, create file or get with ftp. Then user can display the file in the editor.

3.2.3.3 USE CASE DIAGRAMS

File Explorer



Flow of Events: Browsing a file use case

Objective : To enable the user bringing the file to the development environment.

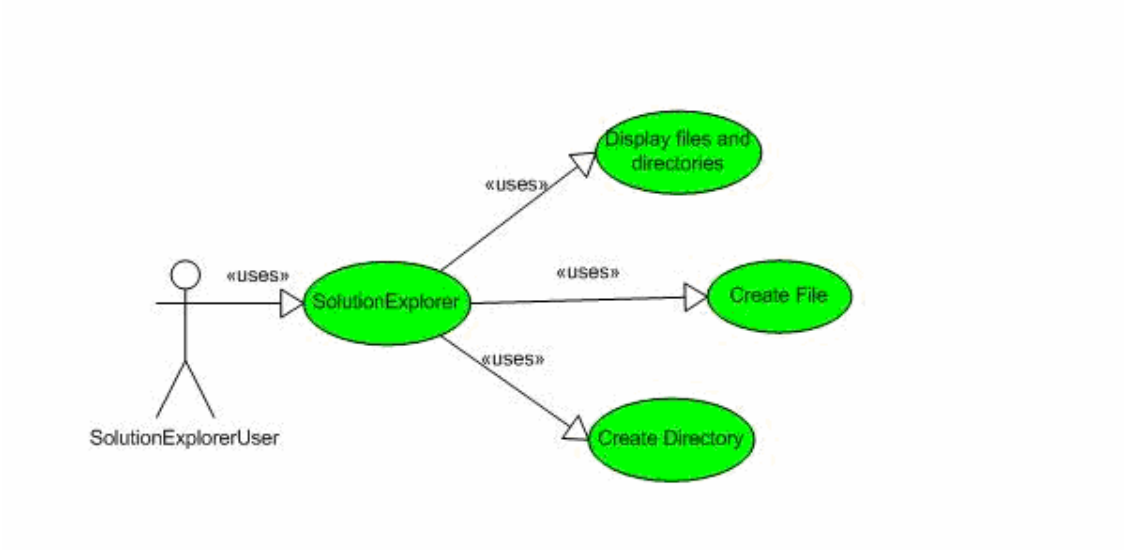
Precondition : No precondition

Main Flow

1. The user interacts with the File Explorer interface
2. The user selects Displaying, creating directories and other file operations.
3. The user can create new file or get the new files using FTP.

Post-condition The user has come to the stage of bringing a file to the editor.

Solution Explorer



Flow of Events: Browsing a Project file use case

Objective : To enable the user bringing the project file to the development environment.

Precondition : No precondition

Main Flow

1. The user interacts with the Solution Explorer interface
2. The user selects Displaying, creating directories and other file operations.

Post-condition The user has come to the stage of bringing a project file to the editor.

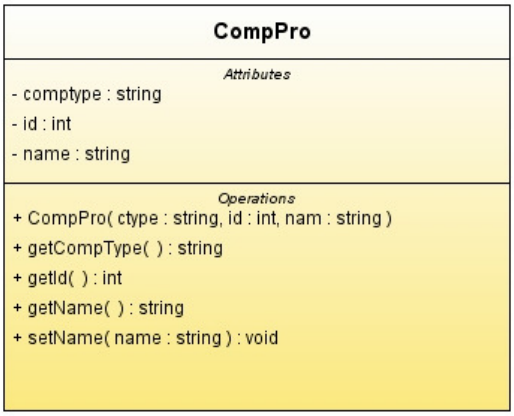
3.2.3.4 USAGE SCENARIOS

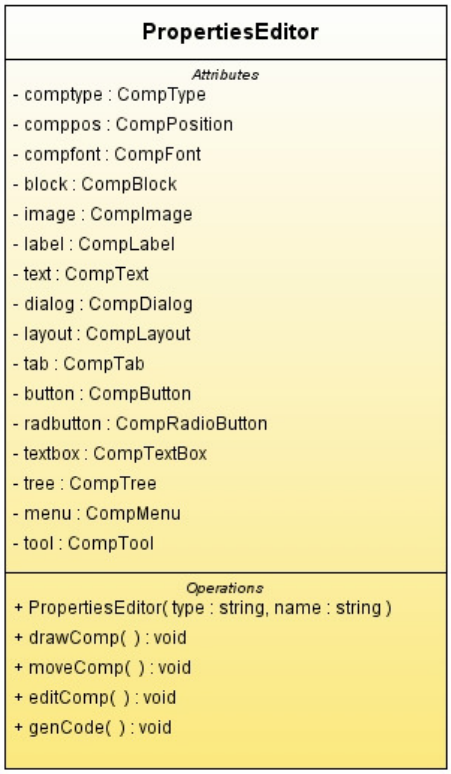
When user opens the file explorer module he will face with the Desktop, C,D and other directories if available of his system. Then he will look for his file or by right clicking the current directory, he will be able to face with a menu containing create file and create directory options. When he selects one of these options, a file/directory creation come. User determines the name and type of file and presses OK button this file is created under the clicked directory and can be seen in the file explorer tree.

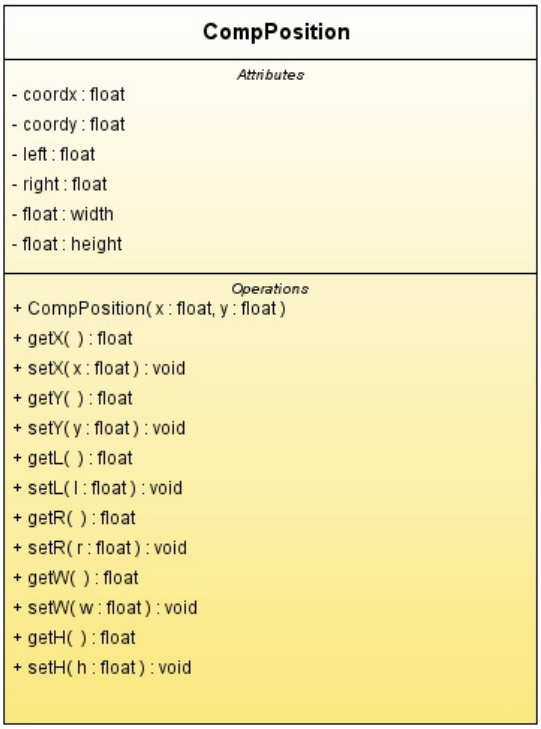
3.2.4 TOOLBOX MODULE

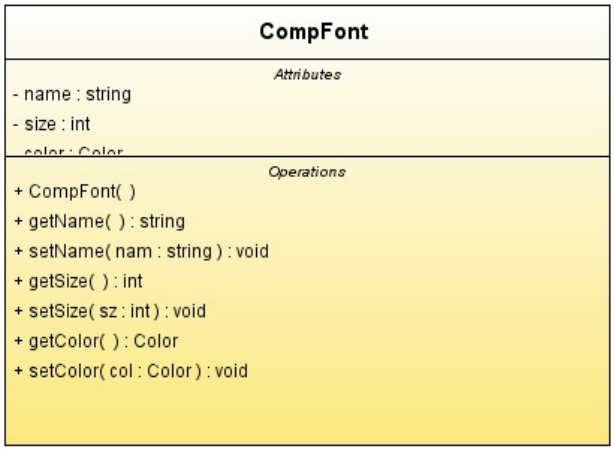
3.2.4.1 CLASS DIAGRAM

This module of our Project is used to make easy the GUI design of the user providing him a drag&drop functionality. User will be able to pick a GUI element from the Toolbox panel, drag and drop it to the design view form. After this he/she will be able to edit the properties(component type, component name, component position etc.) of this GUI element and add/remove events to this element. So we will provide a GUI library composing several kinds of elements which are heavily used today's web applications.
Every component in our Project will have two panels Properties Editor and Event Editor. Let's see which elements a component is composed seeing the class diagram.


CompPro
Attributes
- comptype : string
- id : int
- name : string
Operations
+ CompPro(ctype : string, id : int, nam : string)
+ getCompType( ) : string
+ getId( ) : int
+ getName( ) : string
+ setName(name : string) : void


PropertiesEditor
Attributes
- comptype : CompType
- comppos : CompPosition
- compfont : CompFont
- block : CompBlock
- image : CompImage
- label : CompLabel
- text : CompText
- dialog : CompDialog
- layout : CompLayout
- tab : CompTab
- button : CompButton
- radbutton : CompRadioButton
- textbox : CompTextBox
- tree : CompTree
- menu : CompMenu
- tool : CompTool
Operations
+ PropertiesEditor(type : string, name : string)
+ drawComp( ) : void
+ moveComp( ) : void
+ editComp( ) : void
+ genCode( ) : void


CompPosition
Attributes
- coordx : float
- coordy : float
- left : float
- right : float
- float : width
- float : height
Operations
+ CompPosition(x : float, y : float)
+ getX( ) : float
+ setX(x : float) : void
+ getY( ) : float
+ setY(y : float) : void
+ getL( ) : float
+ setL(l : float) : void
+ getR( ) : float
+ setR(r : float) : void
+ getW( ) : float
+ setW(w : float) : void
+ getH( ) : float
+ setH(h : float) : void


CompFont
Attributes
- name : string
- size : int
Operations
+ CompFont( )
+ getName( ) : string
+ setName(nam : string) : void
+ getSize( ) : int
+ setSize(sz : int) : void
+ getColor( ) : Color
+ setColor(col : Color) : void

CompFont Class

Method Name	Type	Arguments	Description of Method
CompFont()	Constructor	void	Constructs the font of characters initializing them with the default values given by our program
setFont(string)	Public:void	font: string	Sets the font type of component with the given type in the string argument
getFont( )	Public:string	Void	Gets the font type of component
setSize(int)	Public:void	Int : size	Sets the font size of component with the given size in the integer
getSize()	size: int	Void	Gets the font size of component
setColor(Color)	Public:void	col:Color	Sets the font color of component with the given color in the color object
getColor()	Public:Color	Void	Gets the font color of component

CompPosition

Method Name	Type	Arguments	Description of Method
CompPosition()	Constructor	float: x, float: y	Constructs a component in the design view with the given coordinates
setX(float)	Public:void	float: x	Sets the new x coordinate of the component with the given value in the float argument
getX( )	Public:string	Void	Gets the x coordinate of component
setY(float)	Public:void	float : y	Sets the new y coordinate of the component with the given value in the float argument
getY( )	Public: float	Void	Gets the y coordinate of component
setW(float)	Public:void	float: w	Sets the width of component with the given value in the float argument
getW( )	Public:float	Void	Gets the width of component
setH(float)	Public:void	float:h	Sets the height of component with the given value in the float argument
getH( )	Public:float	Void	Gets the height of component
setL(float)	Public:void	Float:l	Sets the left margin of component with the given value in the float argument
getL( )	Public:float	Void	Gets the left margin of component
setR(float)	Public:void	Float:r	Sets the right margin of component with the given

			value in the float argument
getR( )	Public: float	Void	Gets the right margin of component

CompPro Class

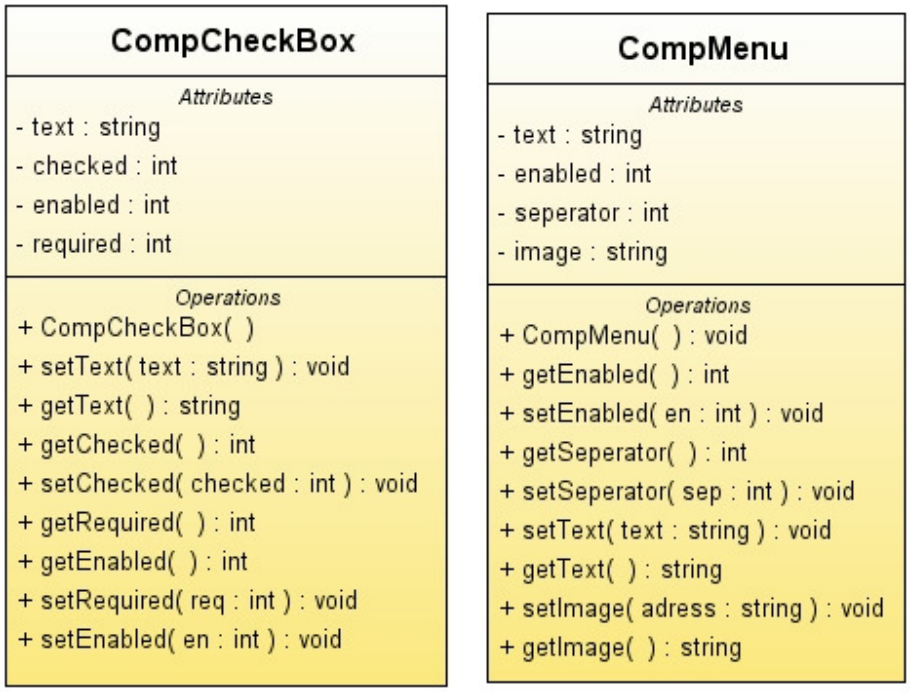
Method Name	Type	Arguments	Description of Method
CompPro(string, int ,string)	Constructor	String : ctype, id: int, string: name	Constructs the component taking its type from the first argument of constructor, second argument is an identifier assigned by our system to identify the components and third argument is name of the component
getId()	Public:int	Void	Gets the id of component
getCompType( )	Public:string	Void	Gets the type of component
setName(string)	Public:void	name:string	Sets the name of component with the given string
getName()	name: string	Void	Gets the name of component

PropertiesEditor Class

Method Name	Type	Arguments	Description of Method
PropertiesEditor(string, string)	Constructor	ctype: string, name:string	Construct the properties editor taking the type and and name
drawComp()	Public:void	Void	Draws the component in the design form
moveComp( )	Public:void	Void	Moves the component to the selected position which is retrieved from GUI

editComp()	Public:void	Void	Edits the component with the adjusted properties retrieving them from GUI
genCode()	Public:void	Void	Generates the code of component with selected properties

As you can see in the class diagram figure, Properties Editor Class has some objects such as button, checkbox, menu etc. These are some component types which are some of the supported by our IDE. Since each of these elements have different kinds of properties peculiar to itself only, we have needed to define classes which are assigned for each component to put the attributes of that component. Below we put two prototype classes CompCheckBox and CompTree which maintains the properties of these components. Since we have 28 types of components having a different class, we did not put the all the classes of these components.

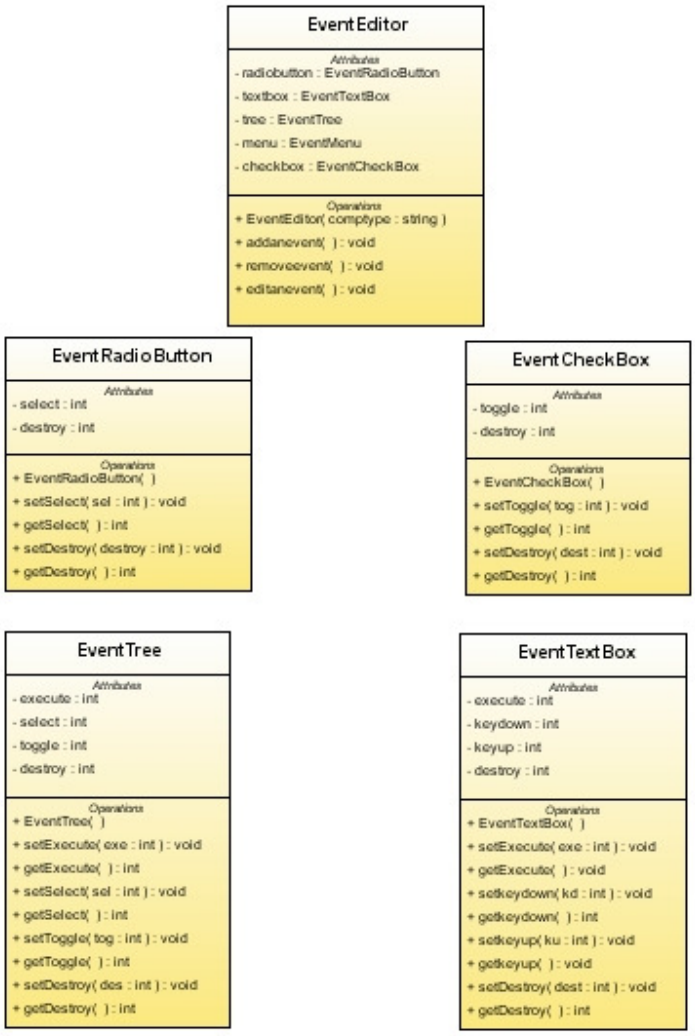


Each attribute of the classes above determines the value of each component that are seen on the GUI and can be changed by the user. All of the components below have such properties.

- **Block Tools:** Block, Image, Label, Text
- **Containers:** Dialog, Layout, Splitter, Stack, Tab, Tabbed Pane
- **Form Element:** Button, Checkbox, Date Picker, Radio Button, Select, Combo, Text Area, Text Box, Time Picker
- **Matrix:** Grid, List, Multi Select, Tree
- **Menus and Toolbars:** Menu, Menu Bar, Task bar, Tool bar, Tool bar Button

EventEditor Class

Since all of the components are event driven, each component`s event property should be determined by the user using the GUI of our program. So we should provide events that a component can support. When we examine the components, we again see that most of the components have different event actions so we again should derive event class of each component. Event classes of some most needed components are below, again we have lots of components having different events, we only put these prototypes.



EventRadioButton Class

This class has the events of radiobutton when the user wants to add an event to the radiobutton, it adjusts this via the GUI and this class forms the environment to fire the event. It has the attributes which provides the event actions supported by the radiobutton. **Select** attribute is fired when the user clicks radiobutton and necessary functions are for the setting and getting this property. **Destroy** attribute is used when the user disables the component so all the events are closed.

EventCheckBox Class

Toggle property is fired when the checkbox state has changed and necessary set and get functions. **Destroy** attribute is used when the user disables the component so all the events are closed.

EventTree Class

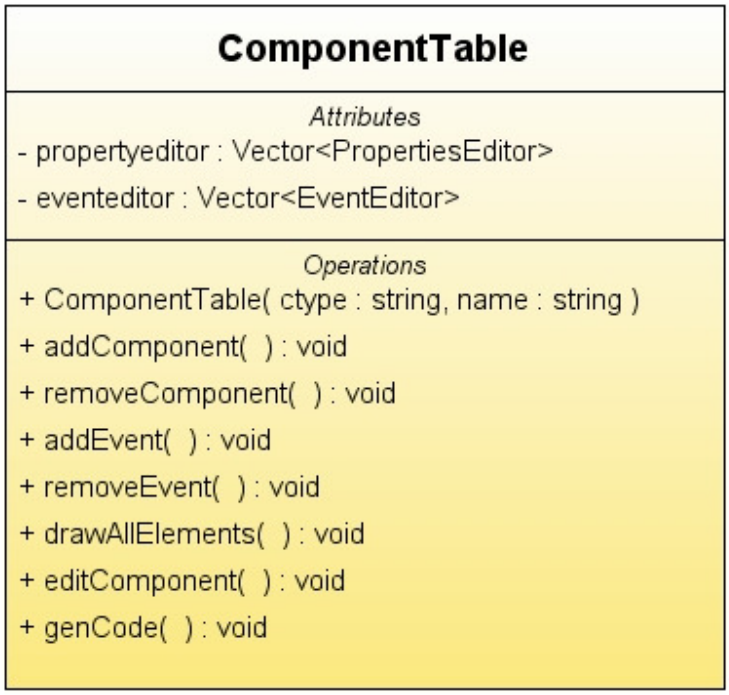
Execute button is used when the user clicks a node of the tree. Selection is fired after the selection has changed. **Toggle** is fired when the node in the tree is toggled. **Destroy** attribute is used when the user disables the component so all the events are closed.

EventTextBox

Execute button is used when the user clicks the textbox. **Keydown** is fired when the user presses the down and keyup button is fired when the user presses the up button. Destroy button is used when the user disables the component so all the events are closed.

ComponentTable Class

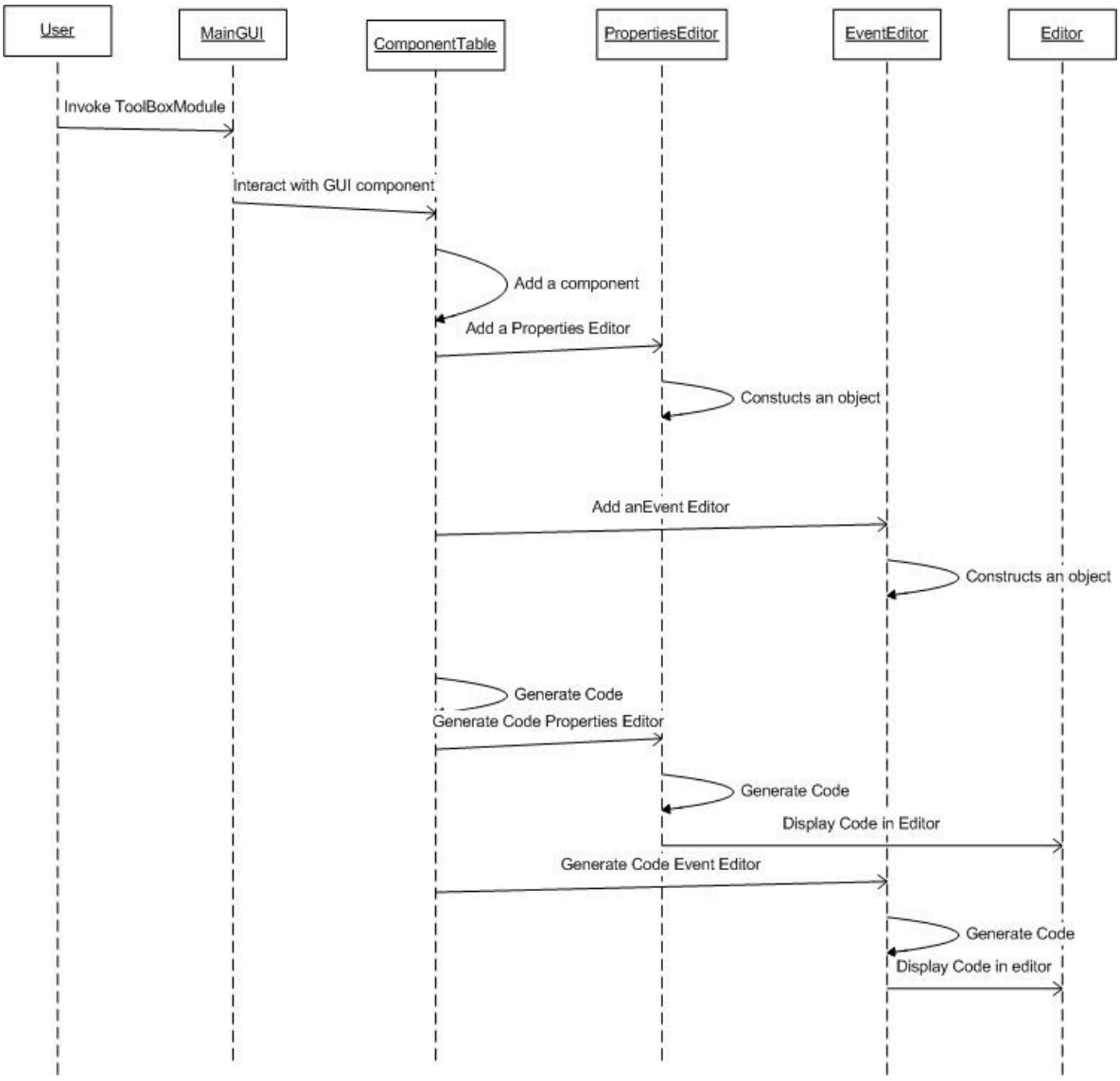
This class is used for the generation of all components in the design form all the operations are done via this class.



Method Name	Type	Arguments	Description of Method
ComponentTable(string,string)	Constructor	void	Constructs a component table for a form in a project
addComponent( )	Public:void	void	Adds a component to the table
removeComponent( )	Public:void	void	Removes the component
addEvent( )	Public:void	void	Adds an event to a component
removeEvent( )	Public:void	Void	Removes the event
drawAllComponents( )	Public:void	void	All the components added till that phase are drawn
genCode( )	Public:void	Void	All the code generation for each component is done
editComponent( )	Public:Void	Void	Edits the component

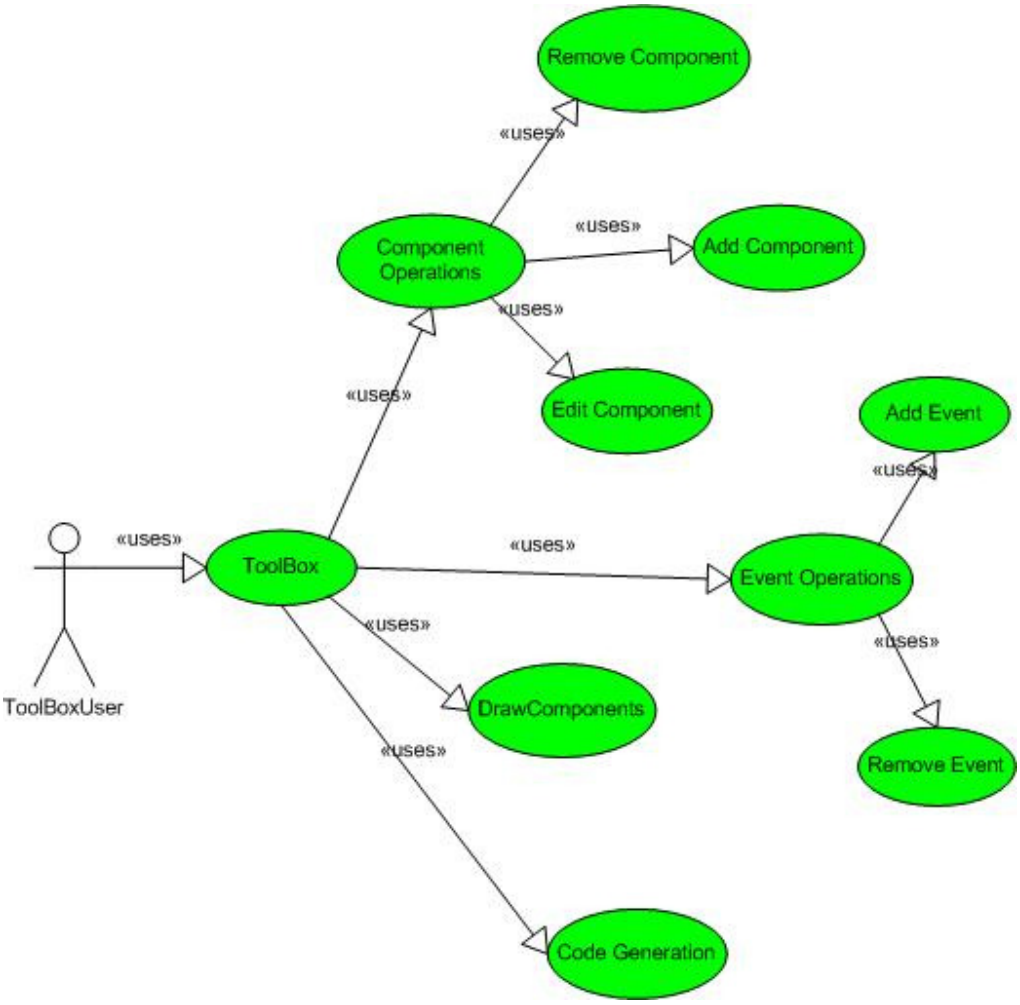
As you see in the diagram above, all the elements which are created until that time by the user held in the data structure vector of Java class with their properties editor and event editor class. All the necessary operations are done via the methods in this class.

3.2.4.2 SEQUENCE DIAGRAM



User interacts with the GUI of our IDE and selects one of the components and adds it to the component table then component table automatically enables the event and properties editor for that component. Code generation is done in the code part of the editor and editor module displays it. The other operations such as remove component works in similar manner.

3.2.4.3 USE CASE DIAGRAM



Flow of Events: GUI Design with Toolbox Use Case

Objective : To enable the user making GUIs easier.

Precondition : No precondition

Main Flow

1. The user drags and drops a component to the design view of the editor.
2. The user has 4 choices: Component Operations, Event Operations, Displaying Component or Code Generation
3. The user can add, remove and edit components if he choices component operations.
4. The user can add, remove events if he choices event operations.

Post-condition The user has done GUI design of the project.

3.2.4.4 USAGE SCENARIO

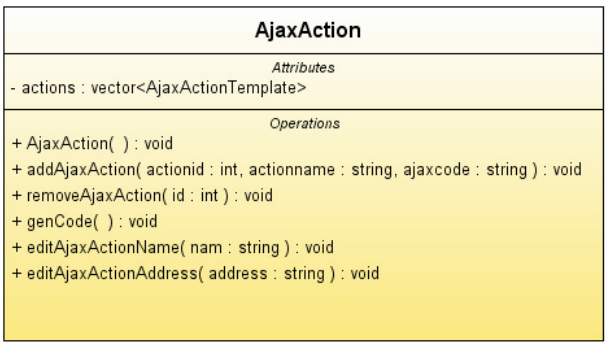
When user wants to do the GUI design of his project, he selects a component from the component table drags and drops that component to the form of our system. Simultaneously, properties editor and events editor panels are enabled in the GUI. Now, user can play with the properties of each component that he dragged and dropped on the from. As soon as the properties of the components are changed, code of these components are automatically changed.

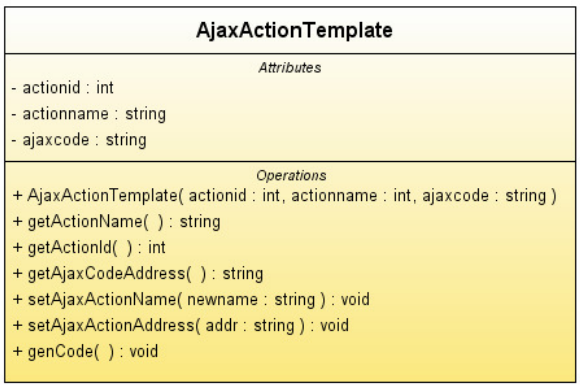
3.2.5 AJAX ACTION MODULE

3.2.5.1 CLASS DIAGRAM

This module enables the user to add new AJAX Action Templates and displaying these templates from the GUI to the user. When the user forms an AJAX action template, he/she can see all the templates that he or she did before and them select one of them and code generation is done in the editor.

As you can see below Ajax Action module of our component is composed of two classes. AjaxActionTemplate class is used to form an AJAX Action Template by the user according to the preferences of him and select one of the templates which are stored in AJAX Action class via the GUI and corresponding code of that template is generated.





AjaxActionTemplate:

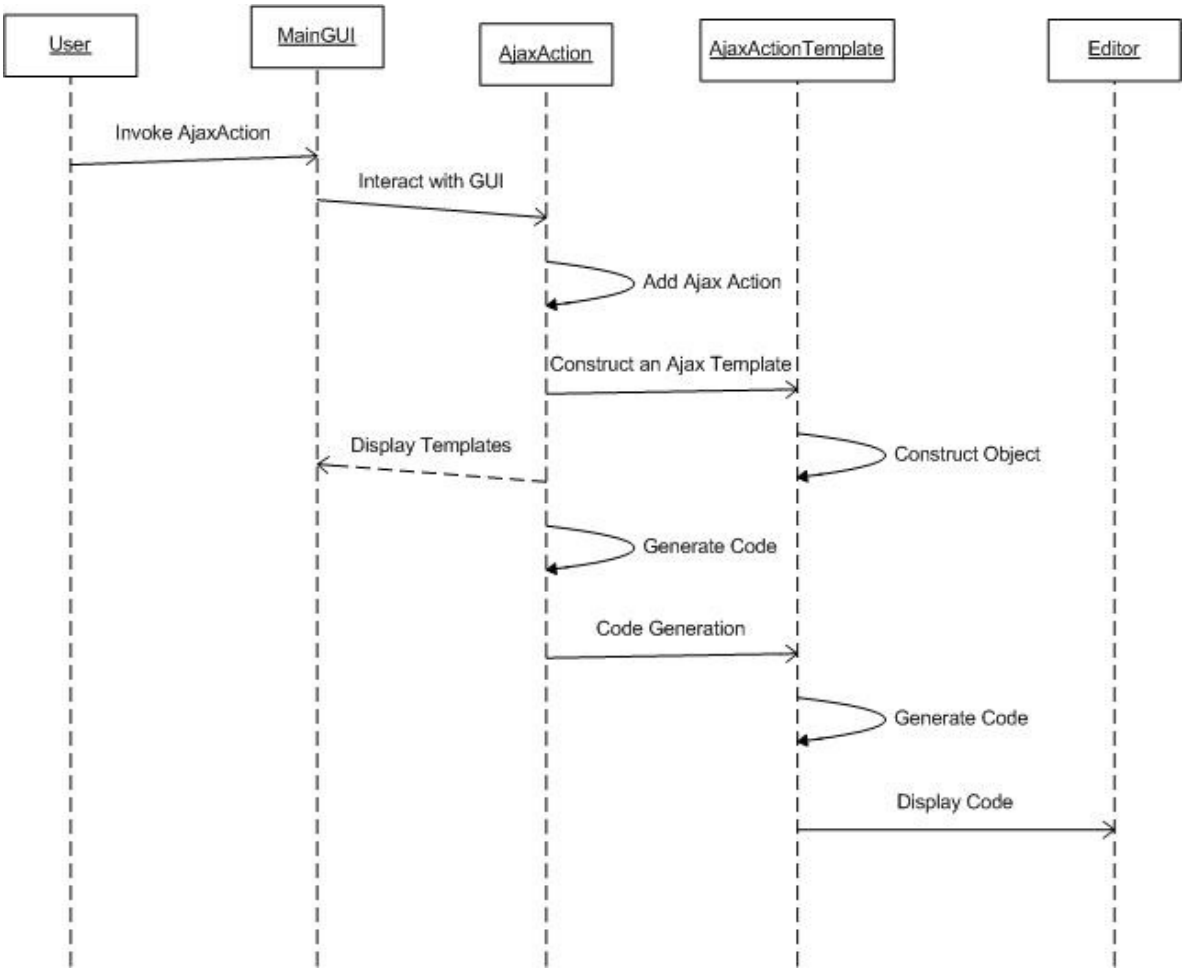
Method Name	Type	Arguments	Description of Method
AjaxActionTemplate(int,int,string)	Constructor	Actionid:int, actionname:int, ajaxcode: string	Constructs an ajax action template
getActionName( )	Public:string	Void	Returns the name of the action
getActionId( )	Public:int	void	Returns the id of action
getAjaxCodeAddress( )	Public:string	void	Returns the path of the ajax action
setAjaxCodeAddress( )	Public:void	String: address	New ajax action is saved another place
setAjaxActionName(string)	Public:void	String: newname	New name is given to the ajax action
genCode( )	Public:void	Void	code generation for the ajax action

AjaxAction:

Method Name	Type	Arguments	Description of Method
AjaxAction( )	Constructor	void	Constructs an ajax action
AddAjaxAction(int,string,string)	Public:void	Actionid:int, actionname:string, ajaxcode:string	Adds an ajax action to data structure using AjaxActionTemplate class
RemoveAjaxAction(int)	Public:void	void	Removes the ajax action from the data structure
EditAjaxCodeAddress(	Public:void	String:adress	New ajax action is

)			saved another place
EditAjaxActionName(string)	Public:void	String: newname	New name is given to the ajax action
genCode( )	Public:void	Void	code generation for the ajax action

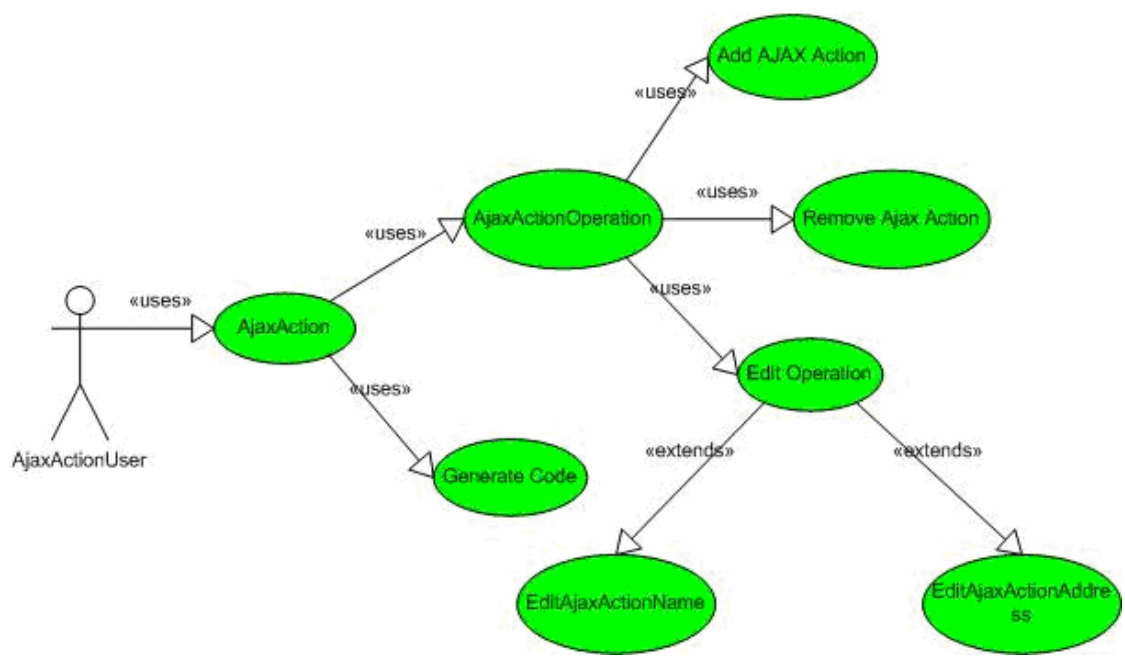
3.2.5.2 SEQUENCE DIAGRAM



User interacts with the Ajax Action panel part of our GUI and sees the choices of adding a new Ajax action or displaying available actions. When a new Ajax action is defined construction AjaxActionTemplate class called. However when the user selects

displaying an Ajax Action, corresponding Ajax Action code is generated and it is displayed in the editor by the editor module.

3.2.5.3 USE CASE DIAGRAM



Flow of Events: Joining Ajax Actions to the source code

Objective : To enable the user adding Ajax Action to the project.

Precondition : No precondition

Main Flow

1. The user interacts with the Ajax Action GUI

2. The user has 2 choices: Selecting an available Ajax Action or Operating on an Ajax action

3. The user can add, remove and edit Ajax Actions if he choices AjaxAction operations.

4. The user can give an AjaxAction new name and change the path of it.

Post-condition Ajax Actions are added to the project.

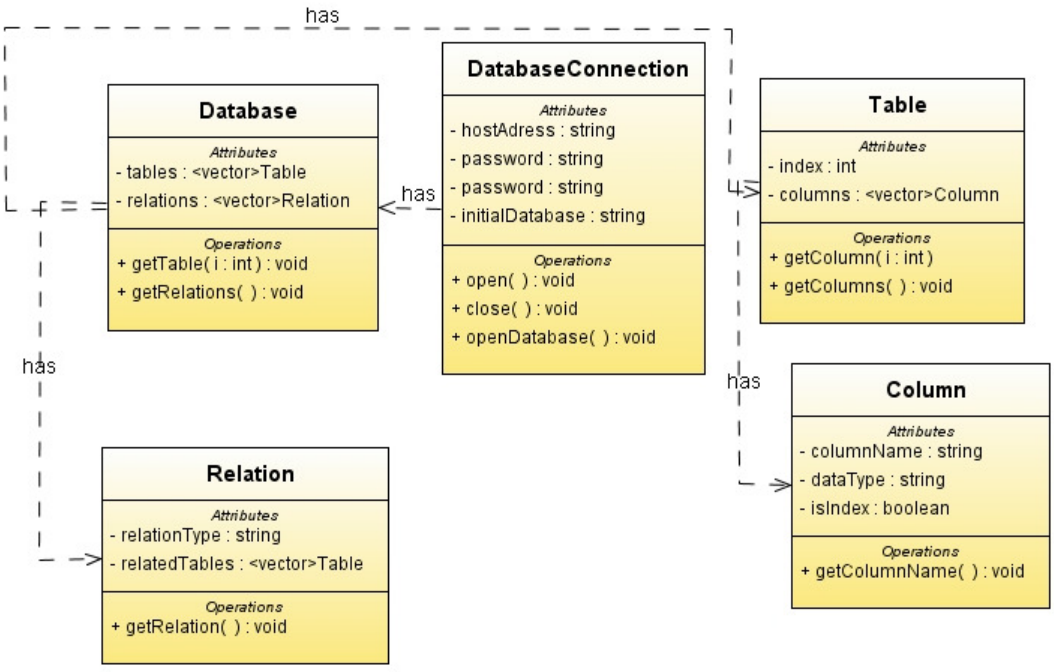
3.2.5.4 USAGE SCENARIO

When user wants to add a AJAX action to his program, he chooses AJAX Action Panel from the GUI and selects adding a new option to the program or wants to use his old actions. If he chooses his old actions, corresponding AJAX code is generated in the editor of our program.

3.2.6 DATABASE MODULE

User can connect database using connection wizard of our database module. Connection wizard asks user some information about database, like database name, place, user name and password to access database. After connection user can see the tables in database and relationships of those tables. Our database module lets user to query the database and the results can be used easily in project.

3.2.6.1 CLASS DIAGRAM



Here is the definition of Database:

Database Class:

Method Name	Type	Arguments	Description of Method
getTable(int)	Public:void	i:int	Gets the table with the given index i.
getRelations()	Public:void	void	Gets the relations in the database.

DatabaseConnection Class:

Method Name	Type	Arguments	Description of Method
open()	Public:void	void	Opens the connection to the database server.
close()	Public:void	void	Closes the connection after database operations have finished.
openDatabase()	Public:void	void	Opens the database in the server.

Table Class:

Method Name	Type	Arguments	Description of Method
getColumn(int)	Public:void	i:int	Gets the column with the given index i.
getColumns()	Public:void	void	Gets all the columns in that table.

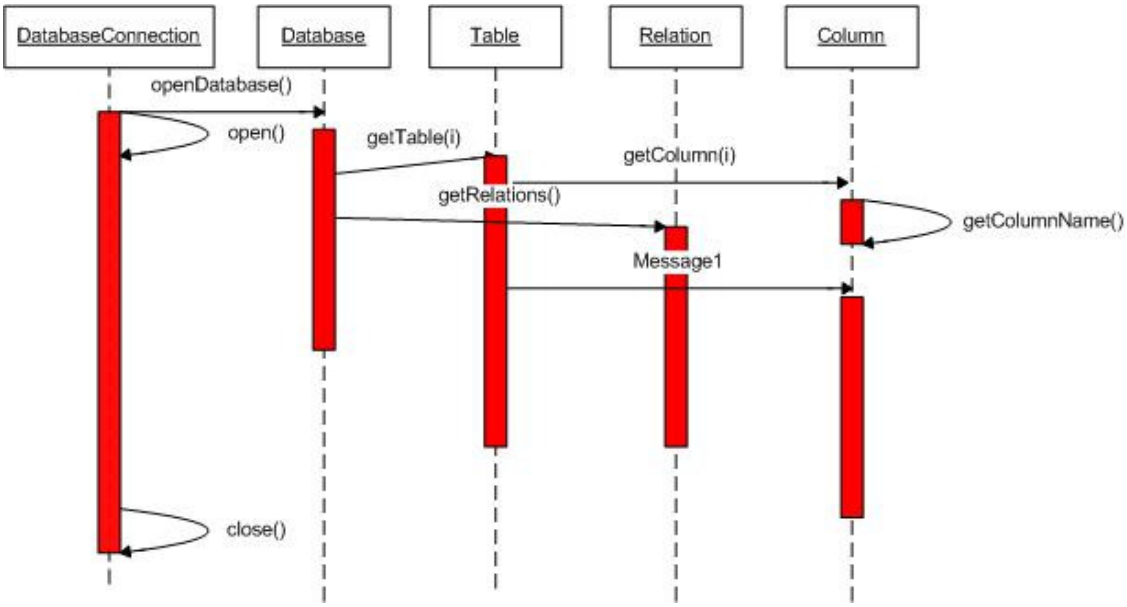
Column Class:

Method Name	Type	Arguments	Description of Method
getColumnName()	Public:void	void	Gets the name of the column.

Relation Class:

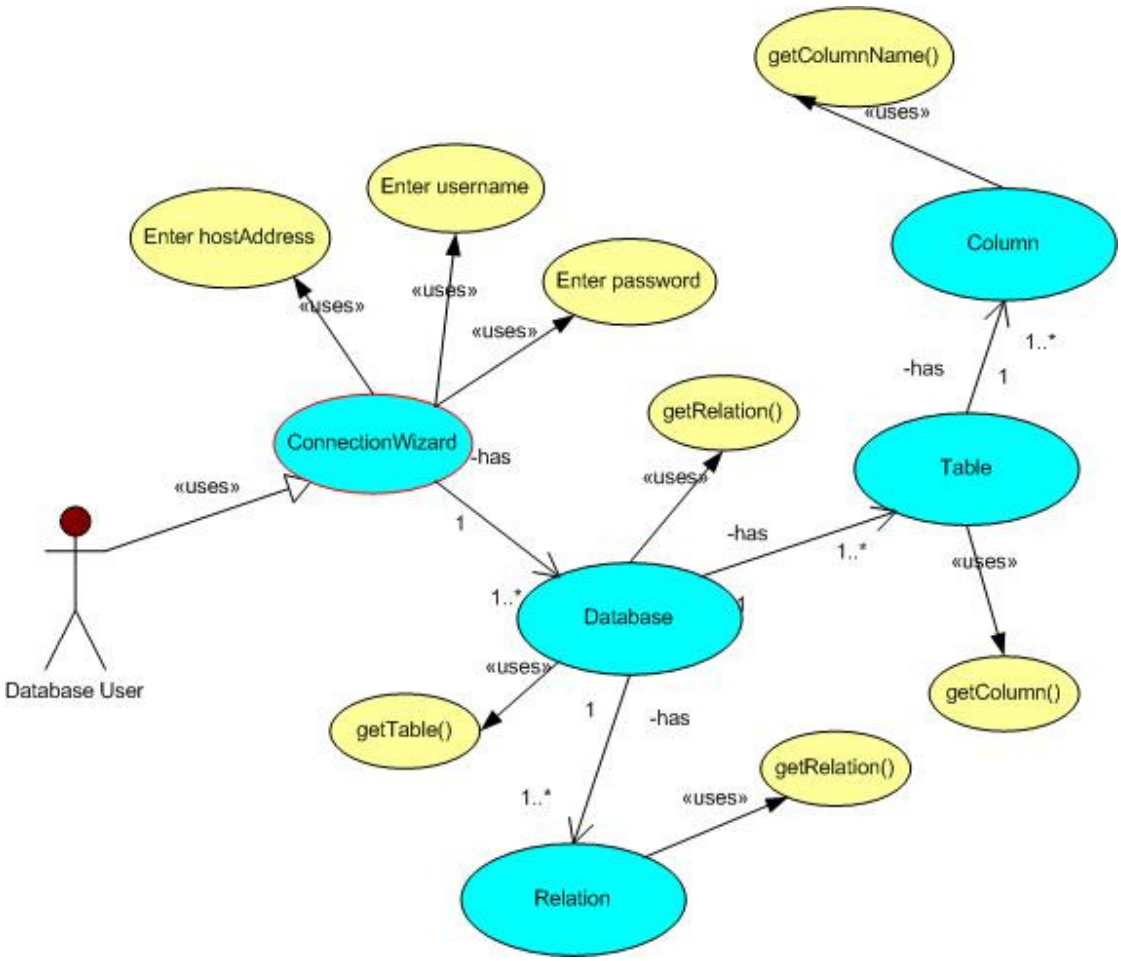
Method Name	Type	Arguments	Description of Method
getRelations()	Public:void	void	Gets the relations in the database.

3.2.6.2 SEQUENCE DIAGRAM



The operations of database module can be seen above in a sequence. Firstly, user opens a database connection from the server using the connection wizard. Then, opens the database and can do the required operations such as retrieving data, viewing relations. After these operations user closes the connection.

3.2.6.3 USE CASE DIAGRAM



Flow of Database Use Case
Objective : To allow the user do database operations.
Precondition : An open project and a database server.
Main Flow

1. The user connects the database server using database connection wizard.
2. The user can see the tables in the connected database.
3. The user can see the table columns in the connected database.
4. The user can see the relations in the connected database.
5. The user closes the database connection.

Post-condition The user has done the database operations of the project.

3.2.6.4 USAGE SCENARIO

User can connect database, retrieve data from database and see the relations via the database module of **kajax**. Usage scenarios of these operations are below.

Usage Scenario 1 (Connect to Database):

-- In the main GUI user selects the "Connection Wizard" under the menu bar->Database.

-- In the Connection wizard user enters "Host Address", "username", "password" and clicks "Next->" button.

-- If the connection successfully established user can see the databases and manage them.

-- Else "Connection Failure Message" message will be displayed.

Usage Scenario 2 (Retrieve Data from Database):

-- After successful connection to the database user can see the tables in that database.

-- After clicking the desired table user can see the "columns" of that table.

-- User can write the "SQL statement" from "SQL View" of database module.

-- After writing SQL statement user clicks the "Execute" button located in the bottom right corner of the screen.

-- If the statement syntactically correct query result will be displayed.

-- Else error message will be displayed.

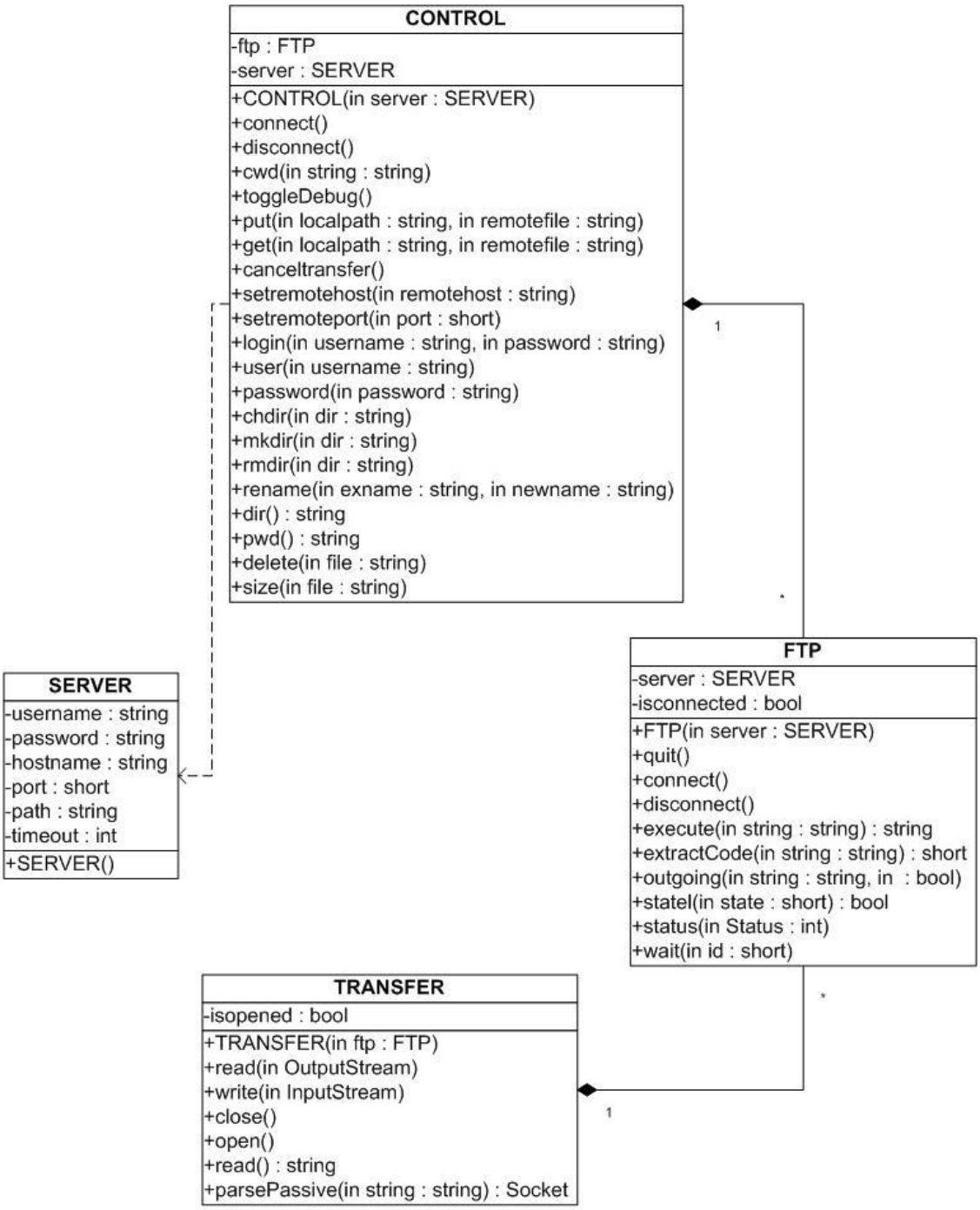
3.2.7 FTP MODULE

Ftp Module of the **kajax** provides the user to use the ftp functionality. In brief, **FTP** (File Transfer Protocol) is a well-documented Internet protocol. It is used for transferring files across networks using **TCP** (Transmission Control Protocol).

A client program and server program are required for it. FTP Client can retrieve files from the server or can upload files to server. **kajax'**s ftp module will act as FTP Client. Ftp module will initiate commands to open a TCP control connection to the server which is used for sending commands and to read servers replies. That control connection is used for the whole session and ends with the **kajax'**s quit command. **kajax** uses transient TCP connections when a file is to be transferred. After file is transferred, the data connection is closed but the control connection is available. **kajax**'s Ftp module

works in active mode so that server has to actively connect to **kajax.** The address and the port number to be listened are specified. Our Ftp Module supports as transfer mode both ASCII and binary modes. **kajax** will used universally decided command standards for its command messages. Most servers require authentication. If authentication is required, server will request this by code upon initial TCP communication. At this point, connection between the Transport layers is successful, and now the Application layers are performing the authentication. Servers can accept anonymous connections, or refuse them all together. These authentication discussions will be important in out FTP Client.

3.2.7.1 FTP CLASS DIAGRAM



kajax's FTP Client consist of 4 Classes as shown above. FTP main class uses CONTROL which also has a connection with SERVER class and TRANSFER classes for

its functionalities. To state the concept of a single command connection and multiple data connections, CONTROL class will be implemented which will be responsible for whole connections to a server. It produces transfer and authentication threads as triggered by its methods. It controls all of the underlying commands, and does not require any knowledge of FTP for the person calling it. FTP class has states associated with it. As the state changes, it updates registered observers with the new so that overall usability of **kajax**'s FTP Client increases.

Some of the important functions e are explained below and class definitions.

The **login ()** is used for logging into a remote FTP account with using a user name and password. **user()** and **password()** methods does the same thing. Since a server can require no password, these methods are given separately. Current Ftp is terminated by **quit()** method.

kajax provides a number of methods for remote directory management. Current directory is obtained by **pwd()** command. **chdir()** changes the remote directory to the determined one. User creates a new directory by **mkdir()** and **rmdir()** is used for deleting a directory. **rename()** method is used for renaming the files and directories. Files can be deleted by **delete()** command. The list of the files are retrieved by **dir()** method as an array of string.

kajax uses **put()** and **get()** to put files into remote server and getting them. Transfer commands can be cancelled by the **canceltransfer()** method.

Since **kajax**'s Ftp Module works in active mode **setremotehost()** and **setremoteport()** methods are used for specifying remote host and port of the FTP server. Here is the definition of FTP:

FTP Class:

Method Name	Type	Arguments	Description of Method
FTP(SERVER)	Constructor	void	Constructs a FTP object with initializing a SERVER object and isconnected boolean.
quit()	Public:void	void	Quits the **kajax**'s FTP Client
connect()	Public:void	void	Triggers Control Class for establishing a new connection
execute(String)	Public:string	string	Triggers Control Class for executing a control operation which returns a result
extractCode(String)	Public:short	string	A helper function for extracting codes from a given string which returns a short integer.
outgoing(String,Boolean)	Public:void	String, boolean	A helper function for transferring operations.
statel(short)	Public:bool	state	A function for querying the state of the FTP Client.
status(Status)	Public:void	void	Determines the status of the FTP Client.
wait(Int)	Public:void	id	Wait is used for thread mechanism of the FTP client.

CONTROL Class:

Method Name	Type	Arguments	Description of Method
CONTROL(SERVER)	Constructor	server	Constructs a Control object for controlling operations
connect()	Public:void	void	Triggers Server connections
disconnect()	Public:void	void	Triggers Server disconnections
cwd(String)	Public:void	string	Gets the path of the current working directory
toggleDebug()	Public:void	void	Used for debugging operations of the FTP Client control commands
put(String,String)	Public:void	localpath,remotefile	Puts the file to the remote server
get(String,String)	Public:void	localpath,remotefile	Gets the file from remote server to the local file explorer
cancelfiletransfer()	Public:void	void	Cancels the current file transaction.
setremotehost(String)	Public:void	remotehost	Used for setting the remote host
setremoteport(Short)	Public:void	port	Used for setting the remote port
login(String,String)	Public:void	Username, password	Used for logging into a remote server
user(String)	Public:void	username	Used for logging a remote server by using only username

password(String)	Public:void	Password	Used for logging a remote server by using only password
chdir(String)	Public:void	dir	Changes directory to the given one
mkdir(String)	Public:void	dir	Makes given directory in the current path
rmdir(String)	Public:void	dir	Removes the given directory
rename(String,String)	Public:void	exname, newname	Renames the given file
dir()	Public:String	void	List the files at the given path
pwd()	Public:String	void	List the files at the current working directory
delete(String)	Public:void	file	Deletes the file with the given name
size(String)	Public:void	file	Returns the size of the file with the given name

TRANSFER Class:

Method Name	Type	Arguments	Description of Method
TRANSFER(FTP)	Constructor	ftp	Constructs a TRANSFER object with given FTP object and initializing the boolean isopened
read(OutputStream)	Public:void	outstream	Reads the outputstream of a remote server
write(InputStream)	Public:void	inputstream	Writes the inputsream of a remote server
close()	Public:void	void	Closes the current file or

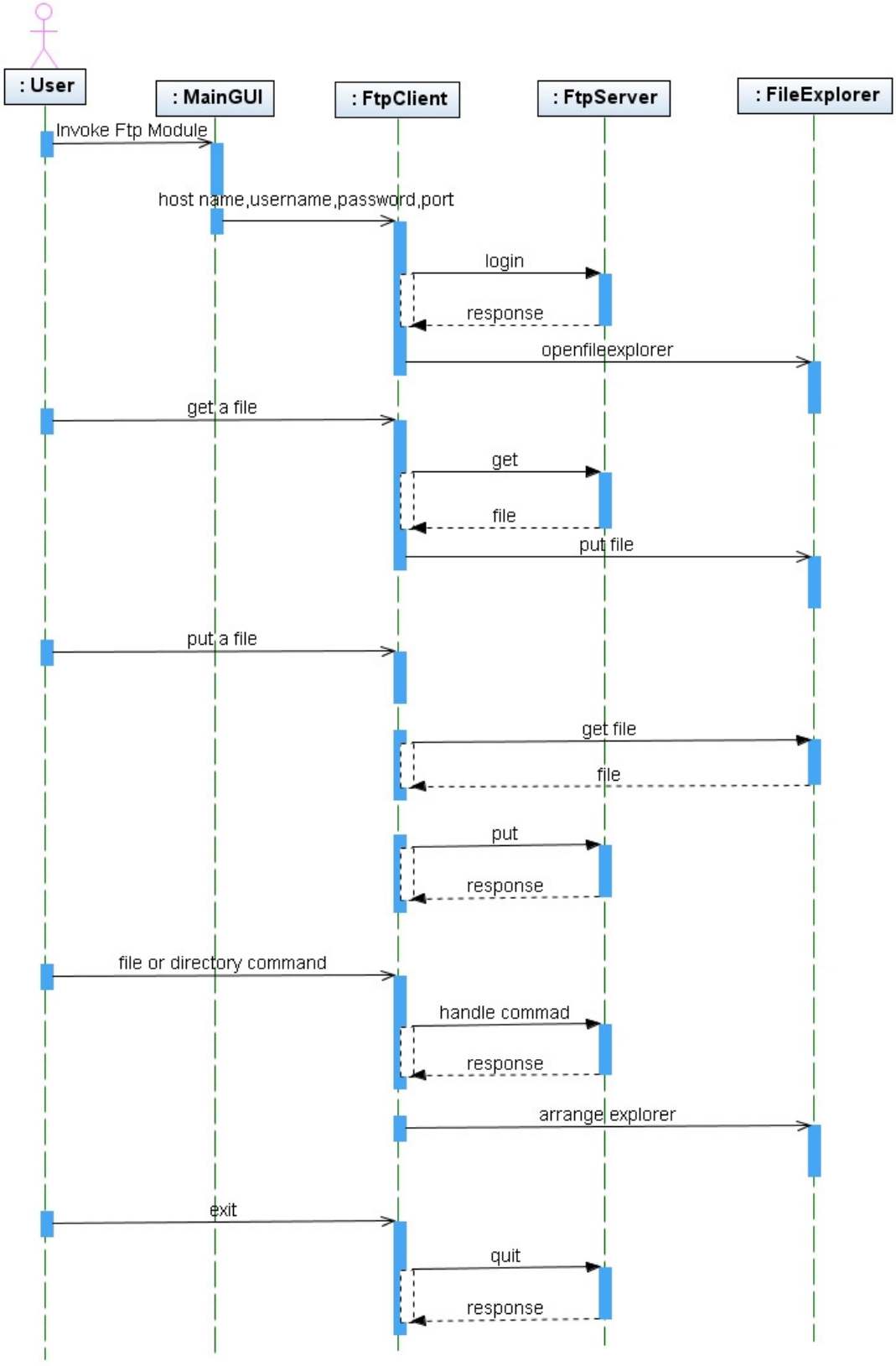
			stream
open()	Public:void	void	Opens a file or stream for a transfer
read()	Public:String	string	Reads a file and return a string of it
parsePassive(String)	Public:Socket	socket	By parsing passive returns the socket for the data connections

SERVER Class:

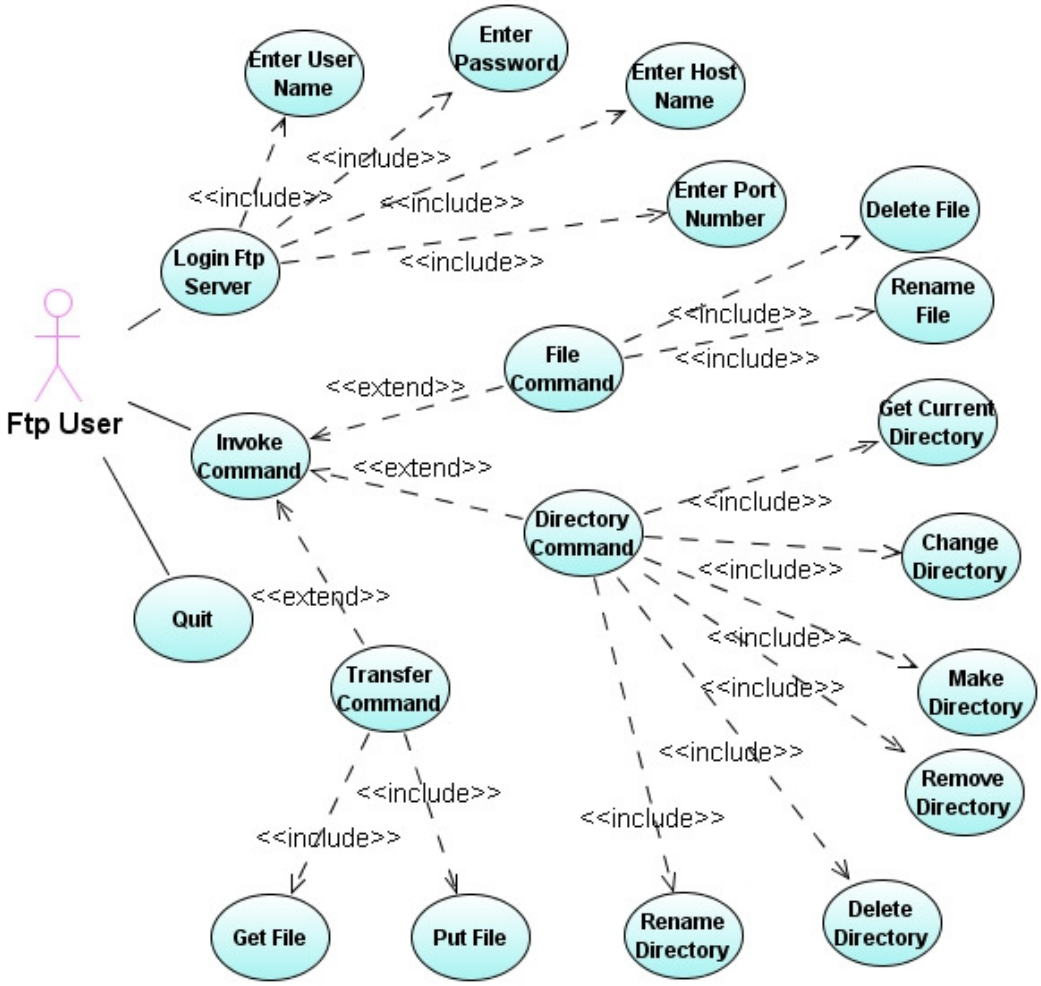
Method Name	Type	Arguments	Description of Method
Server()	Constructor	void	Constructs a Server object with initializing username ,password, hostname, port, path and time out

3.2.7.2 FTP SEQUENCE DIAGRAM

In the following sequence diagram, we have shown login diagram, putting a file diagram, getting a file diagram, invoking file or directory command diagram and log out diagram in the same sequence diagram.


: User
: MainGUI
: FtpClient
: FtpServer
: FileExplorer
Invoke Ftp Module
host name,username,password,port
login
response
openfileexplorer
get a file
get
file
put file
put a file
get file
file
put
response
file or directory command
handle commad
response
arrange explorer
exit
quit
response

3.2.7.3 FTP USE CASE DIAGRAM



Flow of Events FTP Use Case
Objective: To allow the user to use FTP functionalities.
Precondition: A connection string has to be ready to connect a FTP server.
Main Flow
1. The user interacts with the FTP interface
2. .The user selects one of the Login FTP Server, Invoke Command and Quit operations.
3. The user can enter username, enter password, enter hostname, and enter port number to connect an FTP Server
4. The user can invoke File, Directory or Transfer commands.
5. User can delete file or rename a file as a result of File command.
6. User can get file or put file as a result of Transfer command.

7. User can get current directory, change directory, make directory, remove directory, delete directory or rename a directory as a result of a Directory command.
Post-condition The user has connected an FTP server and makes FTP operations.

3.2.7.4 FTP USAGE SCENARIOS

Scenario 1:

To change a directory client sends " > chdir dirname " command. The server sends "Command Successful "as a response. From this simple message we now that the command is completed. However if we attempt to change directory to absent directory with a command like " > chdir absentdir" we will get from the server an error message something like "The System can not find the directory ". From the code of this response message we understand that our operation is failed. And if unless the specified directory is created, our command fails always.

Scenario 2:

To transfer a text file, a "put" command is given to the server. For transferring that file a data connection is to be setup. Since our Ftp Client supports active mode, we have to use "setremoteport" and "setremotehost" commands to set up active mode. If this is a correct settings the server will response with a message something like "Command is successful". From the code number of this code we understand that the data connection is established. After setting up the data connection file can be transferred with "> put acp.txt". Again server response with "opening data connection". The code number of this command implies that it is successful. Client now has to be waiting for the server, until server gives "transfer completed" command. After that command file transfer is completed and the client can give any other commands.

3.2.8 CVS MODULE

CVS (Concurrent Versions System) is a version control system. By CVS developers can keep history of revisions, comment and keep track of changes. For team members it is an indispensable tool. For that reason, we have decided to provide a CVS Client for **kajax.**

The main properties of the **kajax**'s CVS Client will be the followings:

A connection with the CVS server will be established by using **"server"** way which is a kind of connection ways. Our connection method will provide a mechanism for receiving inputs/outputs for communicating with the server. For this aim, connection method setups input and output streams. By the way security of the connection is important. We will not provide any security over the password.

Communication with the server will be established by Request and Response methods, since each command requires several request of client and several response of server.

Command methods are provided which are necessary for accomplishing main properties of a CVS command line client. The commands that are to be implemented are followings:

add, checkout, commit, update, tag, diff, log, remove, status.

As we have mentioned before, these commands will require several request from the CVS server. These requests will be implemented.

Request method use data and send it to the server using output stream. On the other hand Response method read data from the CVS server. After reading the data, it verifies what is expected by the server and depending on that decision performs related functions. **kajax**'s CVS Client will support character data and bytes for file transmissions.

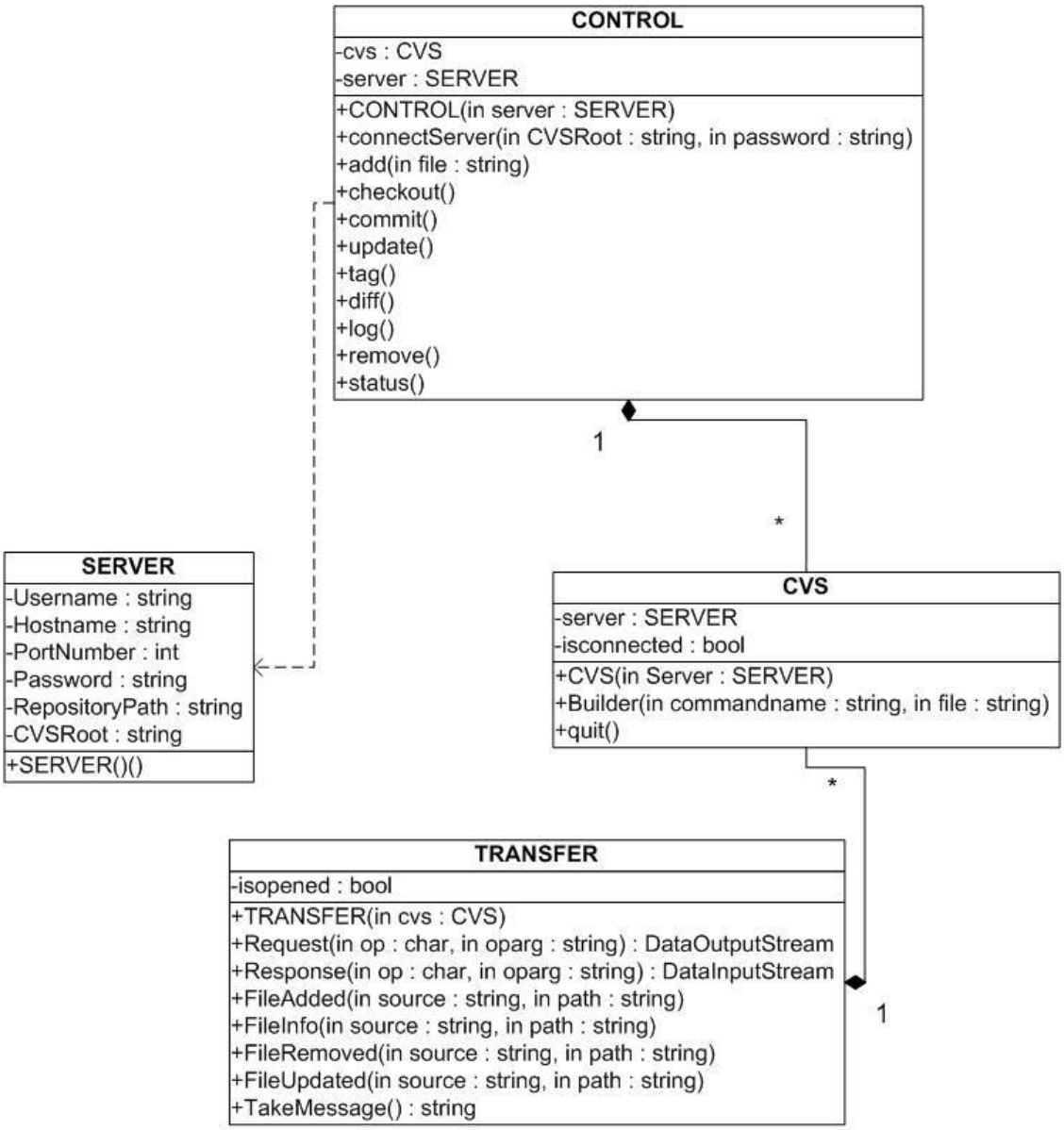
We will provide an Admin method for handling the information of the administrative details.

Since a CVS Client has to be informed by the changes. For this aim **kajax**'s CVS Client provides FileAdded for showing that a file is added. FileInfo informs developer that a structure containing some data has been completed. If a file has been removed FileRemoved method is invoked. FileUpdated method indicates that an existing file has been updated. TakeMessage method is used when a message from CVS server comes.

Builder method is used for parsing the output of the commands which returns a lot of structured data.

Actually a complete CVS Client is a more complicated one. But our **kajax**'s CVS Client module will provide only basic properties of a CVS Client.

3.2.8.1 CVS CLASS DIAGRAM



Here is the definition of CVS Class Diagram:

CVS Class:

Method Name	Type	Arguments	Description of Method
CVS(SERVER)	Constructor	server	Constructs a CVS object with initializing a SERVER object and isconnected oolean.

Builder(String,String)	Public:void	commandname, file	Triggers CONTROL or TRANSFER methods
quit()	Public:void	void	Quits **kajax**'s CVS Client

CONTROL Class:

Method Name	Type	Arguments	Description of Method
CONTROL(SERVER	Constructor	server	Constructs a Control object for controlling operations
connectServer(String,String)	Public:void	CVSRoot, Password	Connects to specified remote CVS server
add(String)	Public:void	file	Executes CVS's add operation
checkout()	Public:void	void	Executes CVS's checkout operation
commit()	Public:void	void	Executes CVS's commit operation
update()	Public:void	void	Executes CVS's update operation
tag()	Public:void	void	Executes CVS's tag operation
diff()	Public:void	void	Executes CVS's diff operation
log()	Public:void	void	Executes CVS's log operation
remove()	Public:void	void	Executes CVS's remove operation
status()	Public:void	void	Executes CVS's status operation

TRANSFER Class:

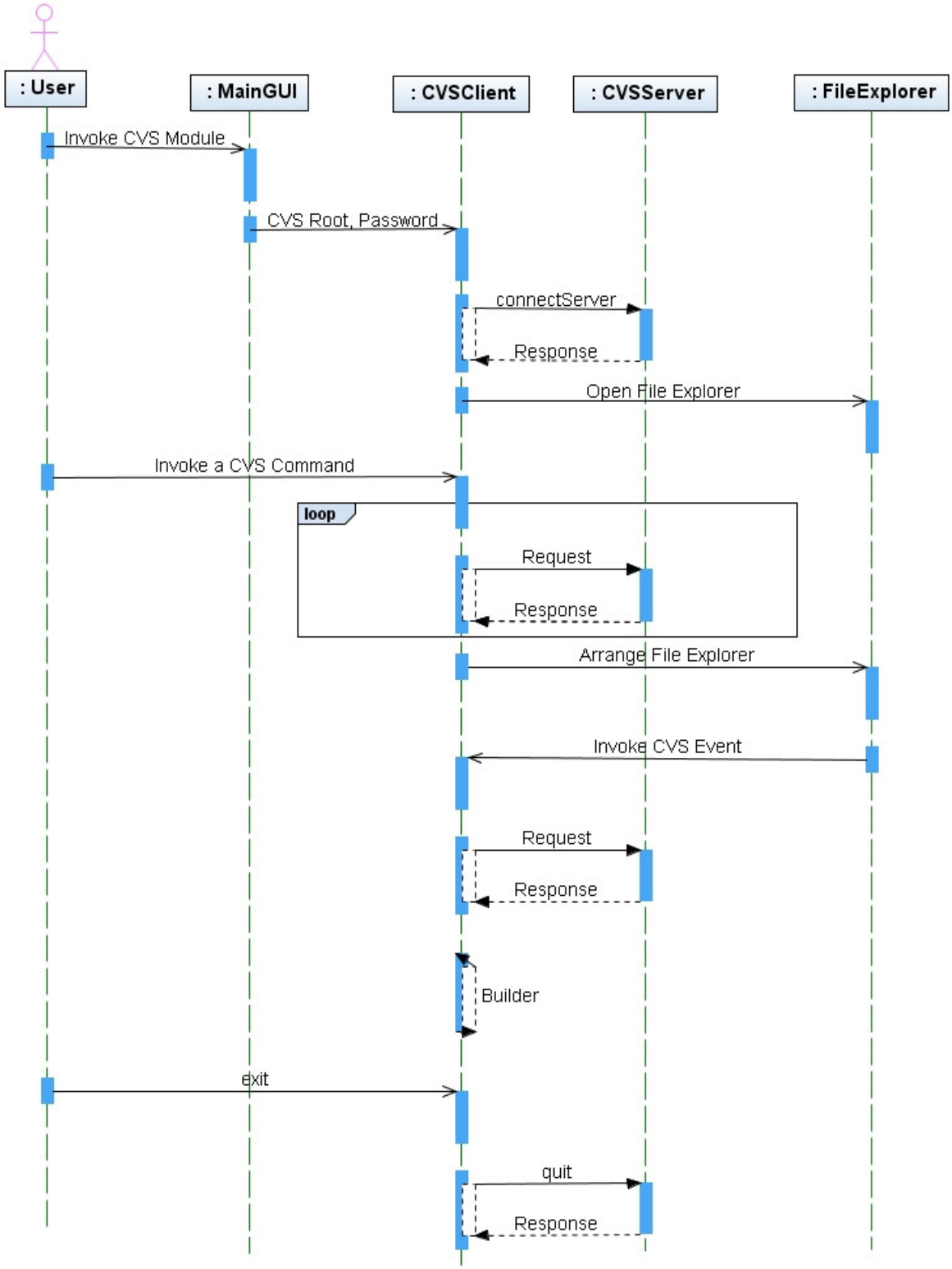
Method Name	Type	Arguments	Description of Method
TRANSFER(FTP)	Constructor	ftp	Constructs a TRANSFER object with given FTP object and initializing the boolean isopened
Request(Char,String)	Public:DataOutputStream	op, oparg	Requests between Client and Server invoked by commands
Response(Char,String)	Public:DataInputStream	op, oparg	Responses between Server and Client invoked by commands
FileAdded(String,String)	Public:void	source, path	Server is informed by addition of the file
FileInfo(String,String)	Public:void	source, path	Server is informed by file info
FileRemoved(String,String)	Public:void	source, path	Server is informed when

			a file is removed
FileUpdated(String,String)	Public:void	source, path	Server is informed by update of the file
TakeMessage()	Public:String	string	Invoked when a message from a server comes.

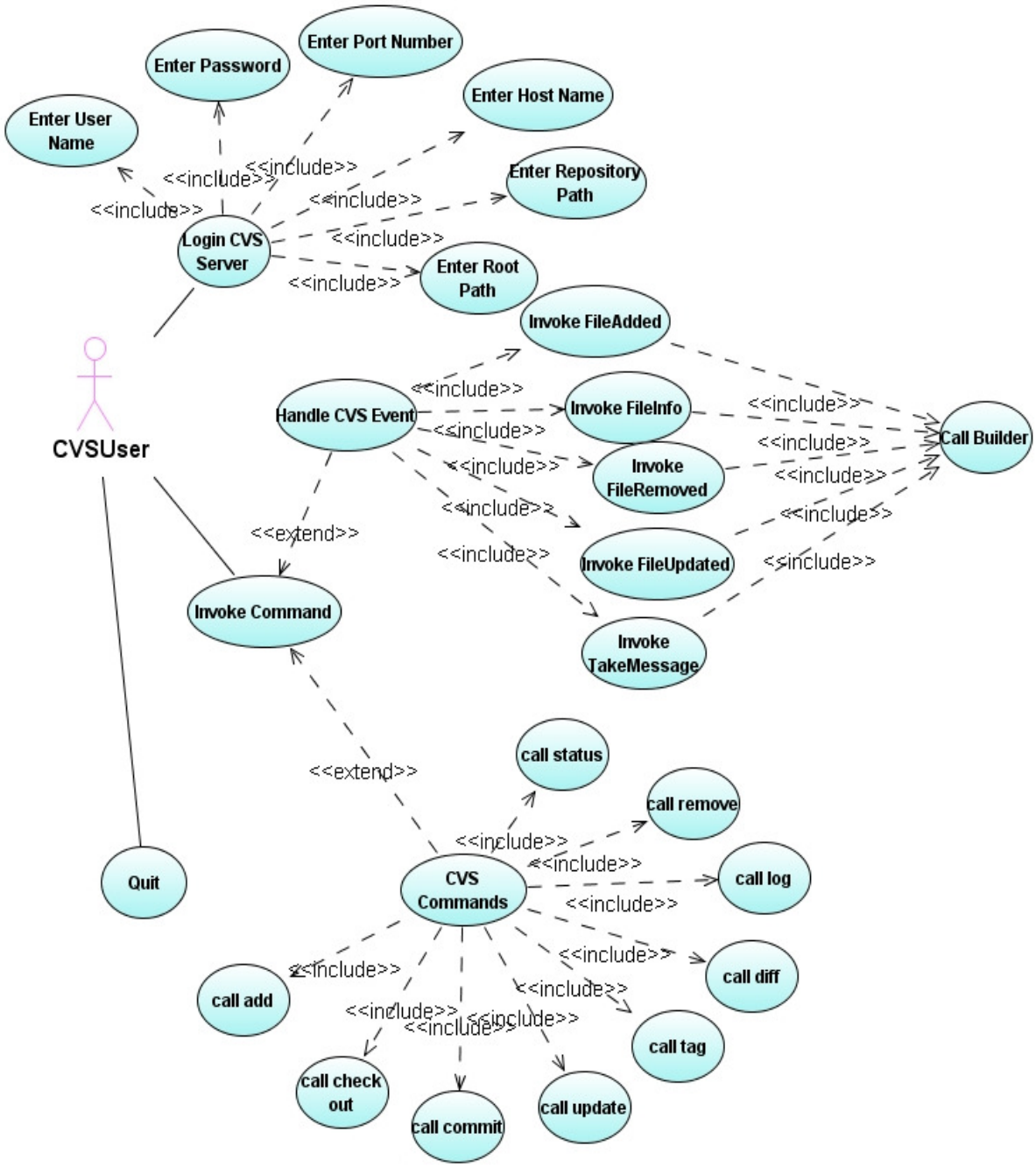
SERVER Class:

Method Name	Type	Arguments	Description of Method
Server()	Constructor	void	Constructs a Server object with initializing username ,password, hostname, port, path and time out

3.2.8.2 CVS SEQUENCE DIAGRAM


: User
: MainGUI
: CVSClient
: CVSServer
: FileExplorer
Invoke CVS Module
CVS Root, Password
connectServer
Response
Open File Explorer
Invoke a CVS Command
loop
Request
Response
Arrange File Explorer
Invoke CVS Event
Request
Response
Builder
exit
quit
Response

3.2.8.3 CVS USE CASE DIAGRAM



Flow of Events CVS Use Case

Objective: To allow the user to use CVS functionalities.

Precondition: A connection string has to be ready to connect a CVS server.

Main Flow

1. The user interacts with the CVS interface.
2. .The user selects one of the Login CVS Server, Invoke Command and Quit operations.
3. The user can enter username, enter password, enter port number, enter Repository Path and enter Root Path to connect a CVS Server.
4. The user can invoke CVS Events and CVS Commands.
5. User can call FileAdded, FileInfo, FileRemoved, FileUpdated, TakeMessage as a result of handling CVS Events which are called by Builder.
6. User can call status, remove, log, diff, tag, update, commit, checkout, add commands as result of CVS Commands.

Post-condition The user has connected a CVS server and makes CVS operations.

3.2.8.4 CVS USAGE SCENARIOS

Below are two usage scenarios for **kajax**'s :

Scenario 1:

In that scenario user will connect to a CVS server. From the MainGUI of the **kajax,** user clicks to the CVS menu. When user clicks to it a CVS Connection Wizard pops up. Our CVS server only provides only "server" Access Method. User enters UserName, HostName, PortNumber, Password and Repository Path. Or User enters CVS Root and password. If the information entered by the user is a correct one CVS connection is established. Otherwise, user enters a new connection setting or exits the CVS connection wizard.

Scenario 2:

In this scenario user will add a new file to add a new file to the CVS repository and updates it. As in the usage scenario 1 user connect to CVS server. The user uses the **kajax**'s CVS clients "add newfile.c" command which is under an Add Button. The CVS Client sends the Request methods of the add command which is abstracted from the user. Then CVS server returns Response methods related with the data. If the operation is

successful, user has added a new file to the CVS repository. For updating this file user saves it. When user updates the files, CVS Clients calls FileUpdated method to update the file.

3.3 FILE FORMATS

We are using xml files to store information about projects. So, we will not deal with any database server to keep persistent data. A project is a directory which has at least a project file. In the project directory there will be also an active projects file. The sources will not be categorized.

For each project in kajax, we are storing the name, path, author, date and active files within a project file. Its name will be same for any project and it will be "project.xml".

3.3.1 PROJECT FILE FORMAT

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
   <name>"Project Name"</name>
   <author>"Project Author"</author>
   <path>"Project Path"</path>
   <date>"Project Date"</date>
   <active_files>
      <file>"File Name1"</file>
      <file>"File Name2"</file>
      <!-- other active files -->
   </active_files>
   <current_file>
      <file>"File Name"</file>
   </current_file>
</project>
```

We are also storing the active projects as “active_projects.xml”. It consists of the paths of active projects. When a new session is opened, by parsing this file, the active projects will be shown on the solution explorer panel.

3.3.2 ACTIVE PROJECTS FILE FORMAT

```
<?xml version="1.0" encoding="UTF-8"?>
<active_projects>
   <project>
      <path>"Path of project1"</path>
      <path>"Path of project2"</path>
      <!-- other active projects -->
   </project>
   <main_project>
      <path>"Path of project"</path>
   </main_project>
</active_projects>
```

User can set his/her preferences in kajax and these preferences should be saved. So we are saving them in an xml file named “preferences.xml”.

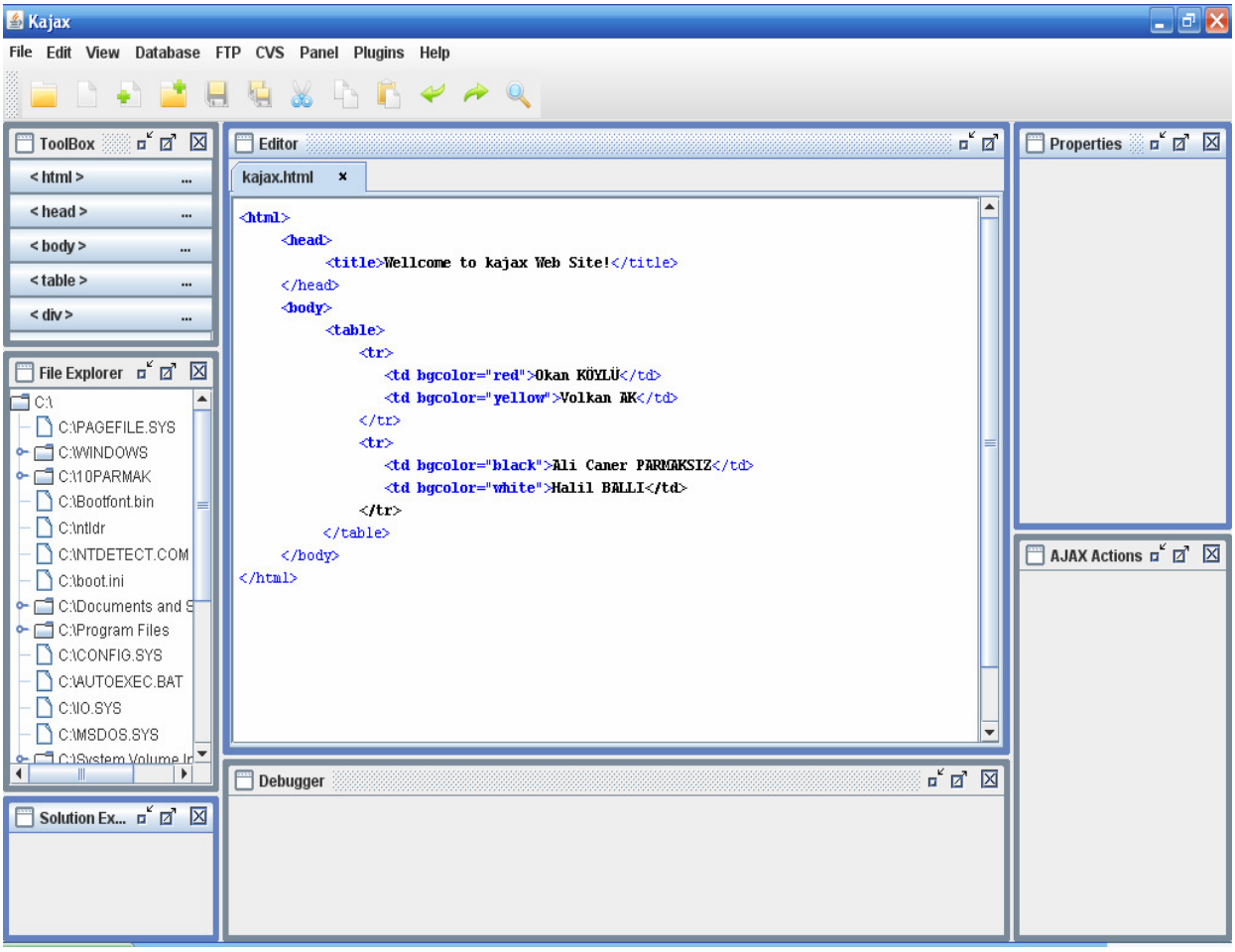
3.3.3 PREFERENCES FILE FORMAT

```
<?xml version="1.0" encoding="UTF-8"?>
<preferences>
   <author>
      <name>"Author Name"</name>
   </author>
   <workspace>
      <path>"Workspace Path"</path>
```

```
    </workspace>
    <font>
        <face>"Font-face"</face>
        <size>"Font-size"</size>
    </font>
    <encoding>"UTF8"</encoding>
</preferences>
```

4. GUI (GRAPHICAL USER INTERFACE)

The Prototype of kajax's MAIN GUI will be look as below:

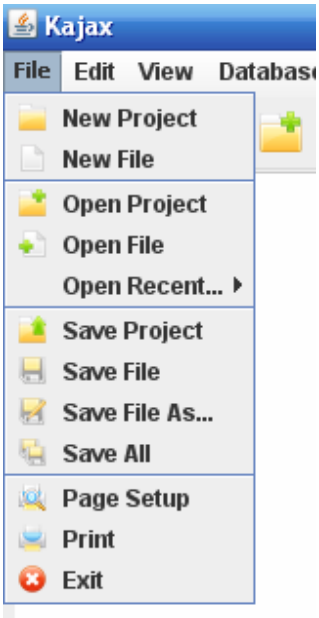


As seen from this screenshot, Panel, Editor and Debugger Modules are on the first coming screen. User can minimize or close these panels. User can activate Database, FTP, and CVS Modules from the Menu Bar.

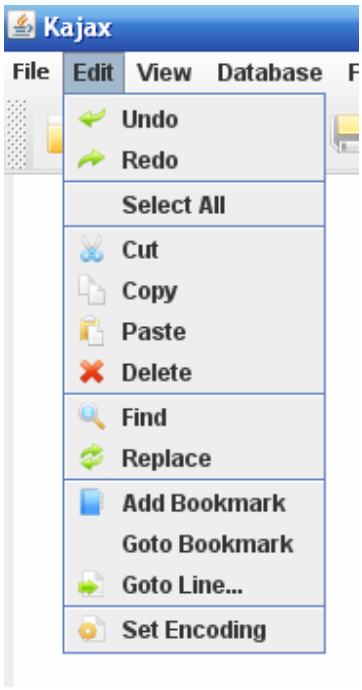
4.1 MENU BAR

4.1.1 FILE MENU

By this menu user can open new **AJAX** project by clicking to "New Project". "New File" opens a new HTML, CSS, JavaScript and Empty Document. User can open an existing project by "Open Project". User can open an existing file by "Open File". User can open recent project by using "Open Recent Projects". User can save project by "Save Project". User can save file with "Save File" or alternatively save all by "Save All". User can set up page view by "Page Setup". User can print the page by "Print". User exits the program with "Exit".

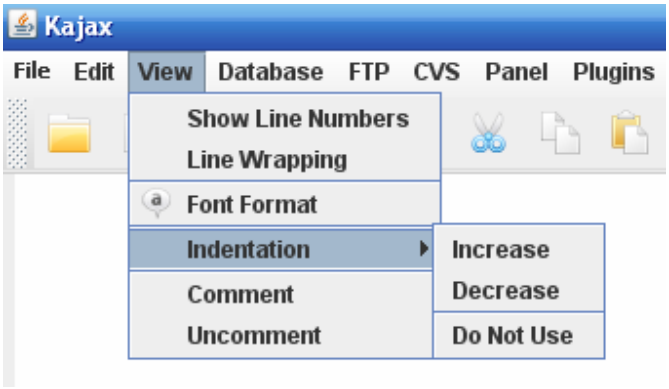


4.1.2 EDIT MENU



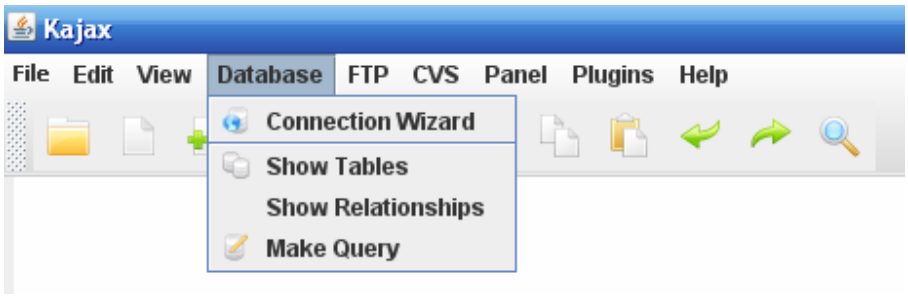
Edit menu is used for Editor Module. User can "Undo" his actions. User "Redo" his/her last undone actions. User cans "Select All" codes of the file. User can "Cut", "Copy", "Paste" and "Delete" the code segments. User "Find" and "Replace" a word. User can "Add Bookmark" or "Goto Bookmark" or "Goto Line" and "Set Encoding".

4.1.3 VIEW MENU



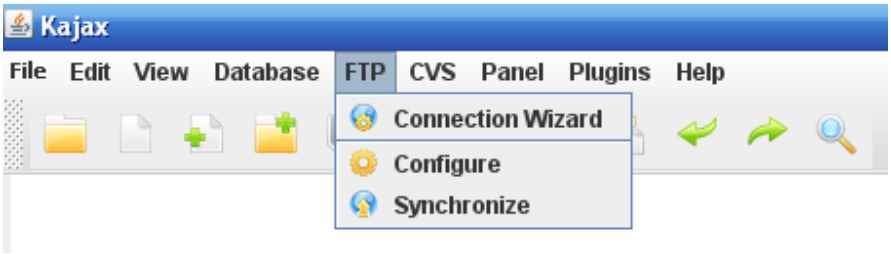
Users can "Show Line Numbers". User can use "Line Wrapping". User can arrange "Font Format". User can "Increase" or "Decrease" or "Do not use" indentation. User can "Comment" or "Uncomment".

4.1.4 DATABASE MENU



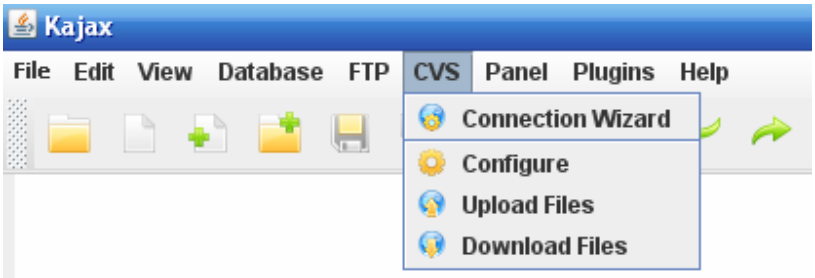
User opens the "Connection Wizard". User "Show Tables", "Show Relationships" and "Make Query".

4.1.5 FTP MENU



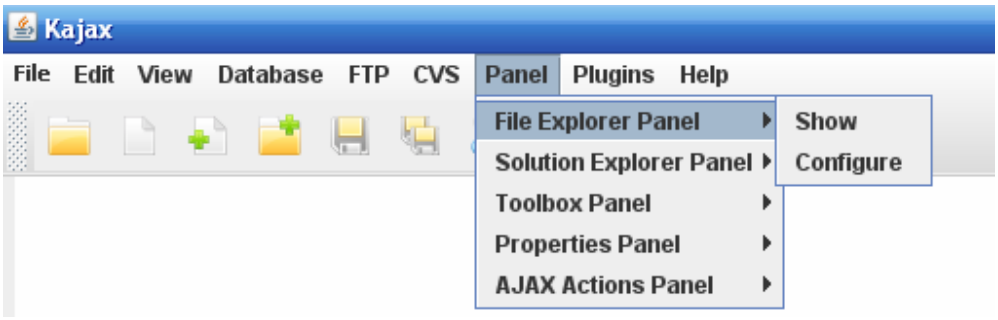
User opens "Connection Wizard". Use "Configure" FTP connection or "Synchronize" between two FTP Connection.

4.1.6 CVS MENU



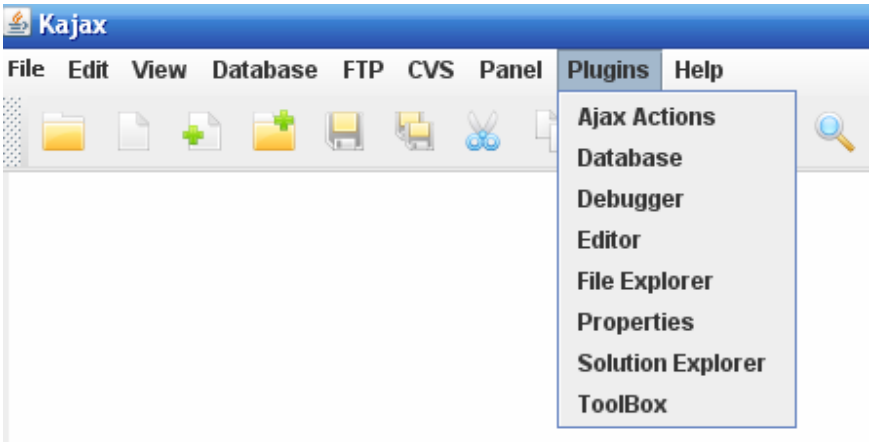
User can open "Connection Wizard" or "Configure" a CVS Server connection. If there exits a connection user can "Upload Files" or "Download Files"

4.1.7 PANEL MENU



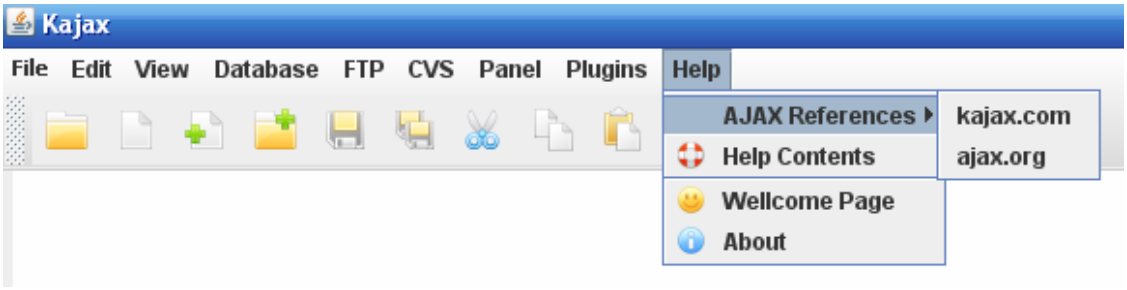
From the Panel Menu user can "Show" and "Configure" the "File Explorer Panel", "Solution Explorer Panel", "Toolbox Panel", "Properties Panel" and "AJAX Action Panel'.

4.1.8 PLUGINS MENU



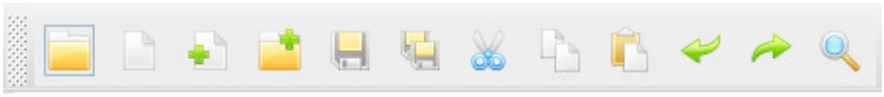
User can select plugins and load them into the development environment. All modules of kajax will be coded as plugin for this reason.

4.1.9 HELP MENU



User can use “AJAX References” which are mainly “kajax.com” and “ajax.org”. User can view “Help Contents” or “Welcome Page” or “About”.

4.2 TOOLBAR



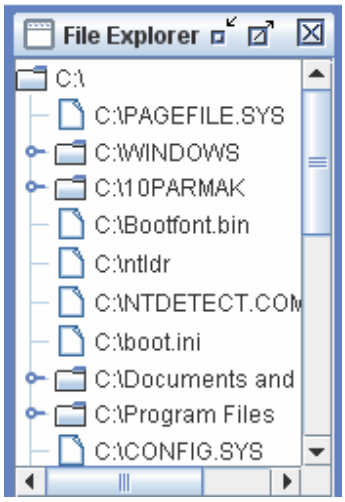
Current Tool Bar functionalities are related with File Menu Bar. User can open “New Project” or “New File”. User can “Save File”. User setup with “Page Setup” and “Print” the selected files.

Other Tool Bars will be implemented in the Final Design Report. Use will be able to select the tools he/she wants to use.

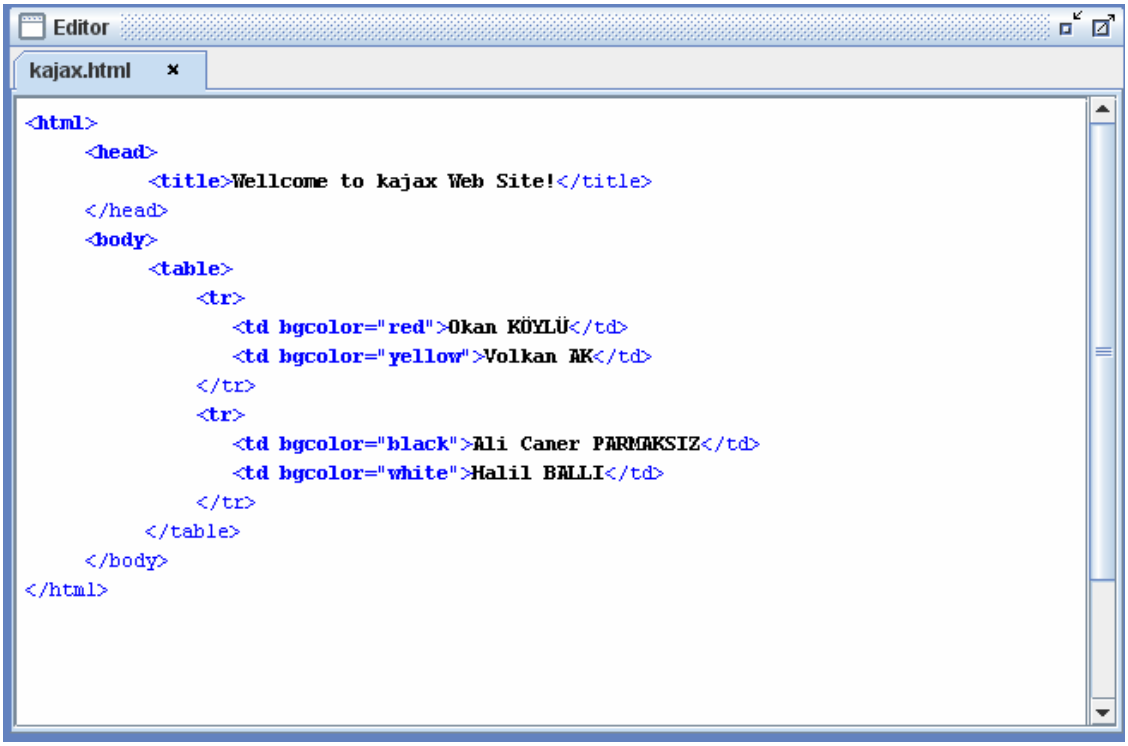
4.3 PANELS AND MODULES

Up to now we have designed the toolbox, file explorer and editor module.The screenshots of these are given below:

Toolbox



File Explorer



Editor

5. IMPLEMENTATION PLAN

ID	Task Name	Duration
1	Implementation of File Explorer	1 wk
2	Implementation of Solution Explorer	1 wk
3	Impelementation of Database Module	4 wks
4	Implementation of CVS Module	2 wks
5	Implementation of FTP Module	2 wks
6	Implementation of Toolbox	3 wks
7	Impelentation of Ajax Actions Module	3 wks
8	Implementation of Editor Module	8 wks
9	Reviewing General Architecture	1 wk
10	Integration of all modules	4 wks
11	Testing	3 wks
12	Documentation	4 wks
13	-----Developer Manual	2 wks
14	-----User Manual	2 wks
15	Deployment	1 wk