MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

SENIOR PROJECT
FALL 2006

DETAILED DESIGN REPORT

18.01.2007

SiHiRBAZ

KODADI®
YAZILIM

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose of the Document

This is the detailed design report for our project *"***SiHiRBAZ***"*. The purpose of this document is to express the final design decisions resulted from the detailed functional requirements and show the way of the development of our project. Firstly, our scope is presented in a detailed way and an overview of the project is added. Secondly, we have explained our design constraints which are people, time, hardware and software requirements for developer side. Then, modules of the system are declared separately with enhanced requirements. After that, we have shown use case diagrams which are partially updated from the analysis report. Besides, sequence, activity and class diagrams are provided for better decide on every component of our modules. Next, we have demonstrated our GUI with all of its functionality in the GUI Design part. Syntax and project management specification part is also added in order to provide a consistency and integrity between modules while writing code. Finally we have added a schedule part and a detailed GANTT chart to the report to show the progress of our project.

1.2 Scope

SiHiRBAZ consists of mainly 6 components which are HTML Text editor, WYSIWYG (What You See Is What You Get) Editor, parser and debugger, GUI Design and Database process handler. Moreover we will provide a CVS and FTP support. Embedded browser will also be supported to test developed application.

HTML Text Editor

An HTML editor is a software application for creating web pages. Although the HTML markup of a web page can be written with any text editor, specialized HTML editors can offer convenience and added functionality. For example, many HTML editors work not only with HTML, but also with related technologies such as CSS, XML and JavaScript. In some cases they also manage version control systems such as CVS or Subversion. We are writing a text editor with extra functionality for manipulating and previewing of typical programming

languages used for web development. Standard features such as syntax highlighting and automatic completion will be supported. HTML, XML and Java Script are supported by this editor.

WYSIWYG (What You See Is What You Get) Editor

WYSIWYG HTML editors provide an editing interface which resembles how the page will be displayed in a web browser. Most WYSIWYG editors also have a mode to edit HTML directly as described above. Because using a WYSIWYG editor does not require any HTML knowledge, they are easier for an average computer user to get started with.
The WYSIWYG view is achieved by embedding a layout engine based upon that used in a web browser. The layout engine will have been considerably enhanced by the editor's developers to allow for typing, pasting, deleting and moving the content. The goal is that, at all times during editing, the rendered result should represent what will be seen later in a typical web browser. Our WYSIWYG Editor will support standard HTML features such as buttons, forms etc. that users will be able to drag and drop. In addition to this, some simple AJAX components will be presented in labor of the user. These components are also available with drag and drop option.

Parser and Debugger

The parser that we plan to write will support XML, HTML and DOM files. Debugger supports only JavaScript because user will create AJAX components with JavaScript. Since it is impossible to develop a debugger for this project due to time constraints, we are planning to find, adapt and use an open source debugger component.

GUI Design

We designed a Graphical User Interface which is similar to the existing Development Environments. “Tibco”, “Aptana” and “Eclipse” are being used as a layout of our design. We are trying to develop a GUI design which shows our functionalities a user friendly and costless way.

Database Process Handler

This component is planed to manage a database connection. User will use this functionality to reach his/her database with a user friendly environment. Standard database functions like connection, table operations and SQL query evaluation are provided with this component.

Embedded Browser Support

Embedded browser will be provided to user to test and see existing file. With the help of design view, user is able to see the HTML view however, since AJAX components are not static, this feature will provide the realistic preview of the application.

CVS Support – FTP Publishing

We will provide a CVS connector to the user and FTP connection support for publishing.

1.3 Project Overview

At the beginning, AJAX was a new technology for nearly all of us. Therefore, we have spent a considerable time for research about this new technique. We tried to divide the project into modules to perform a better research activity. As a result this challenging activity we have specified the requirements which were expressed in analysis report. These were not so detailed but enough to explain what we our product will be like. After releasing analysis report, we have concentrated on deciding design issues. With the help of our requirements and technical research we have done, following principals are decided:

- The most important part of the project is WYSIWYG editor. It should provide the usability of creating and showing an AJAX application with user friendly way. We will implement this module by hand with existing JAVA packages.
- Text editor is the second important part of the project. It should provide all the standard features of a text editor. We are implementing this module by hand with existing JAVA packages.

- User will be provided a JavaScript debugger, which is already mentioned. Because of the time constraints and our preferential features (text and drag-and-drop editor), we have found an open source debugger which can be adapted to our project.
- Since the database applications play a big role in AJAX actions, we have decided to give importance to database connection tool.

1.4 Design Goals

1.4.1 Extensibility:

We will design our product considering that an improvement or plug in will be supported later. So, we can provide an update mechanism to ensure that our product is always up-to-date. Since AJAX is a developing technique, this feature will be really important.

1.4.2 Robustness:

The product should be able to manage invalid user inputs or inconsistent conditions. It provides error checking to ensure the right input format and returns errors and warnings to the user.

1.4.3 Reliability:

The product should produce the expected output for a valid input at all times.

1.4.4 Functionality:

The system should function according to the requirements specified in Requirements Analysis Report.

1.4.5 Usability:

The GUI should be user friendly. The goal is to provide the user an easy- to- use interface. The design of the GUI is based on that of Java based applications. This

design is chosen due to the familiarity of most users with this kind of interface. It consists of a menu bar, which is further decomposed into sub menus. Text boxes, scrollbars and pop-up menus are used to enhance user/system interaction. The user is placed in a familiar environment, which eases the general use of the application.

2. CONSTRAINTS

2.1 Experience & Skills of Members Constraints

As developers, our programming and design skills and experiences is also one of the restrictions. Although we have made software projects before, it was simpler than our current project and we do not have experience about creating development environments. Thus, this restricts our opinions of what we are able to make. In addition, it is very difficult for us to manage unexpected problems about this field but we may consult experienced people to get help about solving problems.

2.2 Time Constraints

We have to finish our project by June and also we should provide a prototype at the end of this semester. Therefore, especially for a software project, this is the most important constraints. Being able to use our time efficiently is very important for us to follow our program regularly. In case of schedule problem, to compensate lost time we should focus on the project instead of other responsibilities and spend more time on it. As a result, although we thought lots of features and special properties for development environment, for timing reasons, we may not able to do some exciting features because we should provide expected functionalities and basics firstly.

2.3 Funding Constraints

Since we will not need any additional hardware and software that have a cost for us to implement our project, we do not have a cost for them. In addition our team members are students and we will not pay anyone to during the project. Therefore, there is not any funding constraint.

2.4 Resource Constraints

While we are doing our project we need different hardware and software resources. We generally get easily these resources; as software requirements, we need web server, databases servers and some of development tools. Many of these are freeware, and we can get others in our department freely. We can also deal with hardware requirements for our project by the help of our personal resources temporarily so we do not think that the resources will be a problem for us to complete the project.

2.5 Performance

We are building our application for easy to understand and efficient to use. In addition there will be excessive user interaction, so performance is a very important constraint for our team. We consider the performance issue in during each steps of our project process.

3. SCHEDULE

GANTT chart can be found in Appendix.

4. SYSTEM MODULES

4.1 Text Editor

We are writing an HTML text editor with singleton design pattern for our development kit. HTML editors are basic text editors with extra functionality for the manipulation and previewing of code, typically of programming languages used for web development. According to the research we have done, we have specified following functional requirements for the text editor of our IDE:

- It will have the ability of reading and writing large files.
    - Open/read/save/load/close/new file operations will be supported by GUI module.

    - Large file reading is available.
- It will provide syntax highlighting for XML, HTML, JavaScript and CSS files.
    - Our system will read the syntax highlighting content when a new word is written.
    - System will skip the commented areas.
    - When the user has written a separate word in an uncommented area, it will be checked from the syntax highlighting content.
    - If it is matched, the related color will be applied.
    - This procedure will be supported for HTML, XML, JavaScript and CSS files.

- Unlimited undo/redo will be provided.
    - Undo
        - Save the modifications the user has done, in a stack.
        - Delete the last modification that has been done and if it is undoable in the editor.
        - Put the deleted item into a stack.
    - Redo
        - Read the last member on the stack.
        - Apply that item in the editor if it is redouble.
        - Remove it from the stack.
- "Markers" for remembering positions in files to return to later will be supported.
    - Store the position of the cursor for every file.
    - Restore the position of the cursor in a file when the file is selected.
    - Kill the marker when the file is closed.
- Any number of editor windows may be opened.
    - Open multiple files with a tab control in GUI.
    - Allow user to change the file he/she is modifying with a keyboard shortcut or tab select.
    - Assign a marker to the old file to remember the position.
    - Chose the marker of the new file and start from there.
- We will provide an auto-completion that does the followings:
    - If you are typing the name of an object (e.g. "document"), when you type the period (".") to call either a method or access a property for that object, it pops

up a small window displaying the available methods and properties for that object. You can also type ‘ctrl + space’ to access this help at any time.

    - This type of automatic completion will be provided for only user defined classes.
    - Specify the class of the object which is at the left of the point.
    - Show all the attributes and classes of that class.
    - Allow user to select an attribute or method from the list described above, put the selected item to the right of the list.
    - Place the cursor.
  - If you are calling a method on that object, when you type the first open parenthesis ("({<["), our editor will automatically create the closing parenthesis ("]>})") for you, and it will pop up a small window with the parameters that the method takes.
    - When the user writes one of the ("({<["") put the suitable ("]>})") and place the cursor between them.

- It will provide intelligent bracket matching, skips quoted literals and comments.
  - () ---- If the user has pressed to ‘%’ when he/she is on a ‘(’ or ‘)’, the cursor will automatically go to the matched parenthesis.
  - {} ---- If the user has pressed to ‘%’ when he/she is on a ‘{’ or ‘}’, the cursor will automatically go to the matched parenthesis.
  - [] ----- If the user has pressed to ‘%’ when he/she is on a ‘[’ or ‘]’, the cursor will automatically go to the matched parenthesis.
  - <> ----- If the user has pressed to ‘%’ when he/she is on a ‘<‘ or ‘>’, the cursor will automatically go to the matched parenthesis.
  - For all of parenthesis above, if there isn’t a matched parenthesis user will be provided an error message and cursor will not move.
  - A stack control mechanism will be used.
- It will provide automatic indentation.
  - If the user has written a ‘<’ and hasn’t closed it, put a ’tab’ space when the user entered a new line.
  - If the user has written a ‘{‘and pressed ‘enter’, move the cursor to the next line and one ‘tab’ space right.

- If the user has written an ‘if’ or ‘else’ clause, didn’t put a ‘{’and pressed ‘enter’, move the cursor next line and one ‘tab’ space right.
- If the user has written a ‘for’ or ‘while’ clause, didn’t put a ‘{’and pressed enter, move the cursor to the next line and one ‘tab’ space right.
- If the user has written a ‘>’ and had opened a ‘<’ before, put the ‘>’ one tab left.
- If the user has written a ‘}’ and had opened a ‘{’ before, put the ‘}’ one tab left.

- It will provide commands for commenting and commenting out code.
  - Enable user to select multiple rows.
  - Understand what language the selected code belongs to.
    - Search JavaScript statements for testing whether it is pure HTML or not.
    - Search HTML statements for testing whether it is pure JavaScript or not.
  - Comment the unselected code by putting the related comment item to it and giving blue color to the code.
  - Comment out the selected commented code by removing the comment items on it and giving black color to the code.
    - Test whether there exists comment items at the beginning and end of the selected rows.
- Search and replace supported.
  - Show a dialog box for search and replace to the user.
  - Search a word, letter, expression when user has pressed on search.
  - If the user didn’t enter an item (i.e. if it is blank) give a warning to the user and don’t do a search.
  - If the wanted item is found, show it to the user in a highlighted way and move the cursor to the end of this found result.
  - Replace the found letter, word, expression with the specified item if the user presses replace button on the dialog box.
  - Search again if the user presses next or previous.
  - Backward and forward search is allowed.
  - Continuous search is allowed.

- There will be a relation with WYSIWYG editor to support code generation while user.
    - Editor will take the design file and create html codes from it.
    - When the user uses an AJAX item, there will be a separate system file that notices this action.
    - Editor will read this file and produce the related AJAX code.
    - When the user is filling the forms of AJAX actions, files will be created.
    - Editor will read those files and produce their codes.

- Automatic save is provided to prevent user from loosing data.
    - Count the modifications the user has made to an already saved file.
    - When this count is 3 save the current entry to the temporary file.
    - When the system crashes, ask to modify changes when the user reopens the file.

4.2 WYSIWYG Editor

- Unlimited undo/redo will be provided.
    - Undo
        - Save the modifications the user has made.
        - Delete the last modification that has been done in the editor.
        - Put the deleted item into a stack.
    - Redo
        - Read the last member on the stack.
        - Apply the read item.
        - Remove it from the stack.

- A Palette for displaying built-in Ajax actions and HTML elements which can be added by dragging and dropping.
    - For Palette, a window will be shown which consists of drag-able Ajax actions buttons and HTML elements buttons.
    - User can drag a button from palette.
    - Drop it in to the Design View area.
    - Call code generation.

- Open properties window if object is an HTML object.
- In the palette we will provide built-in Ajax Actions other than HTML objects such as:
    - AJAX Dynamic Table
    - AJAX Photo Gallery
    - Drag and drop
    - Accordion
    - Tabset
    - Collapsible region
    - Suggest text field
    - Dialog box
    - Rating widget
    - Edit in place

- User will be able to insert text in Design view.
    - Get the written text.
    - Call code generation.

- An added table object can be selected. If it is selected:
    - User can modify its size and size of its rows and columns.
    - After a modification generate code is called.

- Create and modify added objects through properties window.
    - All objects will be selectable.
    - If an object is selected, relevant properties window will be shown with its current properties.
    - User can use properties window for modifications.
    - After a modification call generate code

- Permits files or entire folders to be dragged directly into the editor
    - If the input is a folder, zip the input folder.
    - Generate code is called.

- Drag & drop of image files directly into the editor, as well as file browsing
    - Check the image size and type.
    - If there is any violation show user an error message.
    - If input is suitable after dropping it show the properties window with parameters related with the type of button.
    - Call generate code

- If the dragged object is an Ajax action open event window
    - According to the type of Ajax Action an Event Window will be opened.
    - User will enter the required input for actions.
    - Call generate code

- Code generation will be done after using properties or event windows, dragging & dropping an object from palette or dragging & dropping a file from outside.
    - Appropriate code will be read from file or generated.
        - Take id and type of button from GUI.
        - Generate code.
        - Send the cursor position to Text Editor to add codes the right position.
        - Send the codes to Text Editor.
        - Generate design view function is called to refresh the design view according to the changes in code view.

4.3 Database Editor

The user will be able to connect to a database server if s/he has access rights on it.

- After connecting to a database a GUI window will be provided to user for database operations.
    - Show input dialog box.
        - Ask user account name, password, location of database and type of database (MySQL or Oracle)
        - Get user account information.
        - Try to connect to the database and get result from DBMS.

        - If result is true show user the database.
            - Show user available and selectable schemas.
        - If result is false show an error message and request account information again.

- User will be able to execute queries on the database.
    - Show a query window with execute button to the user for entering queries.
    - Query window will be shown on top.
    - If execute button is pressed get the query.
    - Check the query if it is empty or not.
    - If query is empty show user a message to enter a query.
    - If query is not empty send it to DBMS and get the result.
    - If result is true show the result to user.
        - If it is a SELECT, UPDATE or CREATE TABLE query show the result otherwise show a message saying "query has been successfully executed".
        - If result is false show user the error message returned from DBMS.

- The user interface will provide user the ability to execute queries (table, column or row creation, modification, deletion) without the need to know the proper syntax by just clicking on the appropriate action.
    - An attribute of a tuple will be selectable.
    - When an attribute is selected its background color will change and user will be able to enter a new value for that attribute.
        - If the new value is empty NULL will be used.
        - After user enters a new value for an attribute, an UPDATE query will be generated. Update query is generated when update row is clicked.
        - Generated update query will be sent to DBMS and the DBMS will be listened for a result.
        - If result is true the new value of the tuple will be shown to user otherwise the error returned from DBMS will be shown.
    - User can select a row. If user selects a row, its background color will change.
    - After selecting a row user can delete it by a delete icon.
        - If the delete icon is pressed, a delete query will be generated.

        - Generated query will be sent to DBMS and the result will be listened.
        - If result is true updated table will be shown to user otherwise the error returned from DBMS will be shown.
    - There will be a create table button.
    - If the button is pressed a dialog box will be shown with 'Create' and 'Cancel' buttons.
        - User will be listened for name of table and columns of table and properties of columns (primary key, foreign key, auto increment, data type, NULL or NOT NULL and unique).
        - If 'Cancel' is pressed no change is done and the dialog box is closed.
        - If 'Create' is pressed name of table and names of columns will be checked for emptiness.
        - If at least one of them is empty user will be prompted to enter a name for it.
        - If none is empty a query will be generated.
        - Generated query will be sent to DBMS and DBMS will be listened for a result.
        - If result is false error returned from DBMS will be shown.
        - If result is true newly created table will be shown.

    - User can insert a new row with an icon.
        - After the icon is pressed, a dialog box will open asking user values for NOT NULL attributes with 'Insert' and 'Cancel' buttons.
        - After 'Cancel' is pressed no change will be done and dialog box will close.
        - After 'Insert' is pressed a query will be generated.
        - Generated query will be sent to DBMS and DBMS will be listened for a result.
        - If result is false error returned from DBMS will be shown to user.
        - If result is true updated table be shown to user.
    - User can change the columns of a table by selecting them.
        - When a column is selected, its background color will change.
        - User can drop a column by selecting the delete icon.
        - User can change the name of a column by entering it a new name.

        - User can insert a new column by clicking insert icon.
        - After an operation a query is generated.
        - Generated query will be sent to DBMS and DBMS will be listened for a result.
        - If result is false error returned from DBMS will be shown to user.
        - If result is true updated table will be shown to user.

- Schema selection will be provided.
    - After user connects to a database, schemas in that database will be shown to user.
    - User can select a schema. Schemas will be shown as rollouts (can change).

- All the tables of a selected schema will be shown.
    - After a schema is selected its tables will be shown as selectable items.

- User will be able to select a table to view or modify.
    - After a table is selected its rows and columns will be shown.
    - There will be icons for manipulating rows and columns. (Discussed above)

- Detailed information of the selected table (columns, rows) will be shown.
    - There will be an option to pass from these views to table view.
    - In the detailed table view the columns of the table will be shown and all rows of the table will be listed.
    - User will be able to select to view detailed information about columns of a table and modify it.
        - All columns' attributes (primary key, foreign key, auto increment, data type, NULL or NOT NULL and unique) will be shown.
        - NULL or NOT NULL, foreign key and unique attributes can be changeable others not.
        - After an attribute is modified, a query is generated.
        - Generated query is sent to DBMS and DBMS is listened for a result.
        - If result is not true, error returned from DBMS is shown.
        - If result is true, updated table columns are shown.

    - Data types of a table's columns will be shown when selected.
    - User will be able to manipulate rows.

- If the user tries to execute an illegal query or does not have the necessary privileges to execute a query, an error message will be shown.
    - If DBMS returns an error message, it will be shown to user and user will be asked to try again.
- When the user makes a change on database, the result will be shown immediately.
- The user will be prompted if s/he looses his/her connection.
    - If database connection is lost, a message will be shown to user saying "Connection lost".

- The connection information will be provided as an include file to the user.
    - If entered account information is correct, an include file will be generated in PHP format.
    - User can select to include this file to his/her source code.
        - If user selects to include the file, necessary code statement will be generated and sent to text editor.

4.4 Debugger & DOM Inspector

4.4.1 JavaScript Debugger

We will provide the following facilities for user in the JavaScript debugger in our product to control the execution of scripts that users are debugging:

- Instant-on JavaScript debugger will be provided.
- Debug any web page containing JavaScript source or included JavaScript files, or standalone JavaScript files.
    - Debug button is pressed.
        - If web page contains javascript sources between <script></script> tags

        - Code block(s) is/are highlighted.
    - If web page includes .js file
        - .js file is opened in new editor view tab.
    - If the file has already .js file
        - .js file is opened in new editor view tab.
    - User will able to stop debugging by pressing stop debugging button.

- Pause, Resume, step in/over/out, break operations will be provided for debugging.
    - User will able to control debugging operations by buttons provided on the toolbar.
    - Currently executed code is highlighted on the editor view.
- Some views will be shown to user:
    - Call Stack View
        - Currently executed code/function will be showed with its name and value.
    - Watch View
        - User enters variable name s/he wants to trace in to the variable name field.
        - Check whether the variable name is matched.
        - If it is matched.
            - Current value of it is displayed in value field.
        - If it is not matched.
            - Error message is shown to the user.

- User will be able to set and clear JavaScript breakpoints in:
    - JavaScript files
    - HTML with embedded JavaScript and linked JavaScript files

- User will be able to set a breakpoint by:
    - Simply single-clicking on the line number of the line at which s/he wants to set a breakpoint.

    - If the selected line contains executable code a red dot will appear next to the line number and a breakpoint will be set at that location.

- User will be able to clear breakpoint by:
    - Place the cursor on the line at which you want to clear a breakpoint
    - Simply single-click on the red dot or the line number of the line at which you want to clear a breakpoint.

4.4.2 DOM Inspection Tool

Its main purpose is to inspect the Document Object Model (DOM) tree of HTML and XML-based documents by using dom parser. The initial HTML for an Ajax Application is often minimal, and in any event likely to change over time due to DOM Manipulation. All of this is very useful for checking assumptions and diagnosing problems, since many Ajax bugs arise because the programmer misunderstood the DOM state at a particular time.

- Showing the DOM-Tree with nodes.
    - Get the file type of the current file in the editor view.
    - Check whether the file extension is .html or .xml
    - If the result is true
        - Parse the file.
        - Show the nodes on the tree view.
    - Else do not show anything on the Dom inspection.

- Drill down the hierarchy, search for keywords.
    - User will be collapse/expand tree view of a document.
    - User enters the keyword s/he wants to search in the document
    - Check whether the keyword is in document.
    - If it is found
        - The node is highlighted.
    - If it is not found
        - Error message is shown to the user.

- Current element highlighted in page.
    - If user will press the node on the tree view of the document.
    - Send a request to WYSIWYG.
    - The html component which the selected node contained will be highlighted.

- Node name, type and value are shown.
    - If user will press the node on the tree view of the document.
    - Name, type and value of this node on the tree view of the document will be showed in the name, type and value field of the DOM Inspector module.

4.5 FTP Manager

- User will enter required connection information like host, user, password and clicks "Connect" button.
    - If connection cannot be acquired an error is shown to user.
    - If connection can be acquired FTP Window is opened.

- User selects a file and clicks to "Get File".
    - User retrieves a copy of the file at the FTP Server into a local workspace.

- User selects a file and clicks "Send File"
    - After user clicks send file the file is sent to FTP server.

- User presses disconnect button.
    - A close connection signal is sent to FTP server.
    - User is prompted that s/he is disconnected.

4.6 CVS Manager

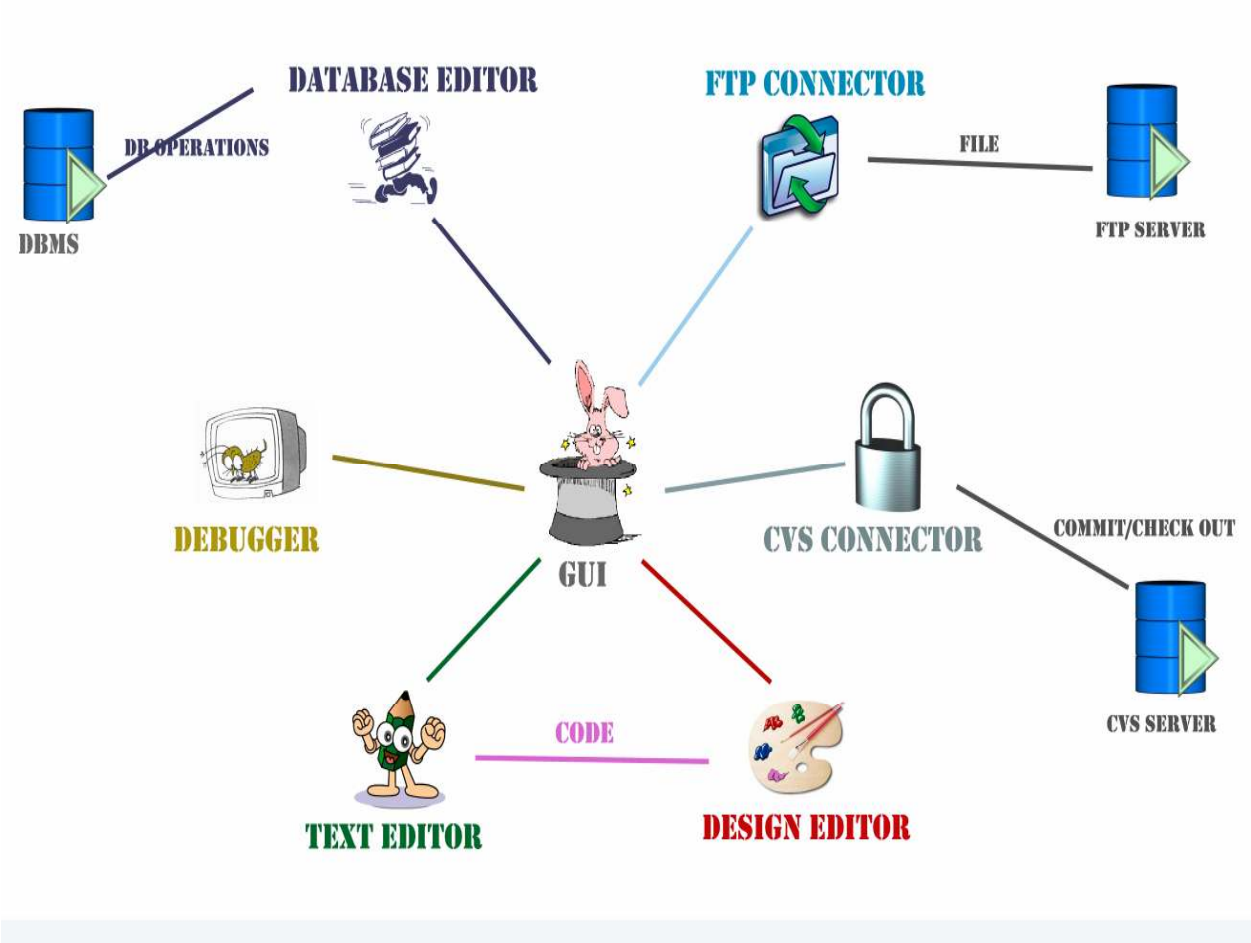
- User will enter required connection information like host, repository path, user, password, connection type, and clicks "Finish" button.
    - If connection cannot be acquired an error will be shown to user.
    - If connection is acquired a CVS repository window, which includes list of files, will be open for user to perform versioning actions like "CVS Check-out" and "CVS Commit".

- User selects a file and clicks "CVS Commit".
    - If request can be done user will be able to create a new revision of the file, containing his/her changes, into the repository.
    - If a file commit is not allowed by server, an error is shown to user.

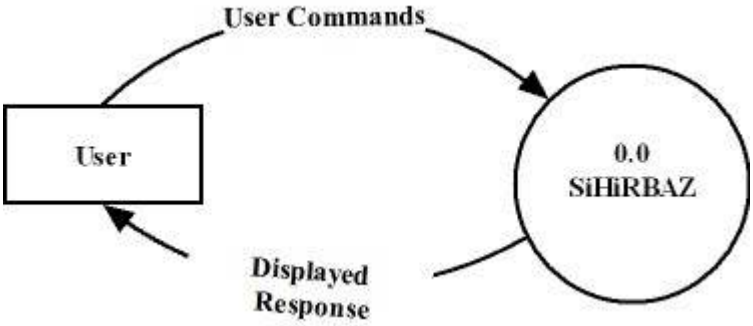
- User selects a file and clicks to "CVS Check-out", s/he will be able to retrieve a copy of the entire repository or a portion of the directory tree in the repository into a local workspace.
    - Selected file is requested from server.
    - If file is not available an error is shown else user acquires the file.

- User can close connection by pressing a button.
    - If user requests a connection close, a close signal is sent to server.

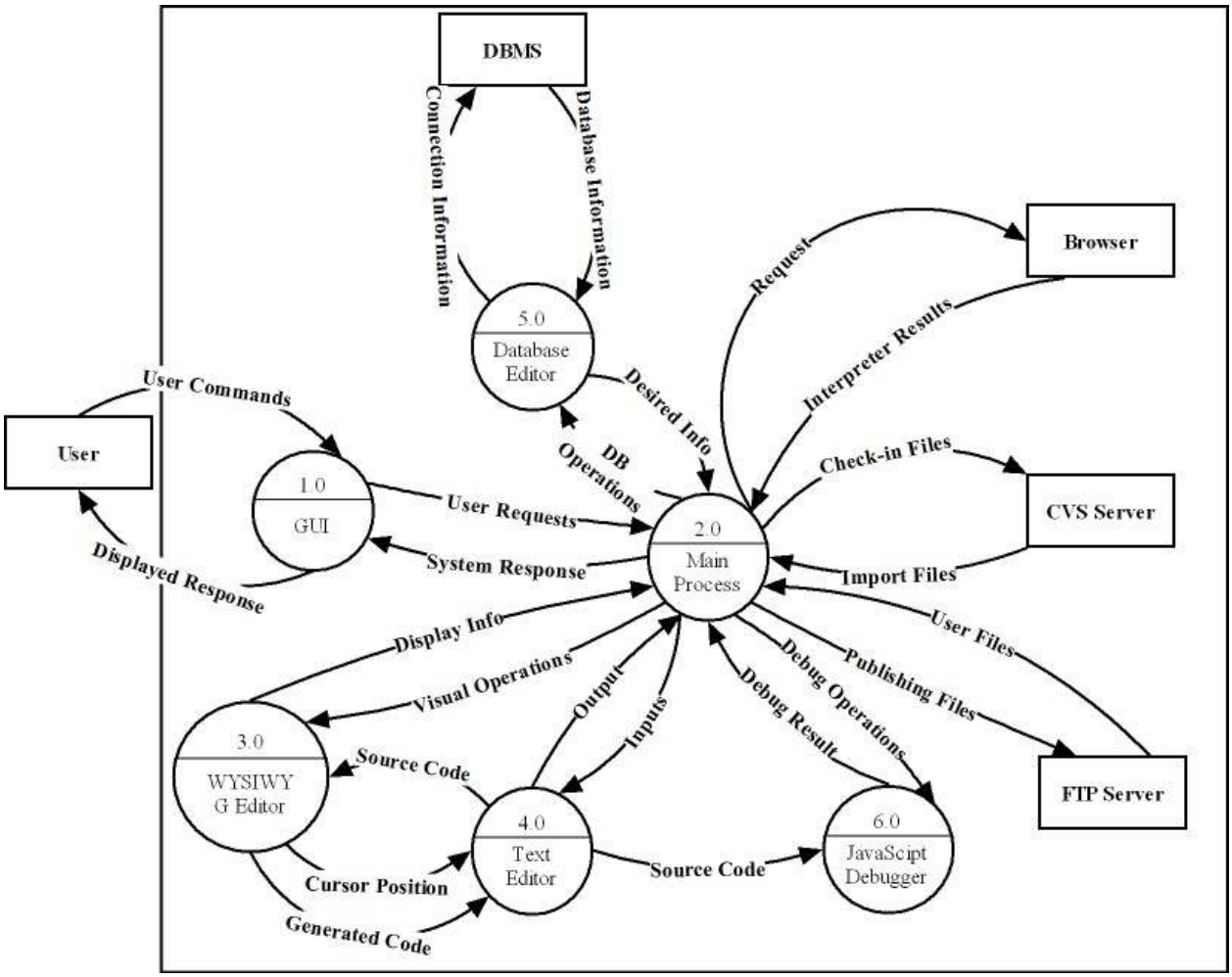
4.7 SYSTEM ARCHITECTURE



4.7.1 Level0 DFD



4.7.2 Level1 DFD



4.7.3 Data Dictionary

name:	User commands
where used / how used:	GUI(1.0) *input*
description:	Every external input that user enters

name:	Displayed Response
where used / how used:	GUI(1.0) *output*
description:	Every output provided by system

name:	Database Information
where used / how used:	Database Editor (5.0) *input*
description:	Information Stored in user's database

name:	Connection Information
where used / how used:	Database Editor (5.0) *output*
description:	Connection information and Queries entered by user

name:	Request
where used / how used:	Main Process (2.0) *output*
description:	Signal to publish application in browser

name:	Interpreter Results
where used / how used:	Main Process (2.0) *intput*
description:	Returned result from JavaScripts Errors

name:	Check-in Files
where used / how used:	Main Process (2.0) *output*
description:	Sending files to CVS server

name:	Import Files
where used / how used:	Main Process (2.0) *input*
description:	Receiving Files from CVS server

name:	User Files
where used / how used:	Main Process (2.0) *input*
description:	Sending files to FTP server

name:	Publishing Files
where used / how used:	Main Process (2.0) *output*
description:	Receiving files from FTP server

name:	Debug Operations
where used / how used:	Main Process (2.0) *output* JavaScript Debugger (6.0) *input*
description:	Debugger related inputs

name:	Debug Result
where used / how used:	Main Process (2.0) *input* JavaScript Debugger (6.0) *output*
description:	Outputs of debug operation

name:	Source Code
where used / how used:	WYSIWYG Editor (3.0) *input* Text Editor (4.0) *output* JavaScript Debugger (6.0) *input*
description:	Source Code of Application

name:	Cursor Position
where used / how used:	WYSIWYG Editor (3.0) output Text Editor (4.0) *input*
description:	Inputs from design view to determine the position of cursor in code view

name:	Generated Code
where used / how used:	WYSIWYG Editor (3.0) output Text Editor (4.0) *input*
description:	Inputs from design view to add generated codes to code view.

name:	User Request
where used / how used:	GUI(1.0) *output* Main Process (2.0) *input*
description:	User inputs

name:	System Response
where used / how used:	GUI(1.0) *input* Main Process (2.0) *output*
description:	System output

name:	Display Info
where used / how used:	WYSIWYG Editor (3.0) *output* Main Process(2.0) *input*
description:	Design View output for display

name:	Visual Operations
where used / how used:	WYSIWYG Editor (3.0) *output* Main Process(2.0) *output*
description:	User inputs related with WYSIWYG editor

name:	Output
where used / how used:	Main Process(2.0) *input* Text Editor (4.0) *output*
description:	Output from Text editor to display

name:	Input
where used / how used:	Main Process(2.0) *output* Text Editor (4.0) *input*
description:	User inputs related with Text editor

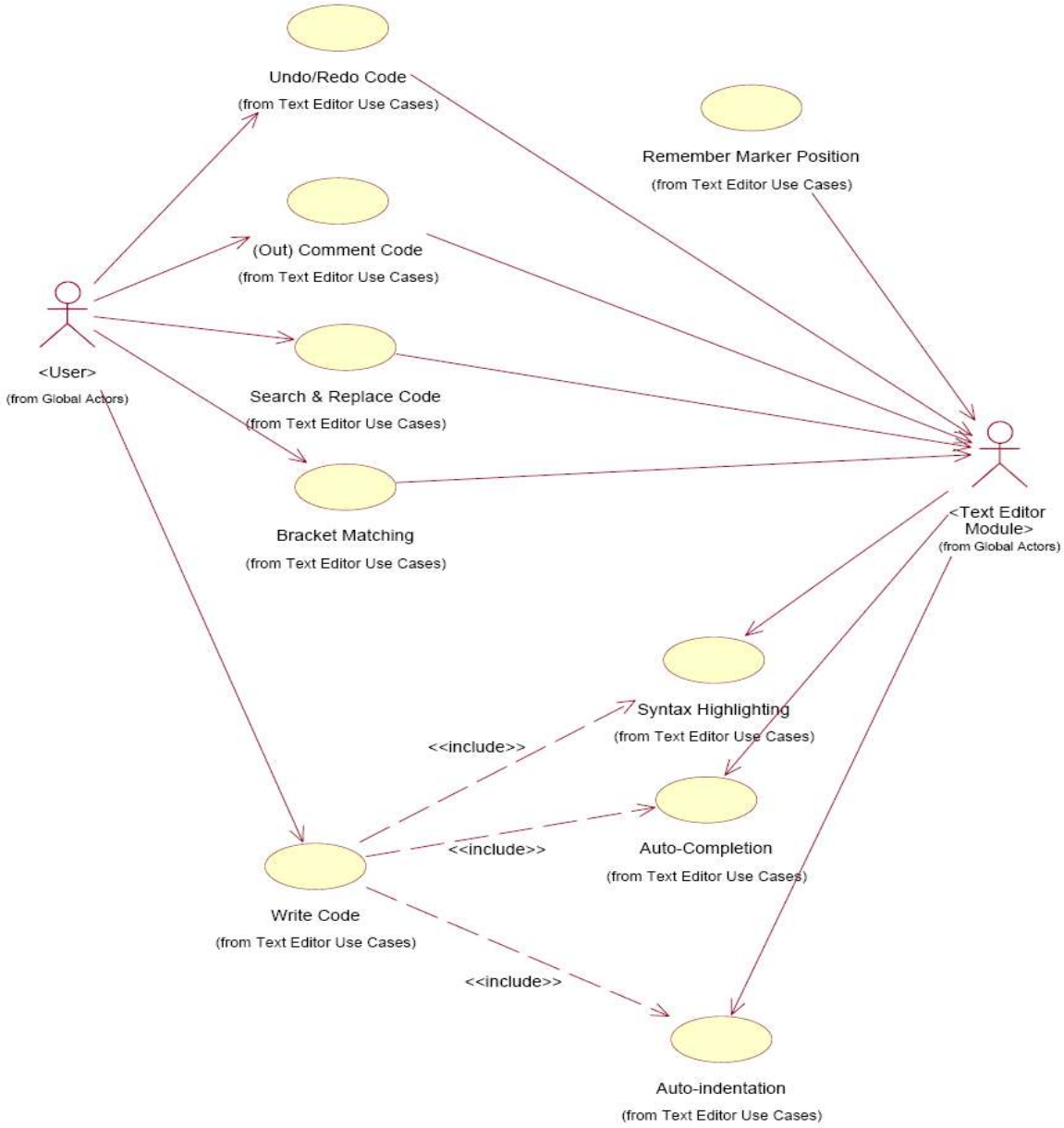
name:	Database Operations
where used / how used:	Database Editor (5.0) *input* Main Process (2.0) output
description:	User requests on database

name:	Desired Information
where used / how used:	Database Editor (5.0) *output* Main Process (2.0) *input*
description:	Information of user database for display

5. SYSTEM DESIGN

5.1 Use Cases and Use Case Scenarios

5.1.1 Text Editor



Undo/Redo Code: User will press undo or redo to disable or enable changes he/she made on his/her file.

Comment/ Comment out code: User will select a part from the file and comment in or out this part.

Search & Replace code: User will find an expression, word or sentence and replace it with another.

Use keyboard shortcuts: User will use keyboard shortcuts to manage the tasks easily.

Select rectangle: User will select a part in a rectangle and change it according to his/her needs.

Bracket Matching: When user comes to a bracket, cursor will automatically shoe the match of that bracket.

Customize toolbar: User will customize the toolbar according to his/her needs.

Use palette: User will use the palette to add the source codes of the built-in components.

Write Code: User will write source code.

Syntax Highlighting: When the user writes his/her code syntax highlighting will automatically highlight the built-in functions or expressions of the related language.

Automatic Completion: When user is typing the name of an object (e.g. "document"), when you type the period (".") to call either a method or access a property for that object, it pops up a small window displaying the available methods and properties for that object. User can also type ctrl + space to access this help at any time. When user is calling a method on that object, when you type the first open parenthesis ("("), our editor will automatically create the closing parenthesis (")") for him/her, and it will pop up a small window with the parameters that the method takes.

Automatic Indentation: When the user is writing a code, automatic indentation will indent his/her code according to the related programming language.

HTML code cleanup/formatting: After user writes the code, editor will check HTML validity and clean the code to make a correct HTML file.

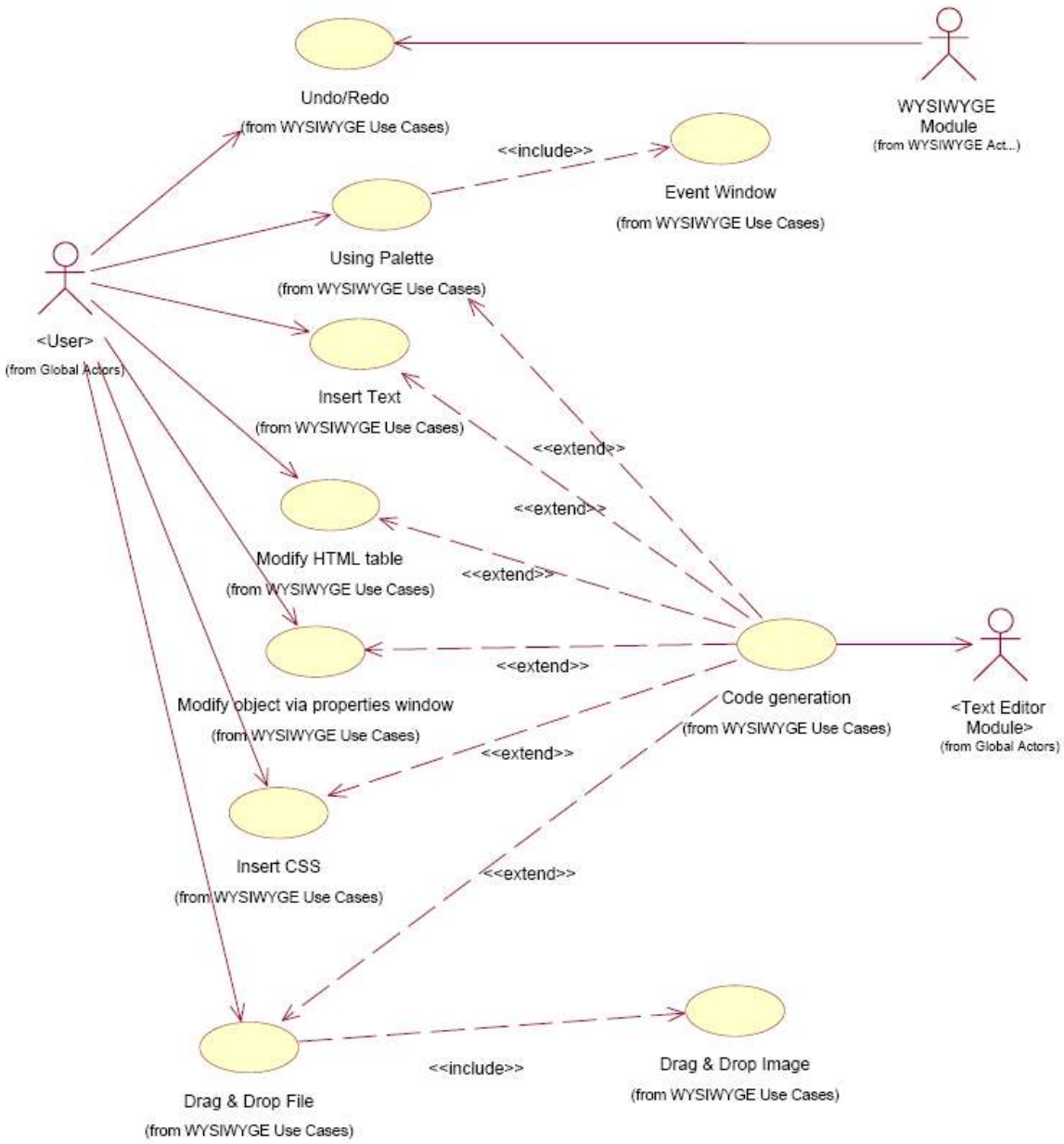
Link Checking: When the user has entered a link, editor will automatically highlight it as a link.

HTML Validation: While user is writing the code, editor will check if he/she is writing HTML code validly.

Code Generation: When the user uses the palette, editor will automatically generate the related code of the component.

Provide Marker: When the user opens another file, "markers" for remembering positions in files to return to later will be supported.

5.1.2 WYISWYG Editor



Undo/Redo operation: User will press undo or redo to disable or enable changes s/he made on his/her file.

Keyboard Shortcuts: User will use keyboard shortcuts to manage tasks easily.

Using Palette: User will use drag & drop option to add built-in component to his/her design view.

Insert Text: User will enter text input to his/her design view.

Modifying Object: User will modify components that are previously added.

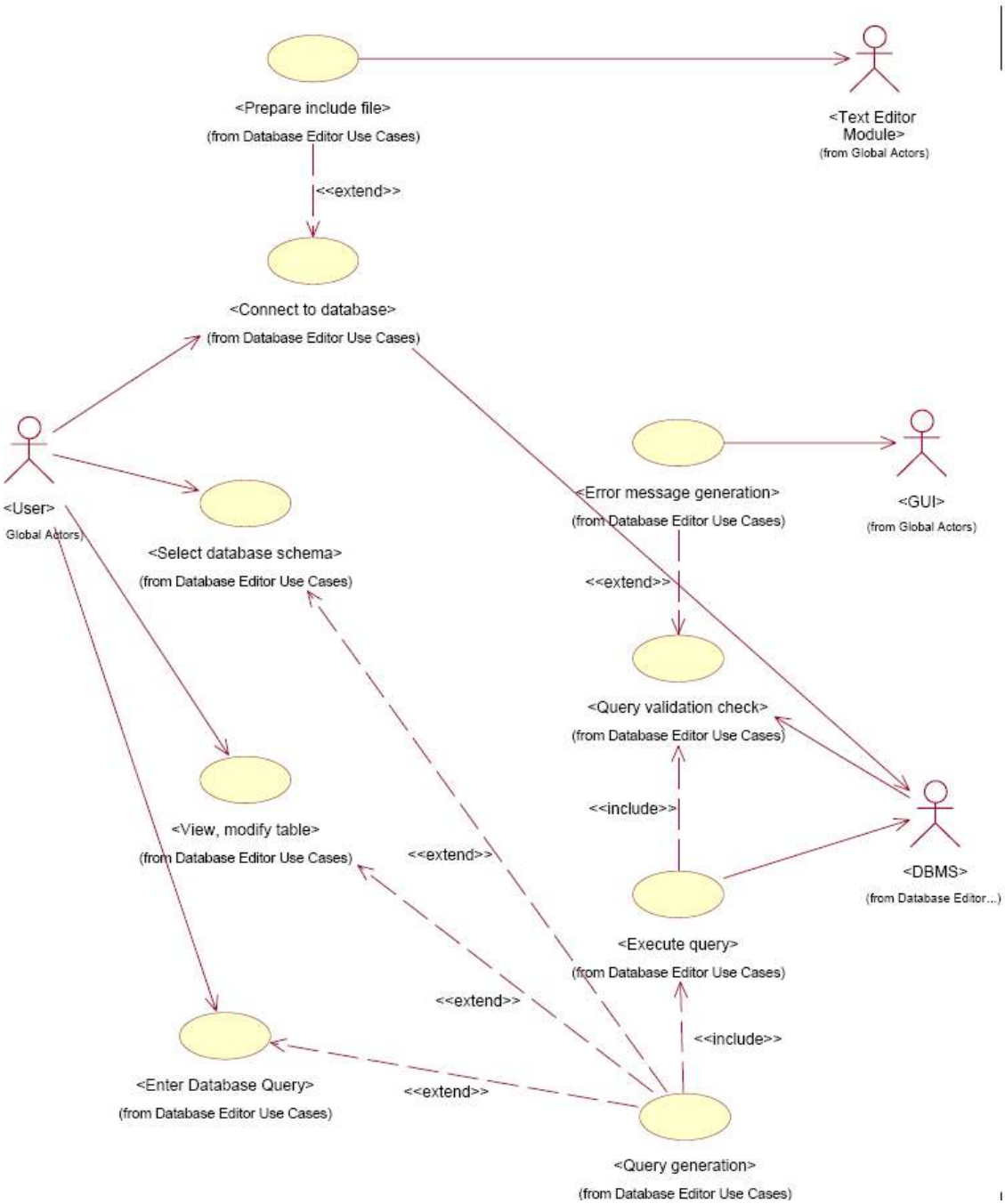
Customizing the properties of element on properties editor: User will arrange the desired properties of elements.

File Operations from desktop: User will add images and files to his/her design view with drag and drop directly from desktop.

Image Operations: User will add, delete, resize etc. images.

Code Generation: When the user use palette/insert text/modify objects/customize properties of elements/make file operations /make image operations.

5.1.3 Database Editor



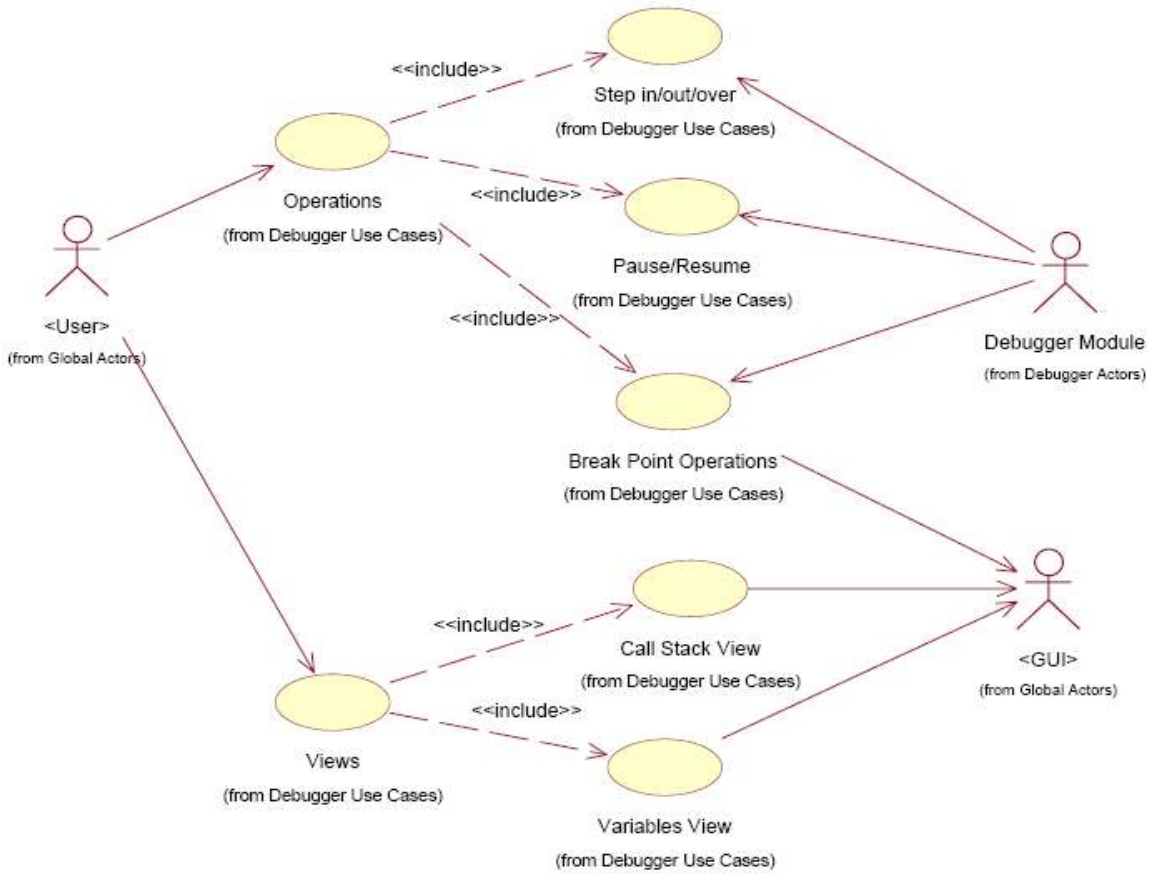
Connect to Database: User will press connect button. Then user interface will bring up connection dialog and waits for the user to enter connection info. After user enters connection info, user interface will send it to DBMS. If the connection info is correct, DBMS will return database info and user interface will show the result to user and also will prepare an include file.

Select Database Schema: User will select to view a schema. User interface will generate query and send it to DBMS. DBMS will execute the query and send the result to UI. UI will show the result to user. If the query is invalid, UI will show an error message to user.

View, Modify Table: User will select an operation on a table. User interface will generate query and send it to DBMS. DBMS will execute the query and send the result to UI. UI will show the result to user. If the query is invalid, UI will show an error message to user.

Enter SQL Query: User will write a query. UI will send it to DBMS. DBMS will execute the query and send the result to UI. UI will show the result to user. If the query is invalid, UI will show an error message to user.

5.1.4 Debugger



Keyboard shortcuts: User will use the keyboard shortcuts to manage the debugger operations which are: Pause/Resume, Step in/over/out.

Pause/Resume: User will press the Pause/Resume button. The debugging engine will stop or continue to control the execution of scripts. Call stack view and variables view are updated according to these operations.

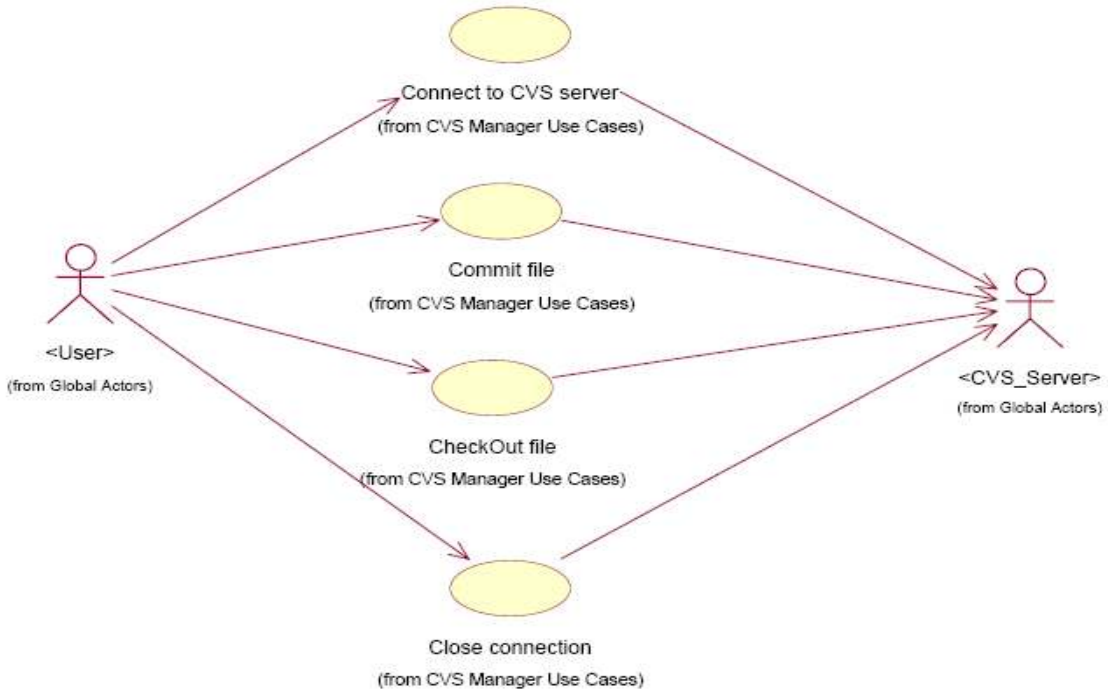
Step in/over/out: User will press the Step in/over/out button. The debugging engine will go in/over/out the execution step of the scripts its debugging.

Set/clear breakpoints: User will click the the line number at which he/she wants to set/clear breakpoints on the editor window. Breakpoint set/clear at this line. The debugging engine will stop/continue at breakpoints. Call stack view and variables view are updated according to these operations and the user will see the values of the variables at that breakpoints.

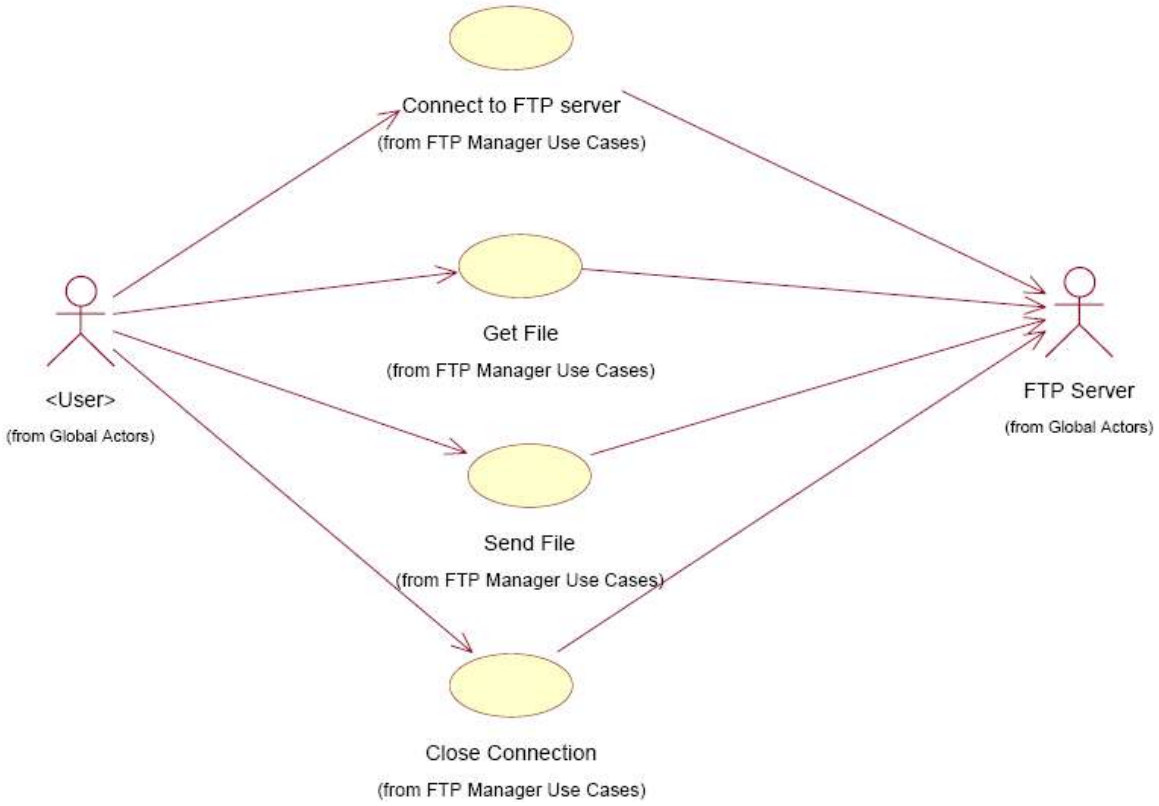
Call Stack view: When the debugger is stopped, the Call Stack view displays the list of active functions.

Variables view: When the debugger is stopped, the variables view displays values for the current function.

5.1.5 CVS Manager

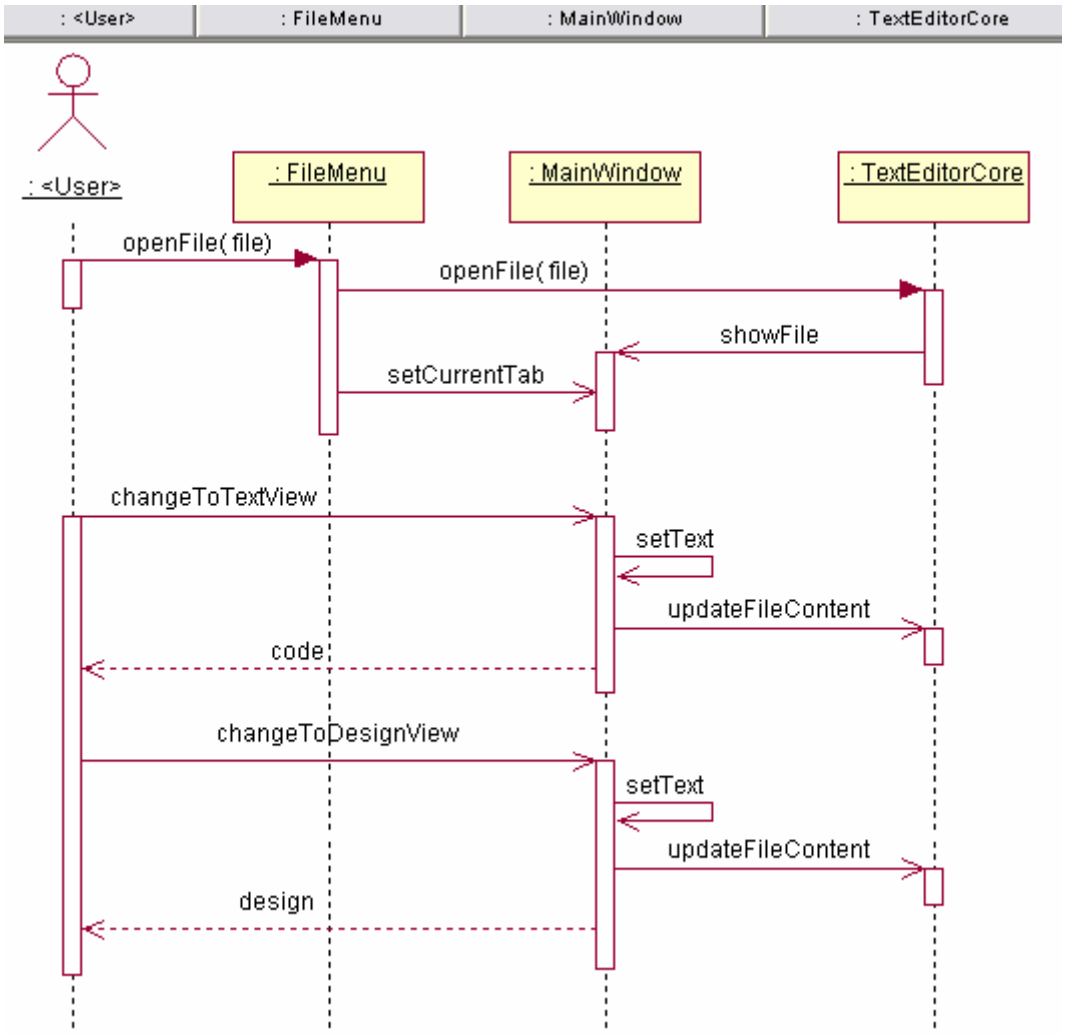


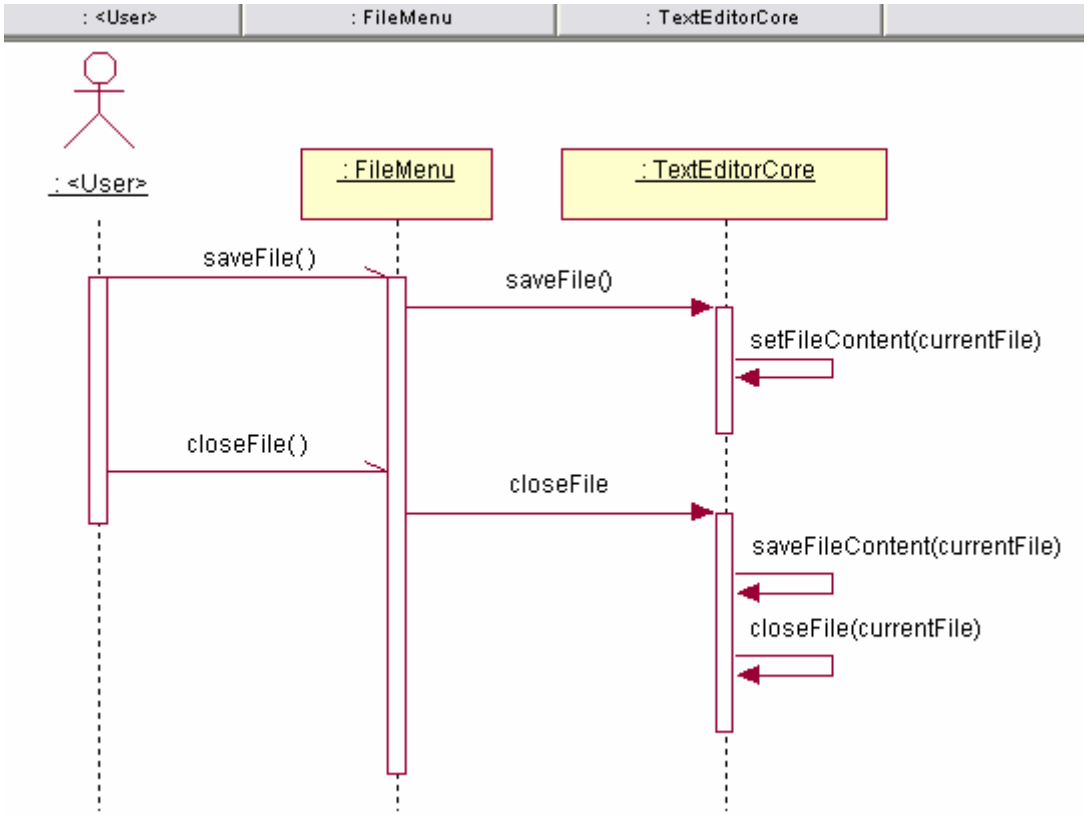
5.1.6 FTP Manager

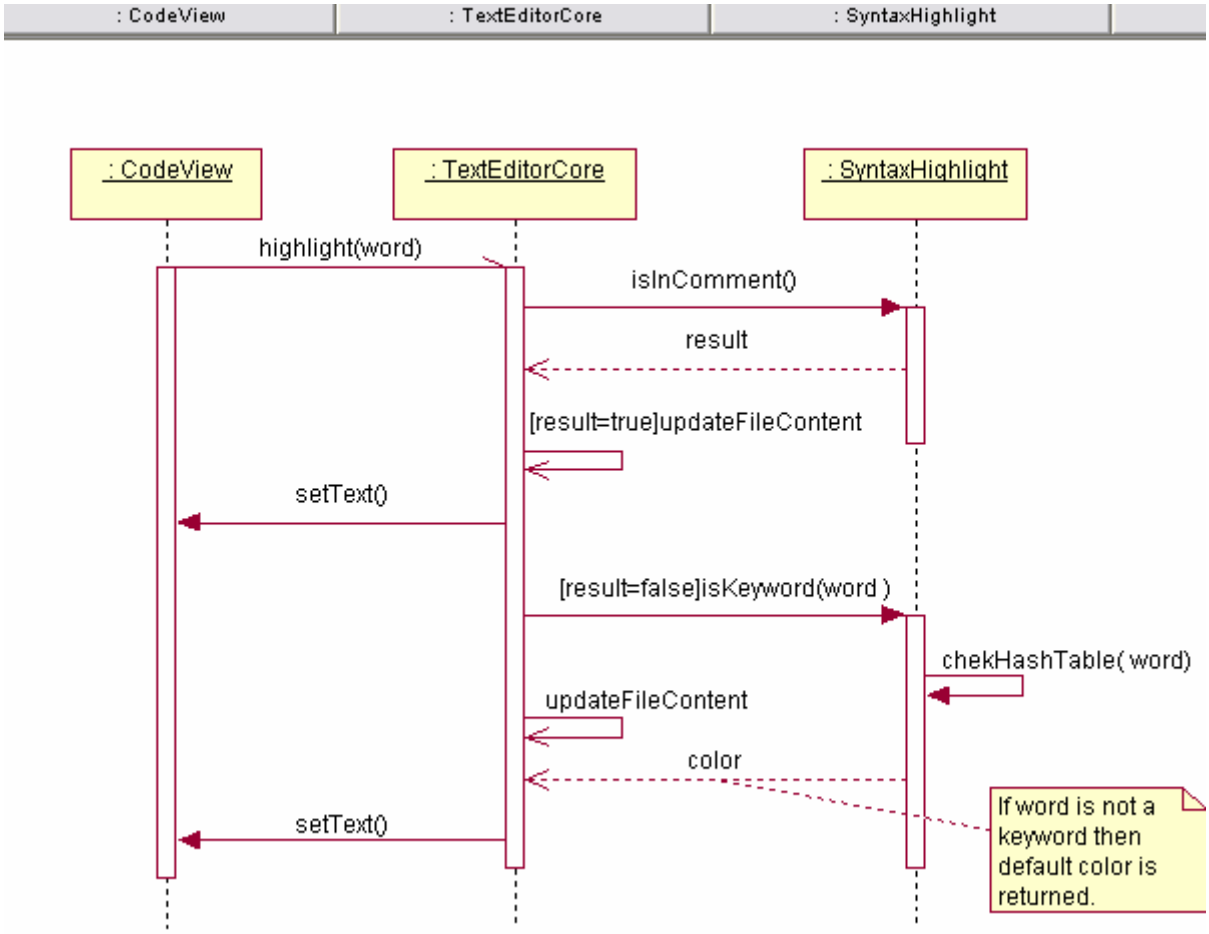


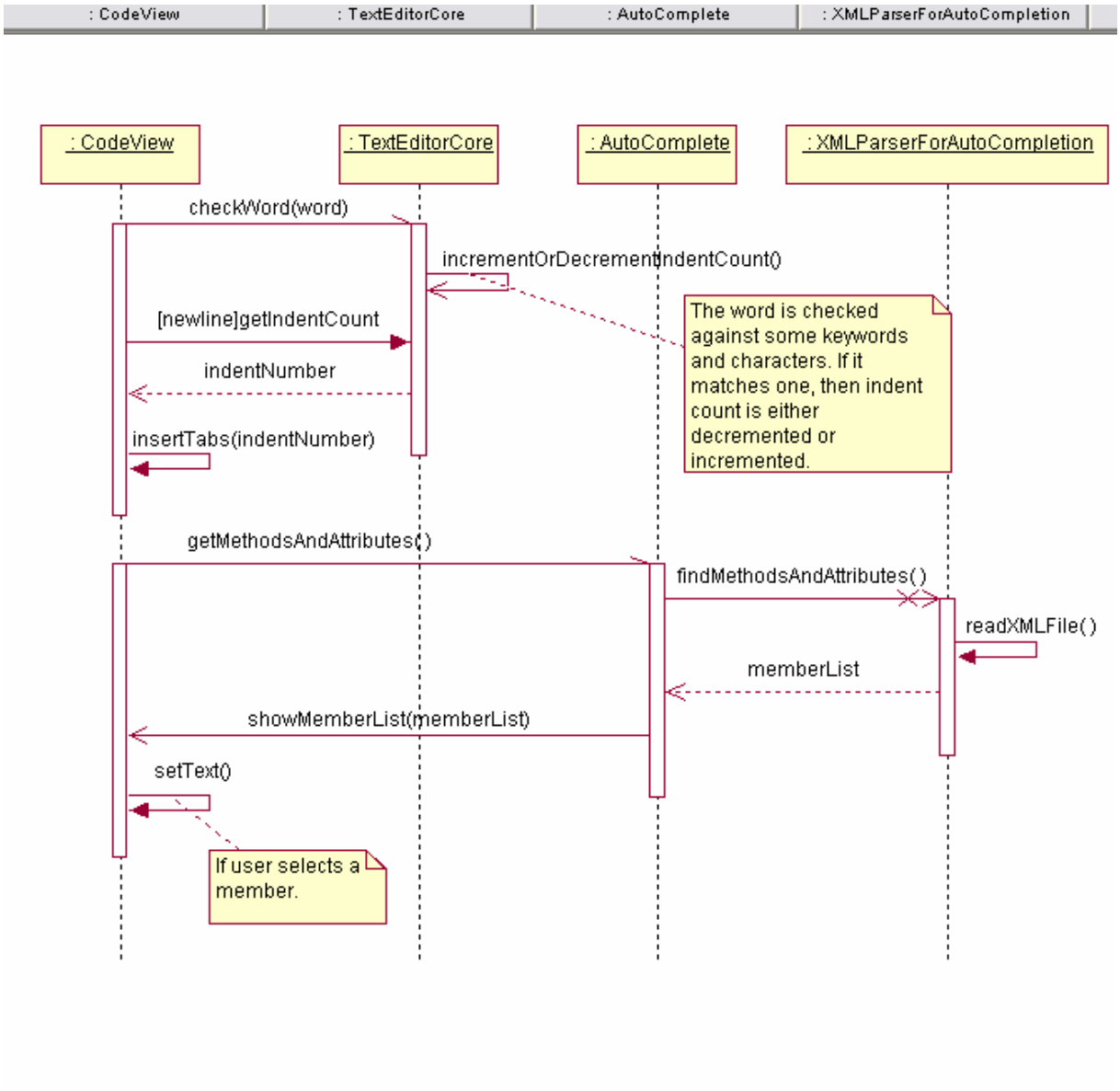
5.2 Dynamic View of the System

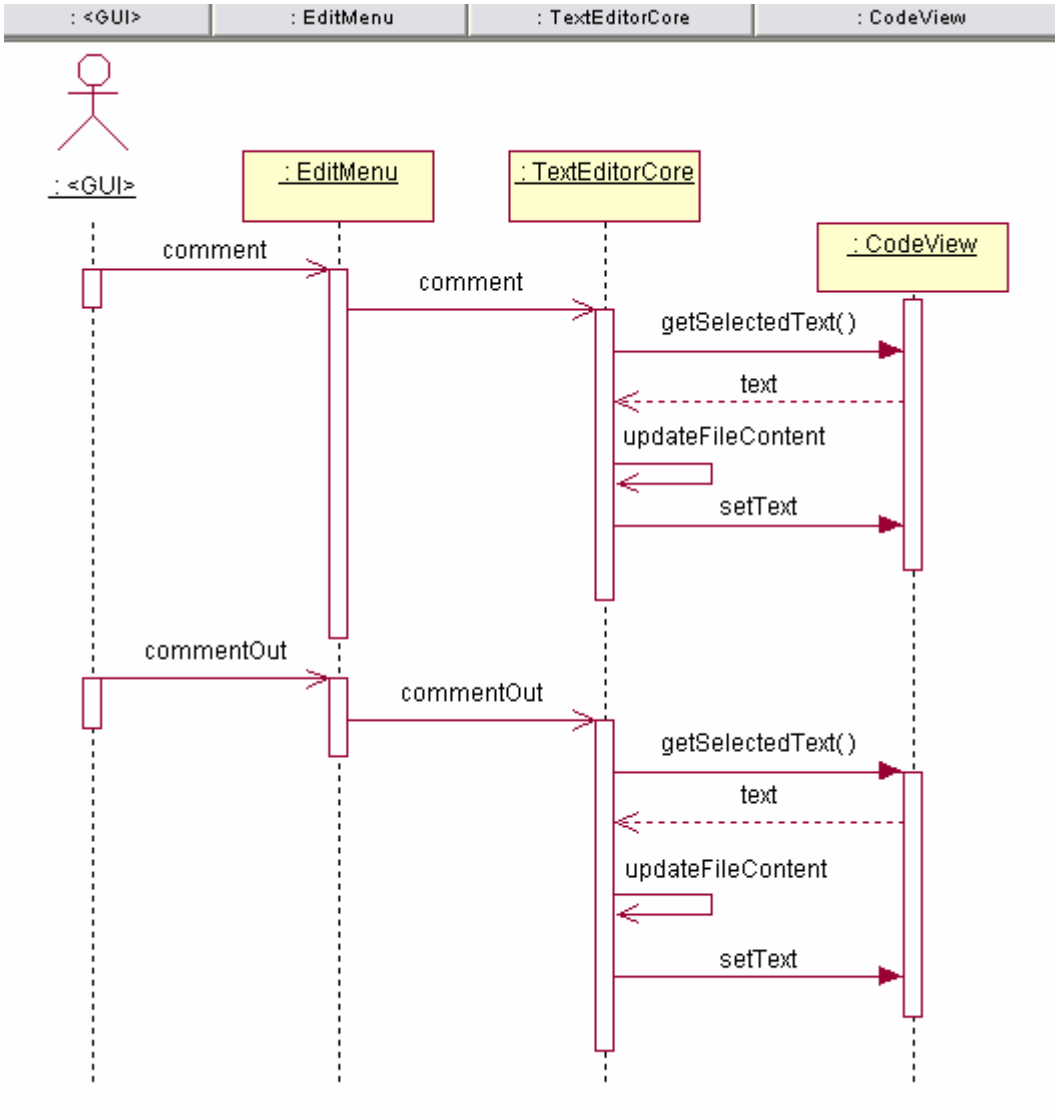
5.2.1 Text Editor

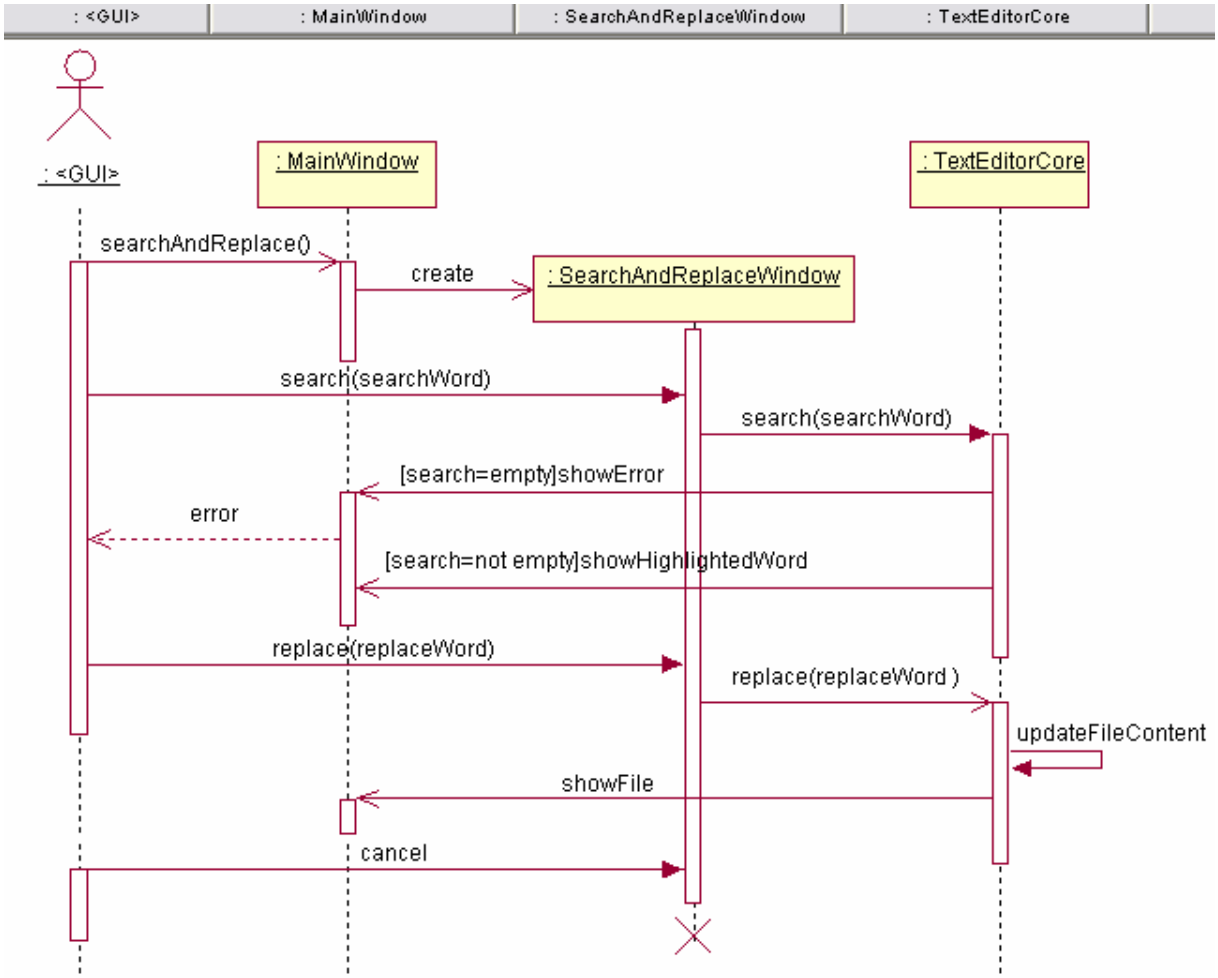


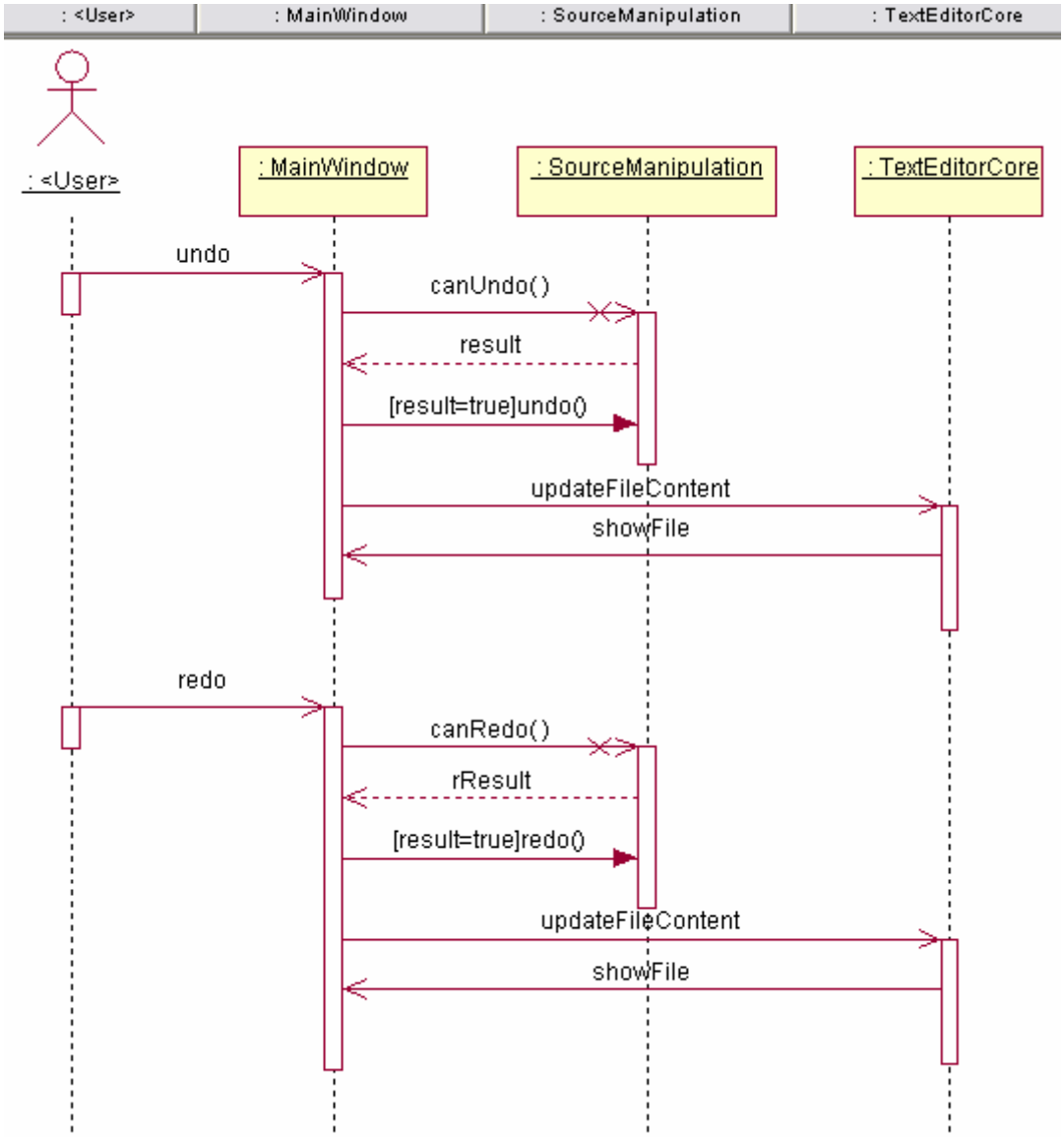

: <User>
: FileMenu
: TextEditorCore
: <User>
: FileMenu
: TextEditorCore
saveFile( )
saveFile()
setFileContent(currentFile)
closeFile( )
closeFile
saveFileContent(currentFile)
closeFile(currentFile)


: CodeView
: TextEditorCore
: SyntaxHighlight
: CodeView
: TextEditorCore
: SyntaxHighlight
highlight(word)
isInComment()
result
[result=true]updateFileContent
setText()
[result=false]isKeyword(word)
chekHashTable(word)
updateFileContent
color
setText()
If word is not a keyword then default color is returned.

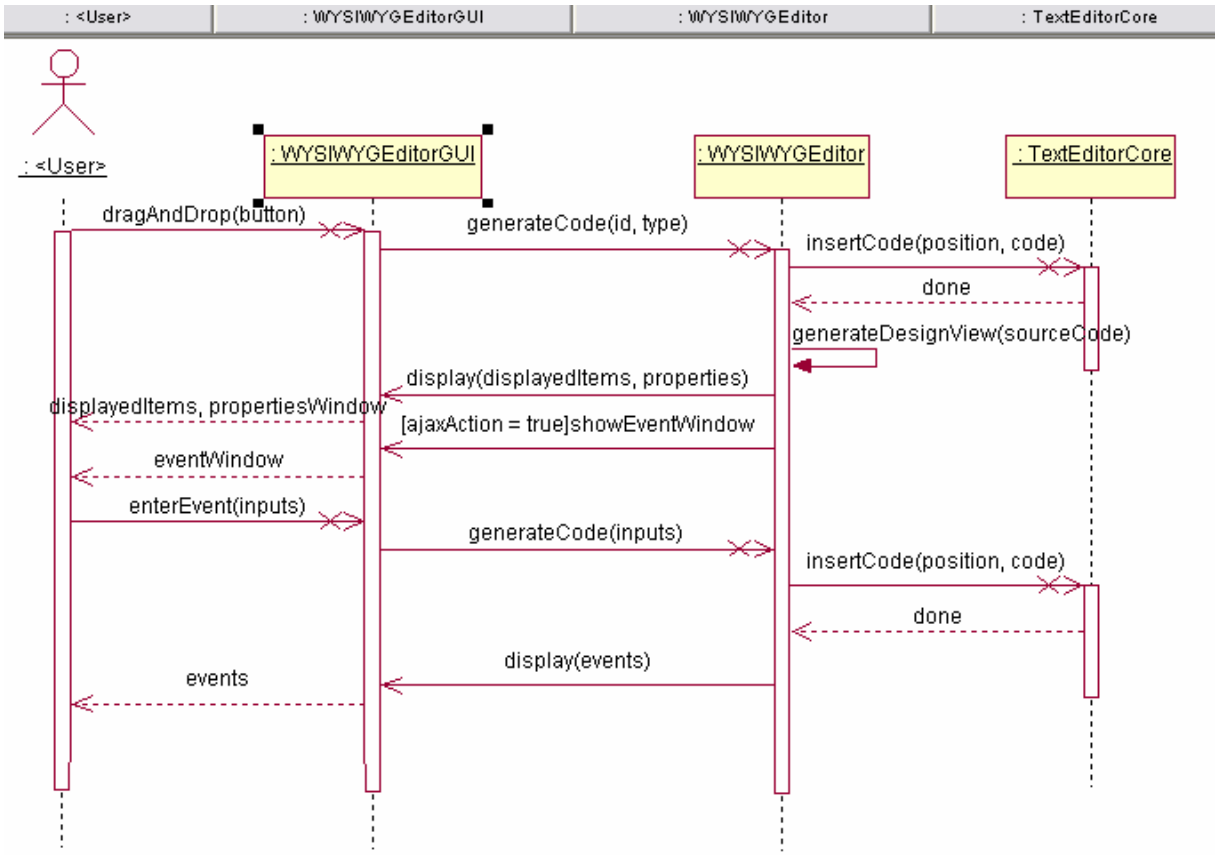

: CodeView
: TextEditorCore
: AutoComplete
: XMLParserForAutoCompletion
checkWord(word)
incrementOrDecrementIndentCount()
The word is checked against some keywords and characters. If it matches one, then indent count is either decremented or incremented.
[newline]getIndentCount
indentNumber
insertTabs(indentNumber)
getMethodsAndAttributes()
findMethodsAndAttributes()
readXMLFile()
memberList
showMemberList(memberList)
setText()
If user selects a member.

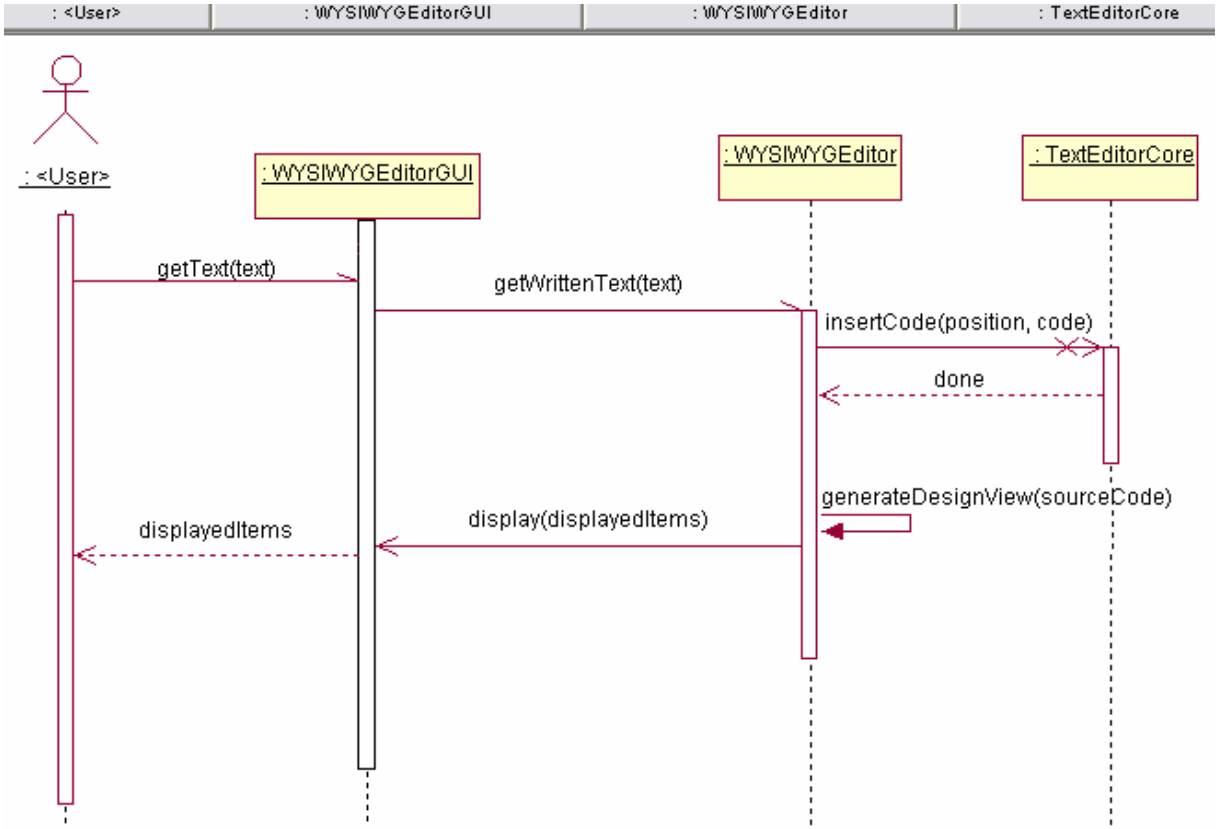

: <GUI>
: EditMenu
: TextEditorCore
: CodeView
: <GUI>
: EditMenu
: TextEditorCore
: CodeView
comment
comment
getSelectedText( )
text
updateFileContent
setText
commentOut
commentOut
getSelectedText( )
text
updateFileContent
setText

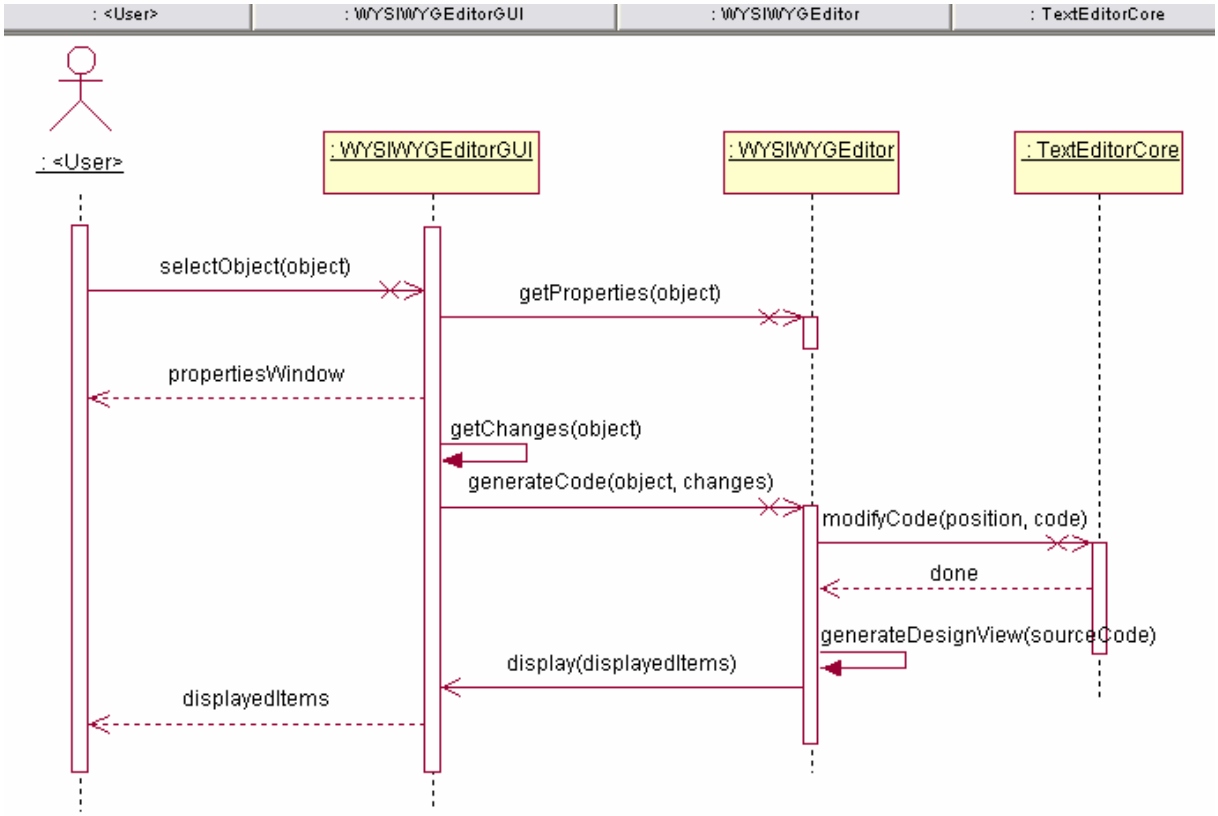

: <GUI>
: MainWindow
: SearchAndReplaceWindow
: TextEditorCore
: <GUI>
: MainWindow
: TextEditorCore
searchAndReplace()
create
: SearchAndReplaceWindow
search(searchWord)
search(searchWord)
[search=empty]showError
error
[search=not empty]showHighlightedWord
replace(replaceWord)
replace(replaceWord)
updateFileContent
showFile
cancel

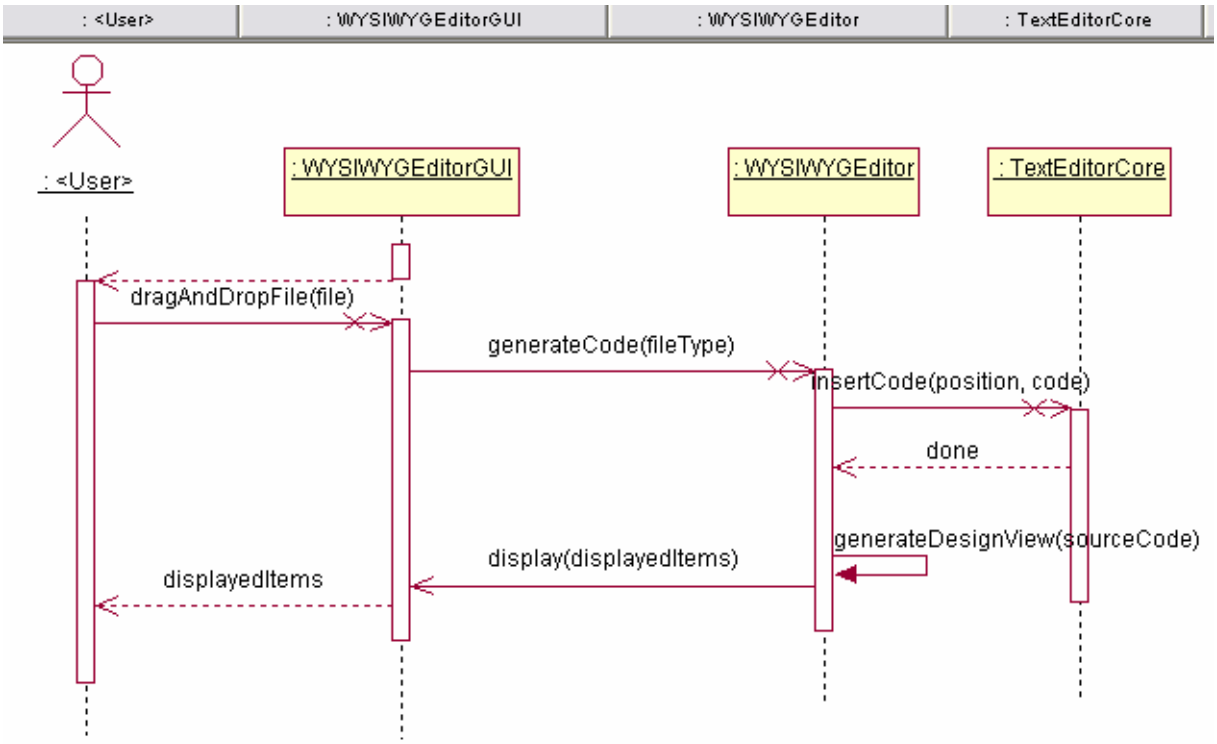

: <User>
: MainWindow
: SourceManipulation
: TextEditorCore
undo
canUndo()
result
[result=true]undo()
updateFileContent
showFile
redo
canRedo()
rResult
[result=true]redo()
updateFileContent
showFile

5.2.2 WYSIWYG Editor

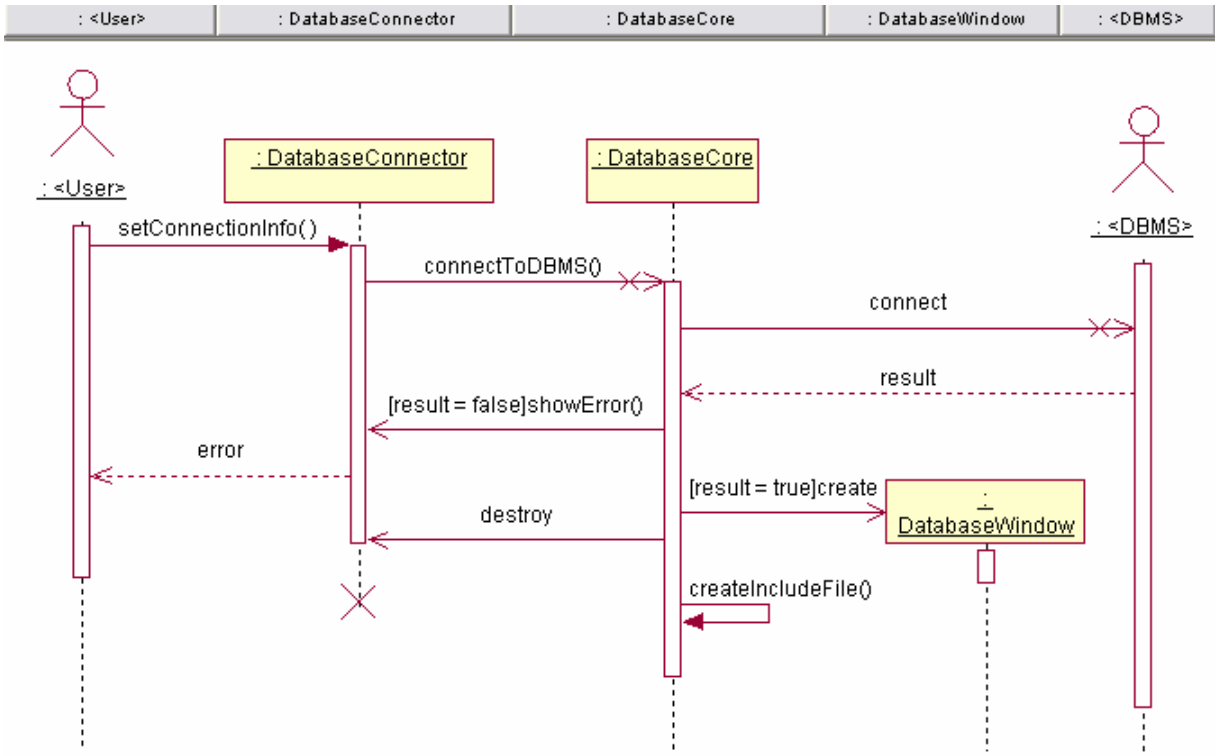


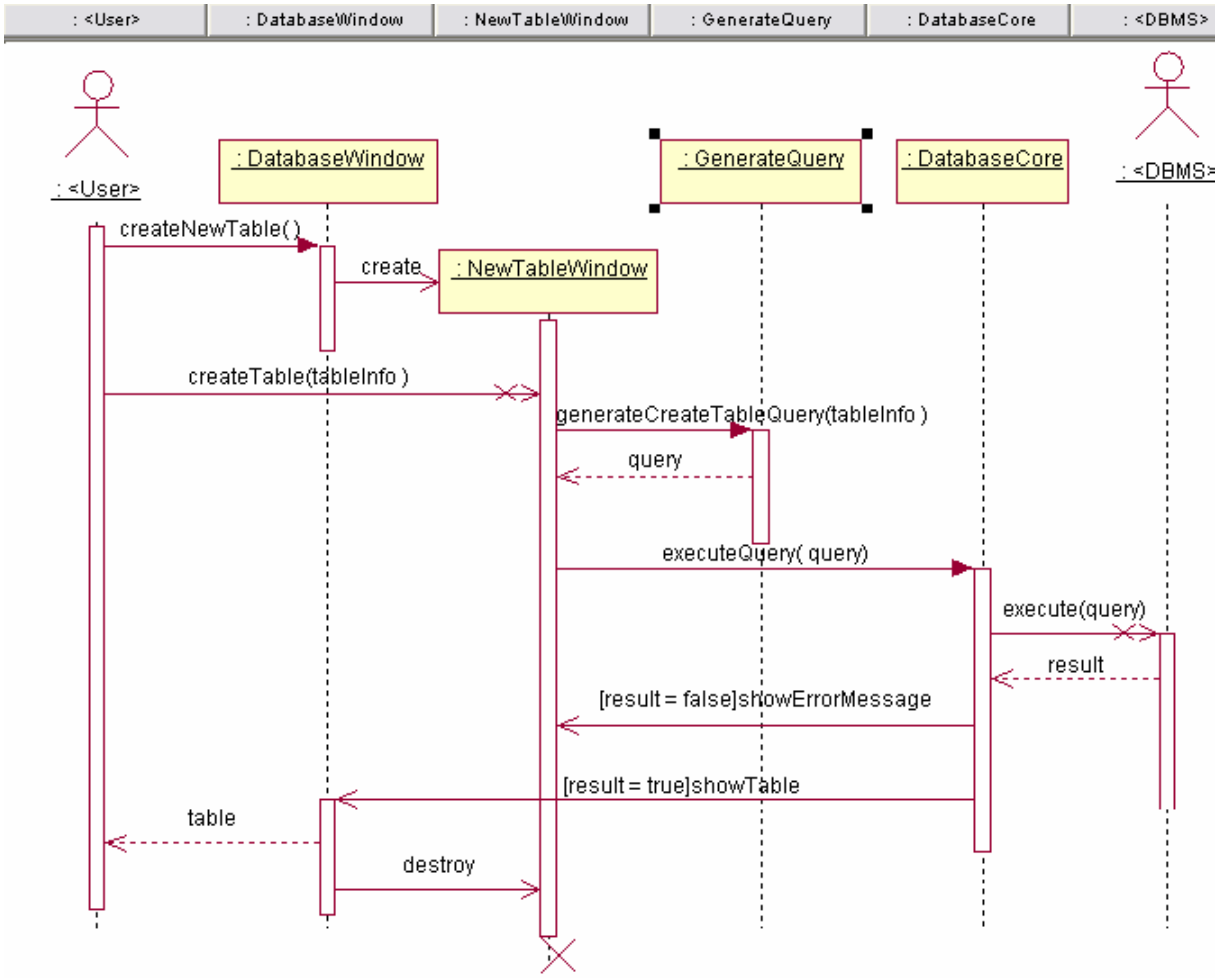

: <User>
: WYSIWYGEditorGUI
: WYSIWYGEditor
: TextEditorCore
getText(text)
getWrittenText(text)
insertCode(position, code)
done
generateDesignView(sourceCode)
displayedItems
display(displayedItems)

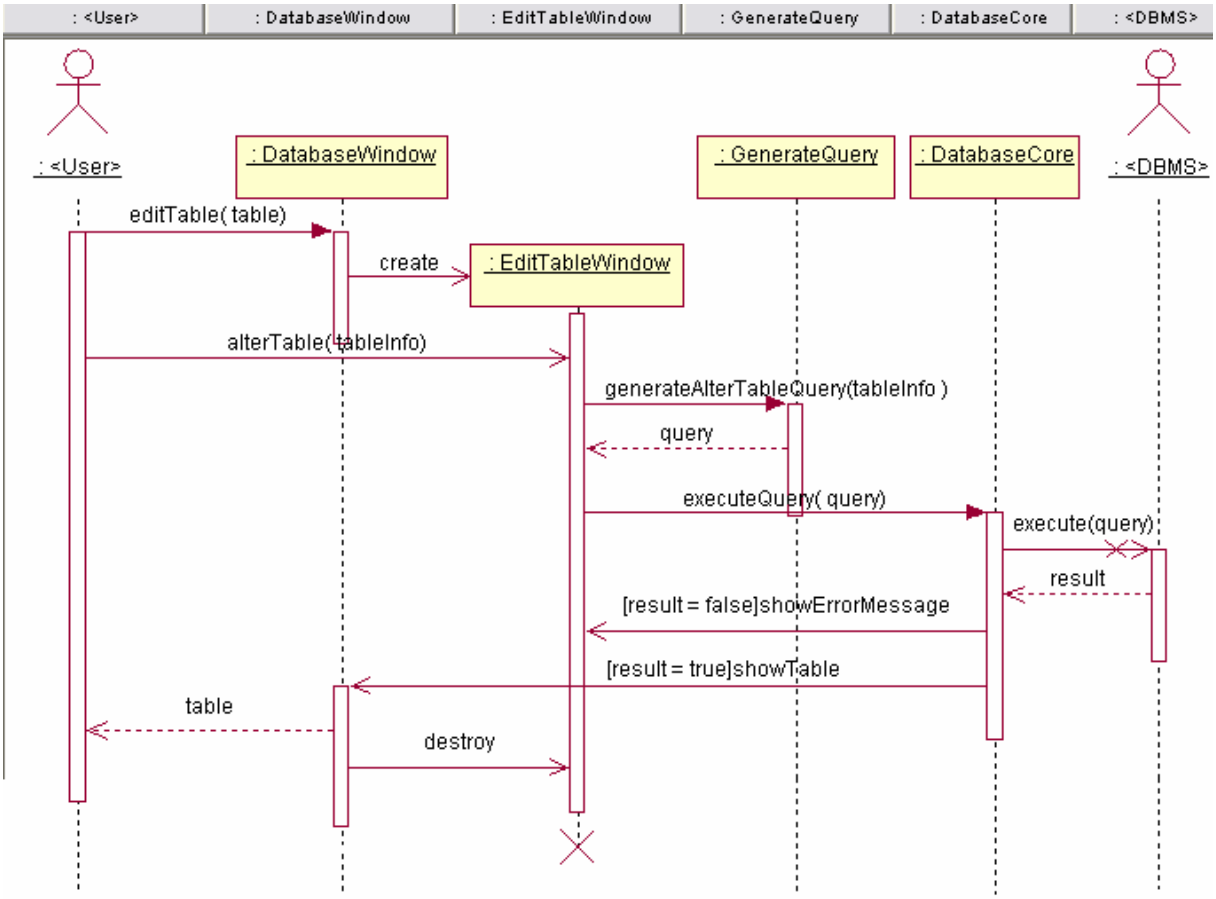

: <User>
: WYSIWYGEditorGUI
: WYSIWYGEditor
: TextEditorCore
selectObject(object)
getProperties(object)
propertiesWindow
getChanges(object)
generateCode(object, changes)
modifyCode(position, code)
done
generateDesignView(sourceCode)
display(displayedItems)
displayedItems

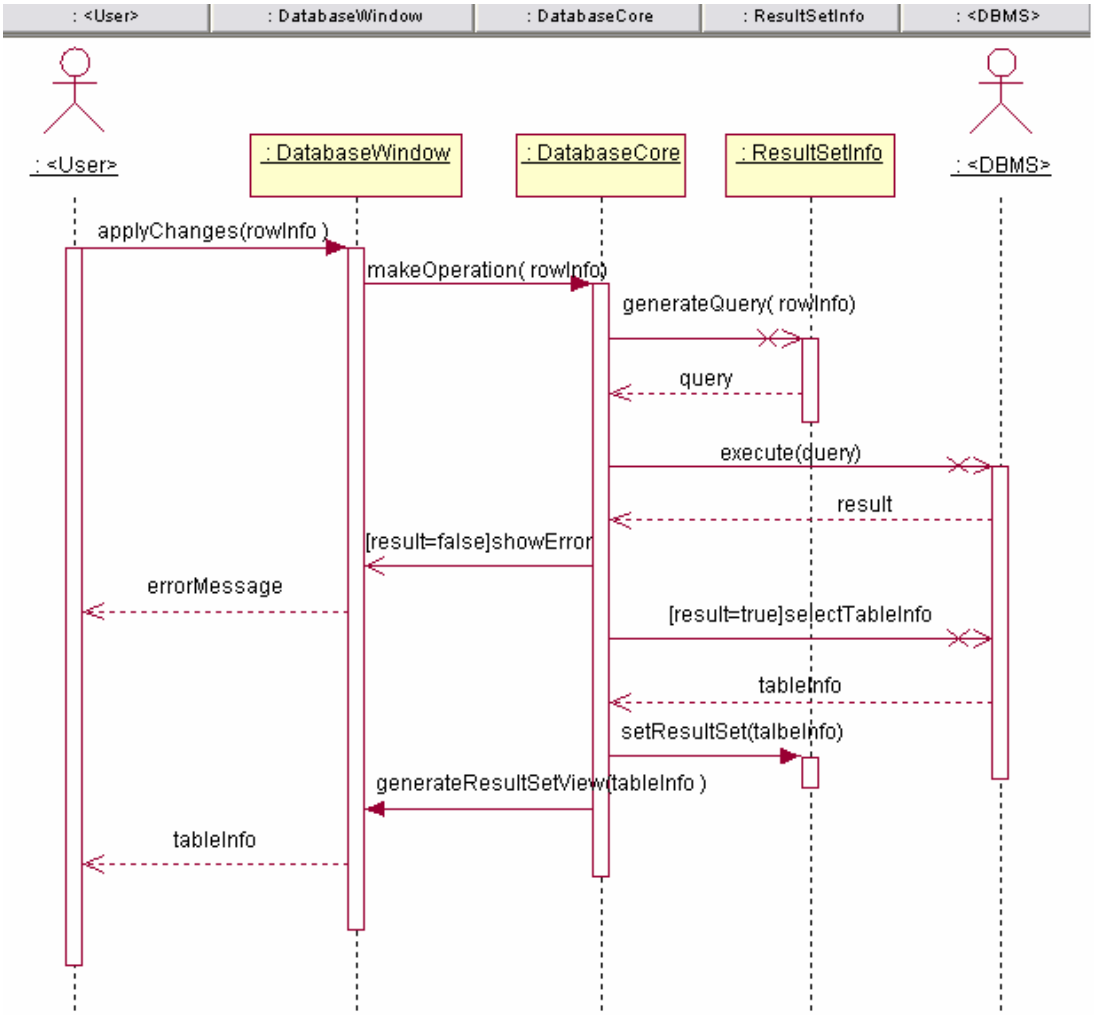


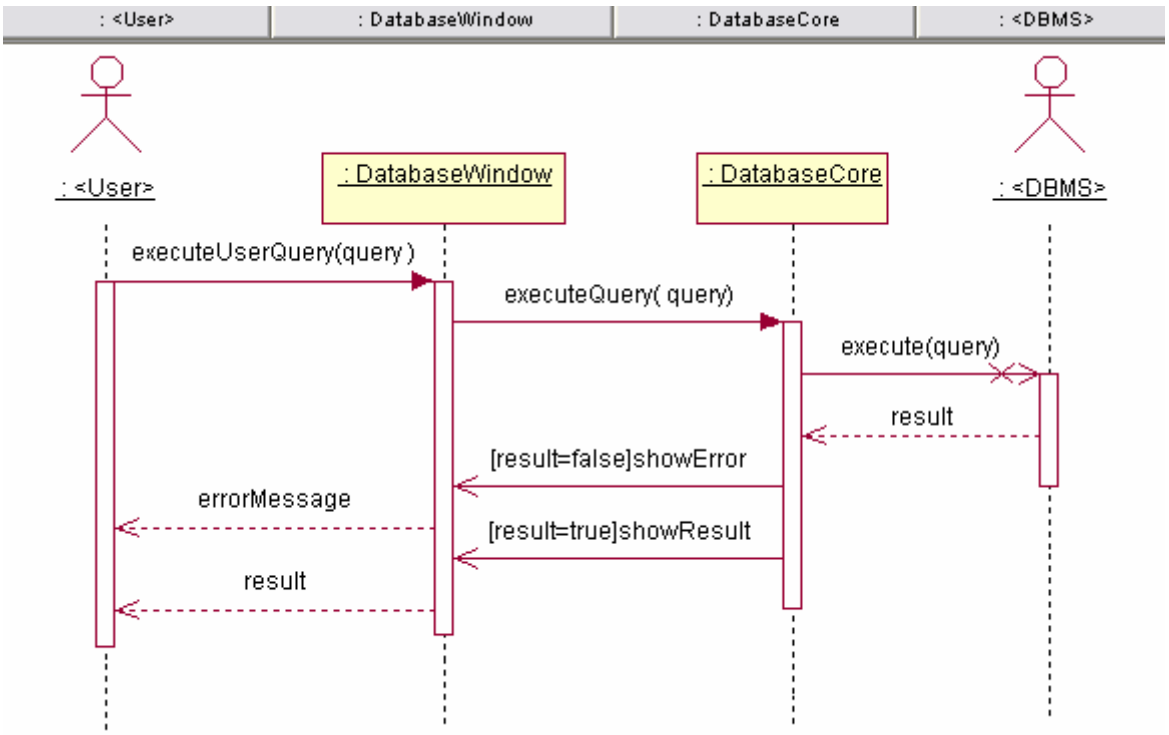
5.2.3 Database Editor



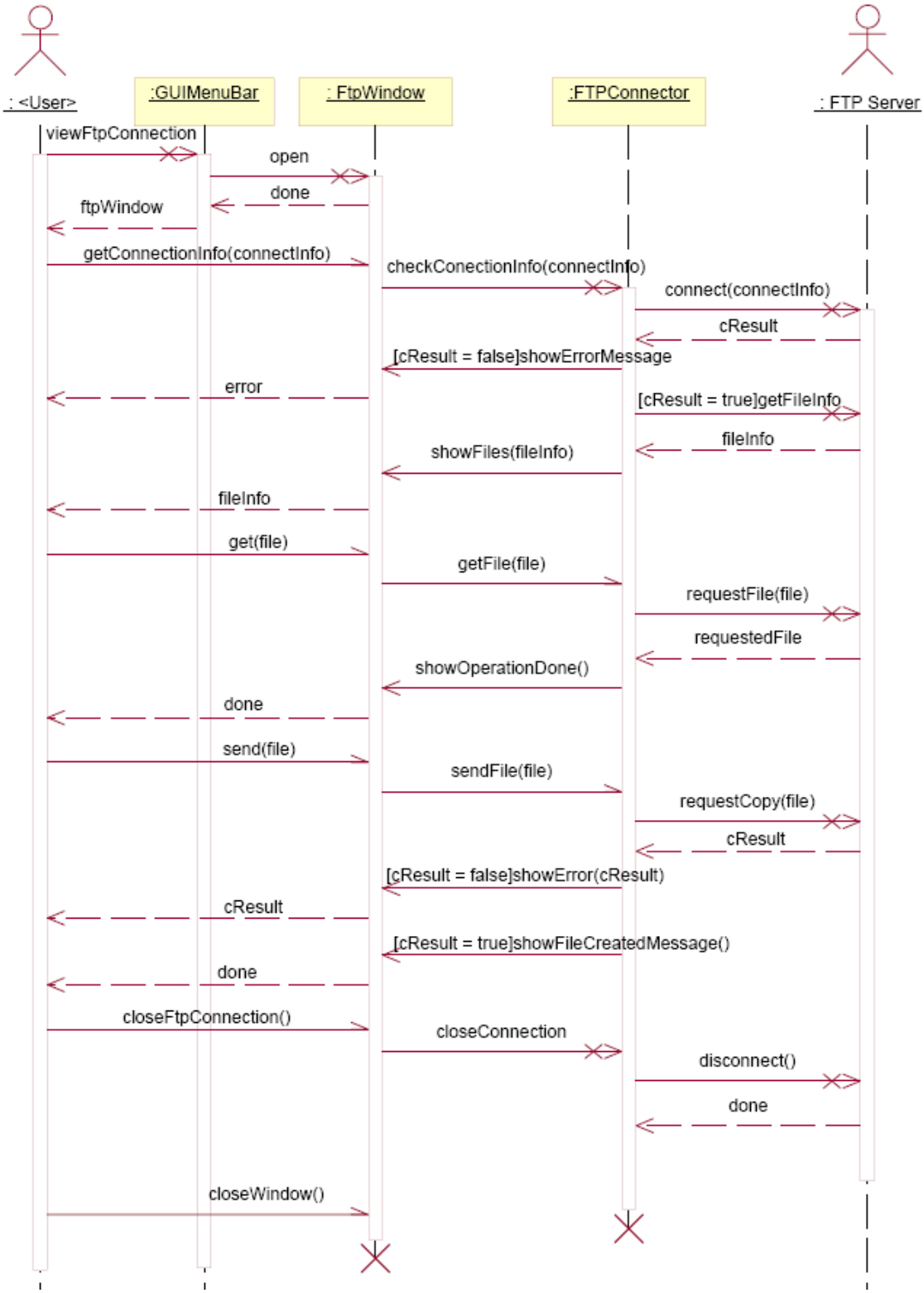

: <User>
: DatabaseWindow
: NewTableWindow
: GenerateQuery
: DatabaseCore
: <DBMS>
: <User>
: DatabaseWindow
: GenerateQuery
: DatabaseCore
: <DBMS>
createNewTable( )
create
: NewTableWindow
createTable(tableInfo)
generateCreateTableQuery(tableInfo)
query
executeQuery(query)
execute(query)
result
[result = false]showErrorMessage
[result = true]showTable
table
destroy


: <User>
: DatabaseWindow
: EditTableWindow
: GenerateQuery
: DatabaseCore
: <DBMS>
editTable(table)
create
alterTable(tableInfo)
generateAlterTableQuery(tableInfo)
query
executeQuery(query)
execute(query)
result
[result = false]showErrorMessage
[result = true]showTable
table
destroy

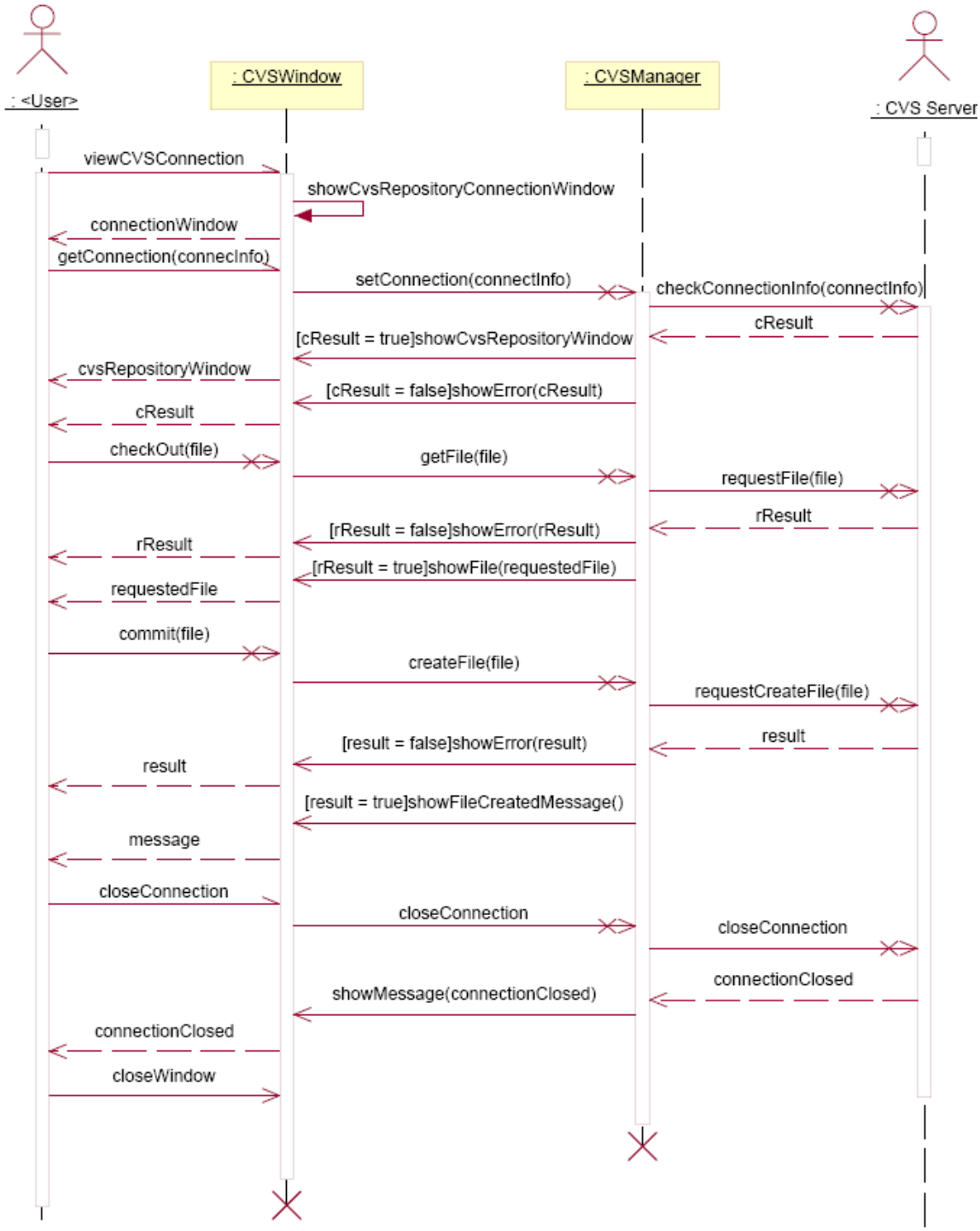

: <User>
: DatabaseWindow
: DatabaseCore
: ResultSetInfo
: <DBMS>
applyChanges(rowInfo)
makeOperation(rowInfo)
generateQuery(rowInfo)
query
execute(query)
result
[result=false]showError
errorMessage
[result=true]selectTableInfo
tableInfo
setResultSet(talbeInfo)
generateResultSetView(tableInfo)
tableInfo


: <User>
: DatabaseWindow
: DatabaseCore
: <DBMS>
executeUserQuery(query)
executeQuery(query)
execute(query)
result
[result=false]showError
errorMessage
[result=true]showResult
result

5.2.4 FTP Manager

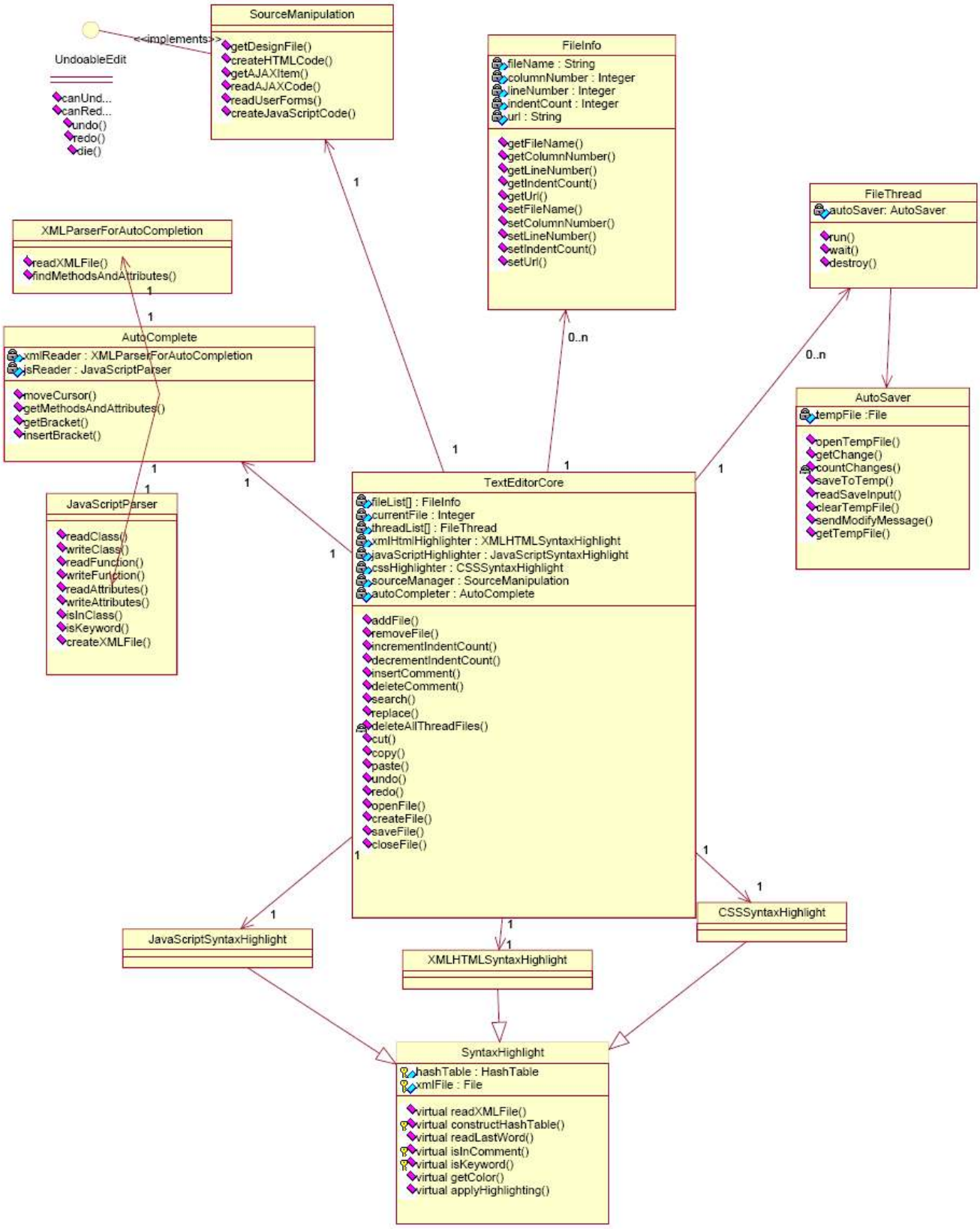


5.2.5 CVS Manager



5.3 Static View of the System

5.3.1 Text Editor



Class TextEditorCore:

Attributes:

Attribute Name	Attribute Type	Description
Filelist	Vector<FileInfo>	The list of the opened files
currentFile	Integer	Index of the current file
threadList	Vector<FileThread>	List of the threads opened for automatic save.
xmlHtmlHighlighter	XMLHTMLSyntaxHihlight	Operator that highlights the xml and html files.
javaScriptHighlighter	JavaScriptSyntaxHighlight	Operator that highlights JavaScript files.
cssHighlihter	CSSSyntaxHighlighting	Operator that highlights CSS files.
sourceManager	SourceManipulation	Operator that handles the code management between design and text editors.
autoCompleter	AutoComplete	Operator that handles automatic completion

Methods:

Method Name	Parameters	Return Value	Description
addFile	fileName: String url: String	Integer	It receives input from GUI and opens and adds adds the file specified with parameters to the file list.
removeFile	fileName: String url: String	Integer	It receives input from GUI and closes the file and removes sit form the list.
incrementIndentCount	Index: Integer	Void	It increments the indent count of the file who has the index same as the parameter in the filelist.
decrementIndentCount	Index: Integer	Void	It decrements the indent count of the file who has the index same as the parameter in the filelist.
insertComment	selectedLines: Text	Void	Inserts comment characters to the beginning of every selected line.

deleteComment	selectedLines: Text	Void	Deletes comment characters from the beginning of every selected line.
search	wantedExpression : String direction: Integer	Boolean	Searches the parameter in the file backward or forward and shows the found expression.
replace	old: String new: String direction: Integer	Boolean	Searches the old word in the given direction, replaces it with the new word.
deleteAllThreadFiles	Void	Void	It removes all threads
cut	toCut: Text	Void	It cuts the parameter from the file
copy	toCopy: Text	Void	It copies the parameter from the file
paste	void	void	It pastes the selected item to the place where cursor is
Undo	Void	Void	Undoes
Redo	Void	Void	redoes
openFile	fileName: String url: String	Void	Opens the specified file
createFile	Void	Void	Creates new file
saveFile	fileName: String url: String	Void	Saves the file to the specified place
closeFile	fileName: String url: String	Void	Closes the file.

Class File Info

Attributes:

Attribute Name	**Attribute Type**	**Description**
fileName	String	Name of the File
columnNumber	Integer	Column number of the marker of the file
lineNumber	Integer	Line number of the marker of the file
indentCount	Integer	Count of the indentation to the left or right
url	String	Url of the file

Methods:

Method Name	Parameters	Return Value	Description
getFileName	Void	String	Returns the filename
getColumnNumber	Void	Integer	Returns the columnNumber
getLineNumber	Void	Integer	Returns the lineNumber
getIndentCount	Void	Integer	Returns the indentCount
getUrl	Void	String	Returns the url
setFileName	fileName: String	Void	Sets filename
setColumnNumber	columnNumber: Integer	Void	Sets columnNumber
setLineNumber	lineNumber: Integer	Void	Sets lineNumber
setIndentCount	indentCount: integer	Void	Sets indentCount
setUrl	url: String	Void	Sets url

Class AutoComplete

Attributes:

Attribute Name	Attribute Type	Description
xmlReader	XMLParserForAutoCompletion	Reads the XML files for autocompletion
jsReader	JavaScriptParser	Reads the users's javascript files and create XML files for autocompletion

Methods:

Method Name	Parameters	Return Value	Description
moveCursor	Void	void	Moves the cursor to the correct position after an auto completion
getMethodsAndAttributes	Word: String	Vector<String>	Sends the methods and attributes of the written variable to the GUI
getBracket	Void	Character	Read the brackets
insertBracket	Bracket: Character	Void	Inserts the matching bracket of the read one.

Class XMLParserAutoCompletion:

Methods:

Method Name	Parameters	Return Value	Description
readXMLFile	Void	void	Reads theXML files
findMethodsAndAttributes	className:String	Vector<String>	Finds the methods and attributes of the given class.

Class JavaScriptParser:

Methods:

Method Name	**Parameters**	**Return Value**	**Description**
readClass	File : FileInfo	String	Gets the classes in the file
writeClass	classInfo : String	Void	Writes the class information to a XML file
readFunction	className: String functionName : String parameters : vector<String> returnType: String	Vector<String>	Gets the information of the function
writeFunction	functionInfo : Vector<String>	Void	Writes the information to a XML file
readAttributes	className :String attributeInfo : Vector<String>	Vector<String>	Gets the attributes of a class
writeAttributes	attributeInfo : Vector<String>	Void	Writes the information to a XML file
isInClass	Word : String	Boolean	Tests whether the word belongs to a class or not
isKeyWord	Word: String	Boolean	Tests whether the given word is a keyword or not.
createXMLFile	Void	Void	Creates the XML file for hiding information.

Class SyntaxHighlighting:

Attributes:

Attribute Name	**Attribute Type**	**Description**
hashTable	Hashtable	It is used for hashing the keywords with colors
xmlFile	File	Stores the keywords

Methods:

Method Name	Parameters	Return Value	Description
readXMLFile	Void	Void	Reads the XML file
constructHashTable	Void	Void	Constructs the hash table from the read XML file
readLastWord	Void	Void	Reads the text user has written and gets the last word
isInComment	Word: String	Boolean	Tests whether the input is in comment
isKeyWord	Word: String	Boolean	Tests whether the input is a keyword
getColor	Word: String	String	Finds and returns the color of the candidate word to highlight.
applyHighlighting	Word: String Color: String	Void	Applies highlighting to the specified word with specified color.

Class FileThread:

Attributes:

Attribute Name	**Attribute Type**	**Description**
autoSaver	AutoSaver	It is the operator that is responsible for automatic save

Methods:

Method Name	Parameters	Return Value	Description
Run	Void	Void	Run method of thread class
Wait	Void	Void	Wait method of thread class
Destroy	void	Void	Destroy method of thread class

Class AutoSaver:

Attributes:

Attribute Name	Attribute Type	Description
tempFile	File	The file opened for saving the changes automatically

Methods:

Method Name	Parameters	Return Type	Description
openTempFile	Void	Void	Creates temporary file
getChange	userFile : File	Void	Specifies the change the user has made if it exists
countChanges	Void	Integer	Return the number of changes user has made
saveToTemp	userFile : File	Void	Saves the file to the temporary file
readSaveInput	Void	Void	Tests if the user has pressed to save button
clearTempFile	Void	Void	Clears the temporary file
sendModifyMessage	Void	Void	After the user has reopened the file after a crash, this functions sends a modification message to the user
getTempFile	Void	File	Returns tempFile

Class SourceManipulation:

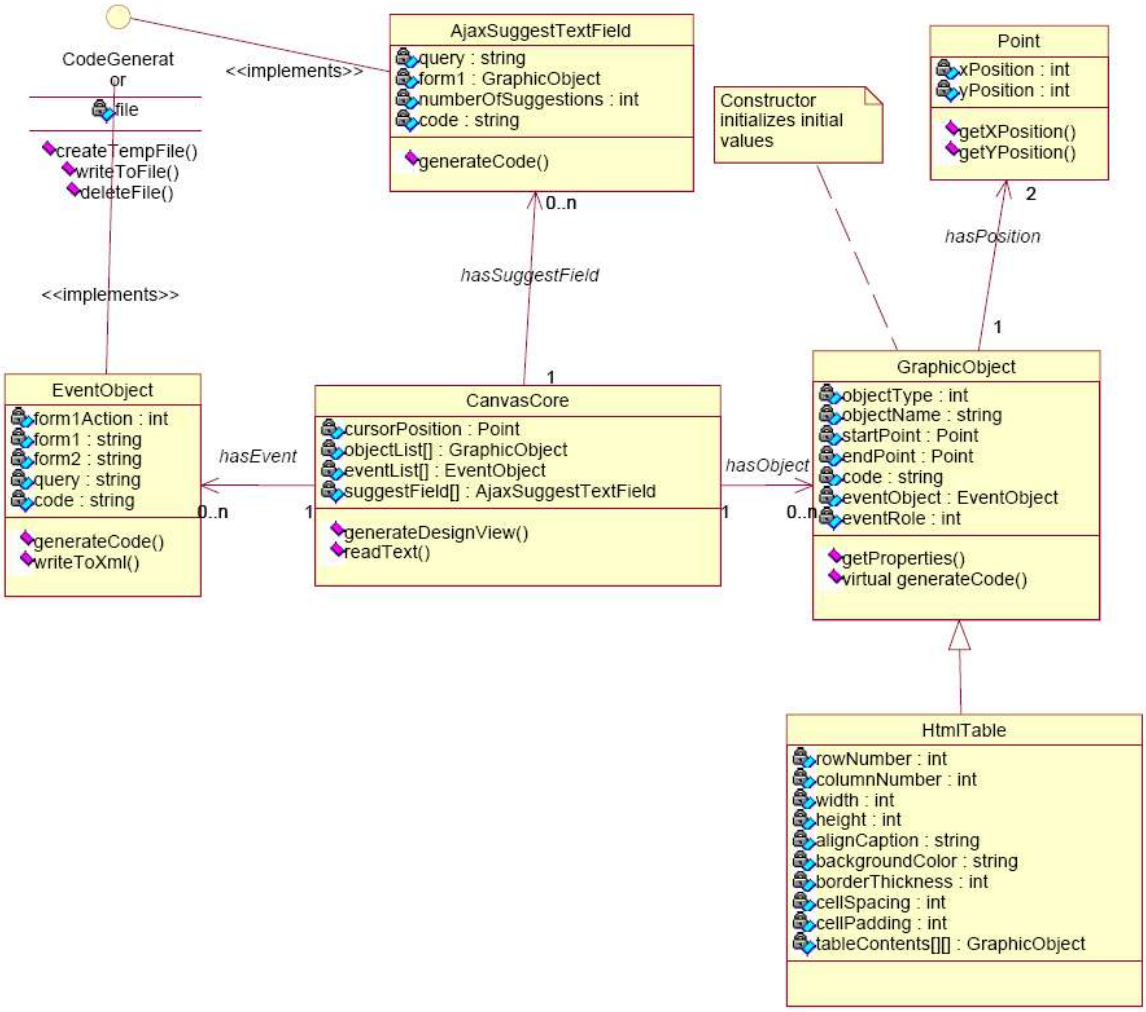
Methods:

Method Name	**Parameters**	**Return Type**	**Description**
getDesignFile	designFile : File	Void	Reads the design file
createHTMLCode	designFile : File	Void	Creates the html code of the design file
getAjaxItem	designFile: File	fileName : String	Reads the ajax items in the design file
readAjaxCode	fileName : String	Void	Reads the AJAX code of the specified file.
readUserForm	userForm: File	Void	Reads the file created when the user has filled the form to make an ajax action.
createJavaScriptCode	Void	Void	Creates JavaScript code from the related files.

Interface UndoableEdit:

Method Name	Parameters	Return	Desciption
canUndo	Void	Boolean	Tests whether undo action can be performed or not
canRedo	Void	Boolen	Tests whether redo action can be performed or not
Undo	Void	Void	Undo
Redo	Void	Void	Redo

5.3.2 WYSIWYG Editor



CanvasCore Class:

Attributes of Class:

Attribute Name	Attribute Type	Description
cursorPosition	class Point	Stores the current position of the cursor
objectList[]	vector<class GraphicObject>	The List of GraphicalObject class instances
eventList[]	vector<class EventObject>	The List of EventObject class instances
suggestField[]	vector<classAjaxSuggestTextField>	The List of AjaxSuggestTextField class instances

Methods of Class:

Method Name	Parameters	Return Type	Description
generateDesignView	string	string	Read the source code and generate the design view
readText	void	string	Get the text input from user

Graphic Object Class :

Attributes of Class:

Attribute Name	Attribute Type	Description
objectType	int	Stores the type of object
objectName	string	Strores the name of object
startPoint	class Point	Stores the start point coordinates of object
endPoint	class Point	Stores the end point coordinates of object
code	string	Stores the related code for generating the Design view of object
eventObject	class EventObject	If object is an ajax action stores the eventObject of ajax action
eventRole	int	If object is an ajax action stores the role of ajax action

Methods of Class:

Method Name	Parameters	Return Type	Description
getProperties	void	string	returns the properties of Graphical Object
virtual generateCode	void	string	virtual function for creating code according to the properties.

EventObject Class:

Attributes of Class:

Attribute Name	Attribute Type	Description
form1Action	int	Stores the action type for ajax action
form1	string	Stores the info of first related form
form2	string	Stores the info of second related form
query	string	Stores desired query for custom ajax action
code	string	Stores code for custom ajax action

Methods of Class:

Method Name	Parameters	Return Type	Description
generateCode	void	string	generates required code for custom ajax action
writeToXml	void	boolean	writes the required information of custom ajax for reuse.

HtmlTable Class:

Attributes of Class:

Attribute Name	Attribute Type	Description
rowNumber	int	Stores the row number of html table
columnNumber	int	Stores the column number of html table
width	int	Stores the row number of

		html table
height	int	Stores the row number of html table
alignCaption	string	Stores the align info of html table
backgroundColor	string	Stores the bg color info of html table
BorderThickness	int	Stores the border thickness info of html table
cellSpacing	int	Stores the cell spacing info of html table
cellPadding	int	Stores the cell padding info of html table
tableContents[][]	string	Stores the contents of rows and column

Point Class:

Attributes of Class:

Attribute Name	Attribute Type	Description
xPosition	int	stores the line number information
yPosition	int	stores the character information

Methods of Class:

Method Name	Parameters	Return Type	Description
getXpositon	void	int	returns the xPosition
getYposition	void	int	returns the yPosition

AjaxSuggestTextField Class:

Attributes of Class:

Attribute Name	Attribute Type	Description
query	string	It stores the sql query which the user has entered to use the AJAX application
form1	class GraphicObject	It stores the information of which element the suggestion will show
numberOfSuggestions	int	It stores the number of suggestions

code	string	It stores the JavaScript code related to the AJAX action

Methods of Class:

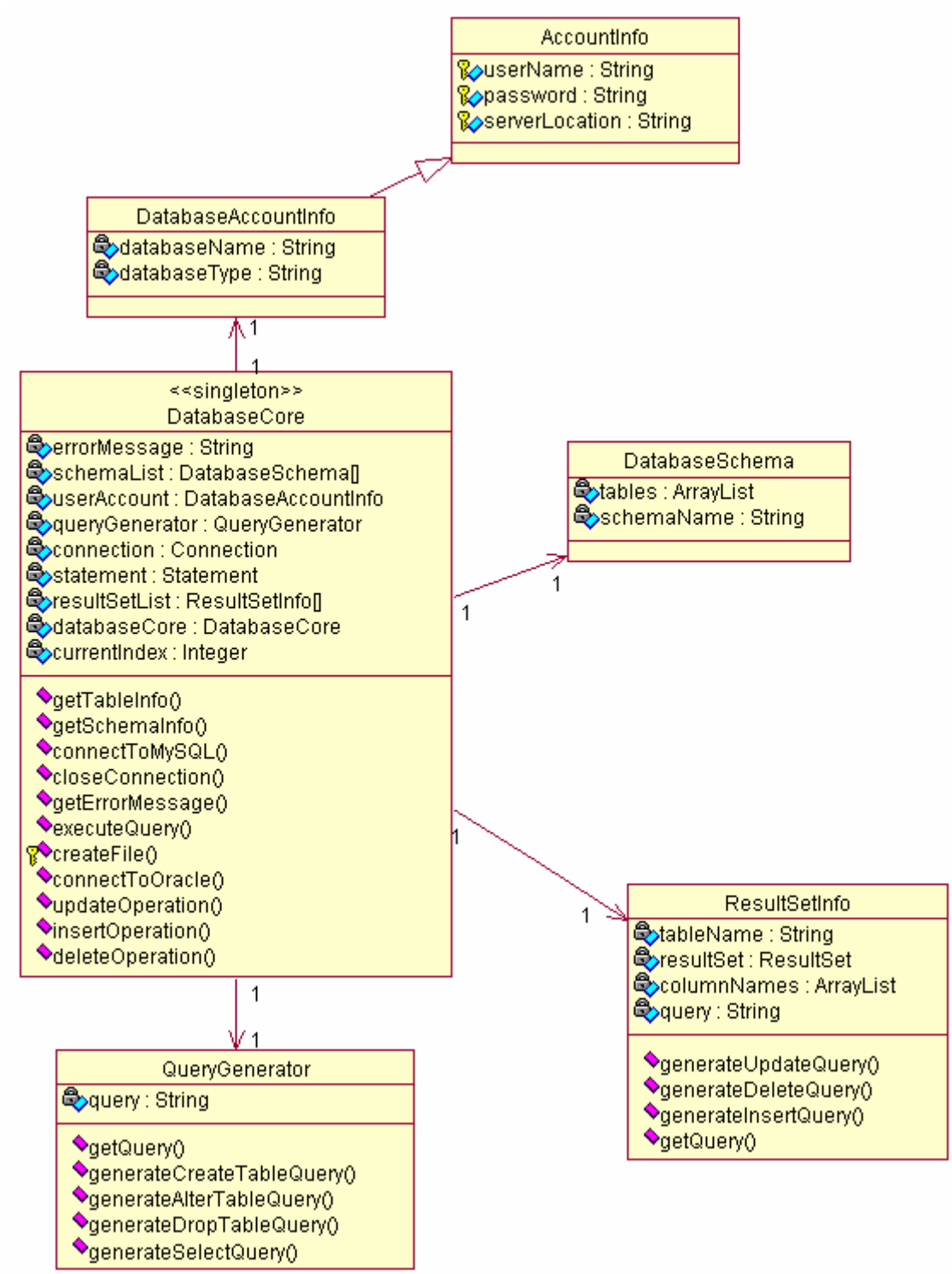
Method Name	Parameters	Return Type	Description
generateCode	void	string	It generates the necessary code for the action.

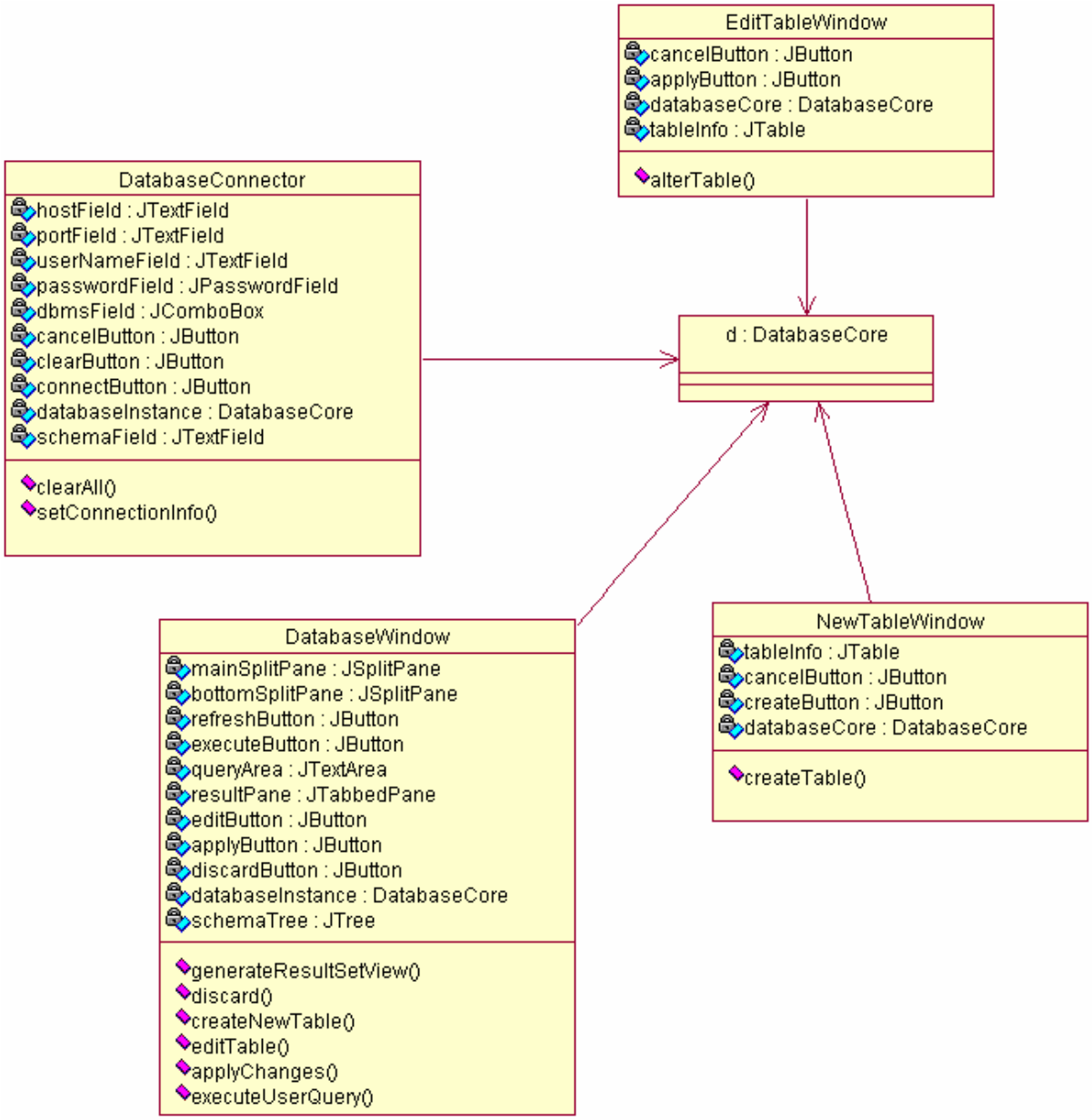
CodeGenerator Interface:

Methods of Interface:

Method Name	Parameters	Return Type	Description
createTempFile	void	boolean	creates a temporary file for sending codes to Text Editor
writeToFile	void	boolean	writes the codes
deleteFile	void	boolean	delete the temporary file after Text Editor gets the codes

5.3.3 Database Editor, CVS Manager and FTP Manager





DatabaseConnector Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
hostField	JTextField	Text field for entering host name of the database server.
portField	JTextField	Text field for entering the port number of the database server.

userNameField	JTextField	Text field for entering a username for connecting to the database server.
passwordField	JPasswordField	Password field for entering a password for connecting to the database server.
dbmsField	JComboBox	A combo box for selecting which type of DBMS (MySQL or Oracle) will be connected to.
schemaField	JTextField	Text field for entering a database for connecting to the database server.
cancelButton	JButton	When pressed closes the connection screen.
clearButton	JButton	When pressed Clears all of the fields.
connectButton	JButton	When pressed a connection is tried to be established using the entered information.
databaseInstance	DatabaseCore	An instance of the class DatabaseCore. Used for connecting to a database server.

Methods of the Class:

Method Name	Parameters	Return Type	Description
clearAll	void	void	Clears all of the text fields and the password field.
setConnectionInfo	void	void	Calls the appropriate connect method of databaseInstance and sets the connection if connection is successful info.

EditTableWindow Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
tableInfo	JTable	Holds the column information of the table to be editted.
cancelButton	JButton	When pressed closes the edit table window.
applyButton	JButton	When pressed applies the changes made to the table.
databaseCore	DatabaseCore	An instance of the DatabaseCore class.

Methods of the Class:

Method Name	Parameters	Return Type	Description
alterTable	void	void	Applies the changes done to the table by using databaseCore which uses its generateQuery attribute's generateAlterTableQuery function and then executes the query.

NewTableWindow Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
tableInfo	JTable	For entering information of the table to be created.
cancelButton	JButton	When pressed closes the new table window.
createButton	JButton	When pressed a new table with the entered information is created.
databaseCore	DatabaseCore	An instance of the DatabaseCore class.

Methods of the Class:

Method Name	Parameters	Return Type	Description
createTable	void	void	Creates the table by using databaseCore which uses its generateQuery attribute's generateCreateTableQuery function and then executes the query.

DatabaseWindow Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
mainSplitPane	JSplitPane	Main split pane for splitting the window. Top part holds the queryArea, executeButton and refreshButtons. Bottom part holds bottomSplitPane.
bottomSplitPane	JSplitPane	For splitting bottom part of the mainSplitPane into two. Left part holds schemaTree. Right part holds resultPane, editButton, applyButton and discardButton.
refreshButton	JButton	When pressed resultPane is refreshed.
executeButton	JButton	Pressed to execute a user entered query.
queryArea	JTextArea	Text area for writing queries.
resultPane	JTabbedPane	Result of the last executed query.
editButton	JButton	When pressed table rows can be editted.
applyButton	JButton	When pressed changes made to rows of a table are applied.
discardButton	JButton	When pressed discards the changes made to a table's rows.
schemaTree	JTree	Holds the information about the tables of the schemas in a connected database server.
databaseInstance	DatabaseCore	Instance of the DatabaseCore class.

Methods of the Class:

Method Name	Parameters	Return Type	Description
generateResultSetView	queryResult: *ResultSet*	void	Updates resultPane with the parameter it takes.
discard	void	void	Discards changes made to rows of a table.
createNewTable	void	void	Opens new table window by creating an instance of NewTableWindow class.
editTable	void	void	Opens edit table window by creating an instance of EditTableWindow class.
applyChanges	void	void	Applies changes made to rows of a table by using methods of databaseInstance.
executeUserQuery	query: *String*	void	Calls executeQuery method of databaseInstance and passes query as a parameter to it.

AccountInfo Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
userName	String	Holds the username used for connection.
password	String	Holds the password used for connection.
serverLocation	String	Holds the url of the server used for connection.

DatabaseAccountInfoClass:

Attributes of the Class:

Attribute Name	Attribute Type	Description
databaseName	String	Holds the name of the database connected to.
databaseType	String	Holds the type of the DBMS connected to.

QueryGenerator Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description

query	String	Holds the query generated.

Methods of the Class:

Method Name	Parameters	Return Type	Description
getQuery	void	String	Returns the query.
generateCreateTableQuery	tableName: *String,* columnNames: *ArrayList*	String	Generates a create table query according to the parameters it takers and saves it into query.
generateAlterTableQuery	tableName: *String*, newValues: *ArrayList*	String	Generates an alter table query according to the parameters it takes and saves it into query.
generateDropTableQuery	tableName: *String*	String	Generates a drop table query and saves it into query.
generateSelectQuery	tableName: *String*	String	Generates a select query according to the parameter it takes and saves it into query.

DatabaseCore Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
errorMessage	String	Holds the error message which is shown to user.
schemaList	DatabaseSchema[]	Holds an array of DatabaseSchema objects.
userAccount	DatabaseAccountInfo	Holds account information used when connecting to database server.
queryGenerator	QueryGenerator	Instance of class QueryGenerator.
connection	Connection	A connection session with a specified database.
statement	Statement	Used for executing a static SQL statement and returning the results it produces.
resultSetList	ResultSetInfo[]	Array of ResultSetInfo objects.
databaseCore	DatabaseCore	Instance of DatabaseCore class. Used in singleton design pattern.
currentIndex	Integer	Holds the current index value for resultSetList.

Methods of the Class:

Method Name	Parameters	Return Type	Description
createFile	void	Integer	Creates a .php file with the connection info written in it.
getSchemaInfo	void	ArrayList	Returns the tables in a schema.
connectToMySQL	owner:*JFrame*	void	Connects to a MySQL server.

connectToOracle	owner:*JFrame*	void	Connects to an Oracle server.
closeConnection	void	Integer	Closes the database connection. Returns 1 if successful, 0 otherwise.
getErrorMessage	void	String	Returns errorMessage.
executeQuery	query:*String*	Integer	Executes given query. Returns 1 if successful, 0 otherwise.
updateOperation	tableName:*String,* rowInfo:*ArrayList,* columns:*ArrayList*	Integer	Calls ResultSetInfo's generateUpdateQuery method and gives the result to executeQuery.
insertOperation	tableName:*String*, rowInfo:*ArrayList*	Integer	Calls ResultSetInfo's generateInsertQuery method and gives the result to executeQuery.
deleteOperation	tableName:*String,* rowInfo:*ArrayList*	Integer	Calls ResultSetInfo's generateDeleteQuery method and gives the result to executeQuery.
getTableInfo	void	ResultSet	Returns the result set taken from DBMS.

DatabaseSchema Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
tables	ArrayList	Holds the tables in a database schema.
schemaName	String	Name of a schema.

ResultSetInfo Class:

Attributes of the Class:

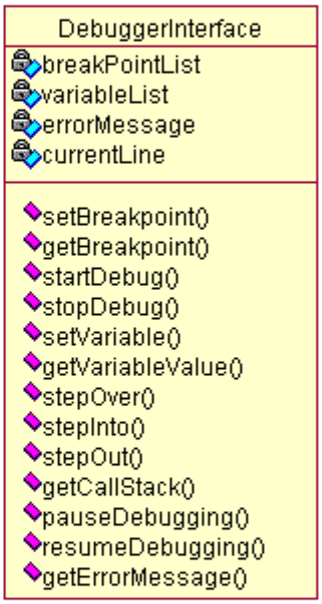
Attribute Name	Attribute Type	Description
tableName	String	Holds the name of a table.
resultSet	ResultSet	Holds the rows of a table.
columnNames	ArrayList	Holds the column names of a table.
query	String	Holds a query generated by an object of this class.

Methods of the Class:

Method Name	Parameters	Return Type	Description
generateUpdateQuery	tableName:*String,* rowInfo:*ArrayList,* columns:*ArrayList*	String	Generates an updates query according to the parameters it takes and returns it.
generateDeleteQuery	tableName:*String,*	String	Generates a delete query

	rowInfo:*ArrayList*		according to the parameters it takes and returns it.
generateInsertQuery	tableName:*String*, rowInfo:*ArrayList*	String	Generates an insert query according to the parameters it takes and returns it.
getQuery	void	String	Returns query attribute.

5.3.4 Debugger



Debugger Class:

Attributes of the Class:

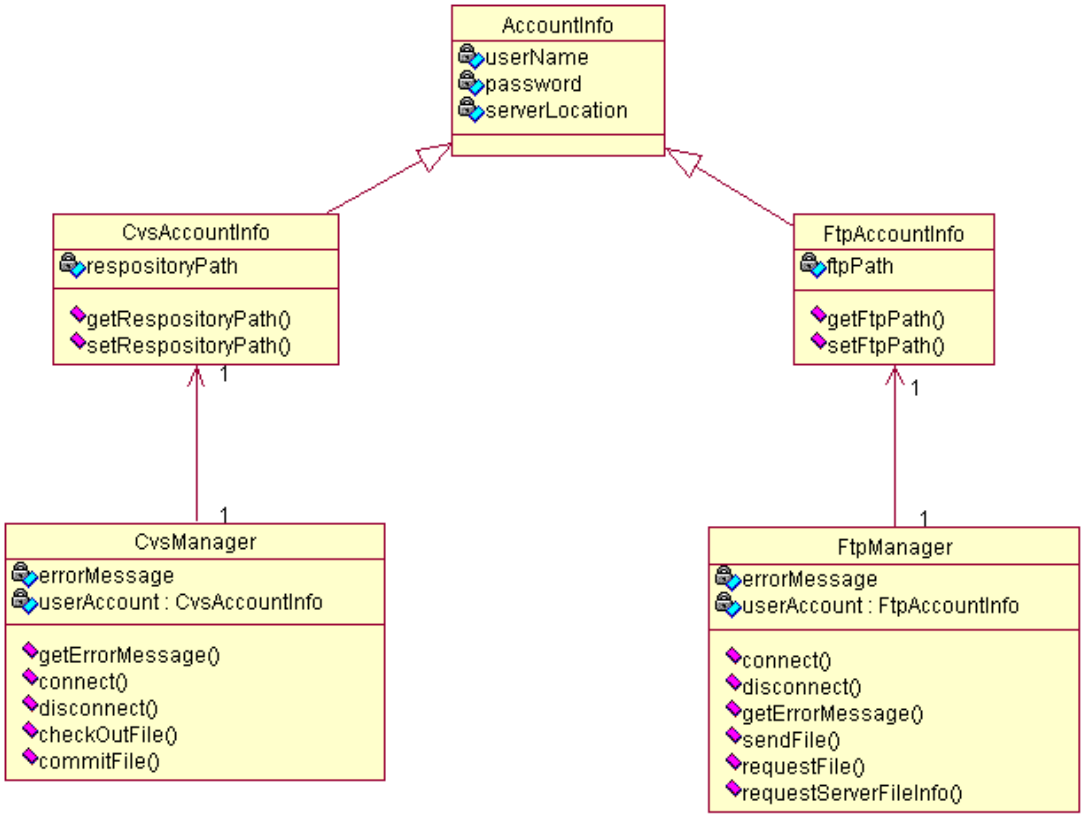
Attribute Name	Attribute Type	Description
breakPointList	ArrayList	It stores breakpoints locations in a list.
variableList	ArrayList	It stores variables in a list.
errorMessage	String	Holds the error message which is shown to

		user.
currentLine	Integer	Holds the currently executed line number.

Methods of the Class:

Method Name	Parameters	Return Type	Description
setBreakPoint	lineNo: *Integer*	Integer	Sets the breakpoint location.
getBreakPoint	id: *Integer*	Integer	Gets the breakpoint location.
startDebug	void	Boolean	It starts debugging. If it succeeds, return true, else return false.
stopDebug	void	Boolean	It stops debugging. If it succeeds, return true, else return false.
setVariable	name: *String* type: *String* scope: *String*	Boolean	Sets the variable that user wants to trace.
getVariableValue	name: *String* type: *String* scope: *String*	String	Gets current value of the variable.
stepOver	void	Boolean	Step over the breakpoint while debugging. If it succeeds, return true, else return false.
stepInto	void	Boolean	Step into the breakpoint while debugging. If it succeeds, return true, else return false.
stepOut	void	Boolean	Step out the breakpoint while debugging. If it succeeds, return true, else return false.
getCallStack	void	Boolean	Gets current call stack of the debugging program.If it succeeds, return true, else return false.
pauseDebugging	void	Boolean	It pauses debugging. If it succeeds, return true, else return false.
resumeDebugging	void	Boolean	It resumes debugging. If it succeeds, return true, else return false.
getErrorMessage	void	String	Returns errorMessage

5.3.5 CVS – FTP Connections



CVSAccountInfo Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
repositoryPath	String	Stores the path of the repository of the CVS Server.

Methods of the Class:

Method Name	Parameters	Return Type	Description
getRepositoryPath	void	String	Sets the repository path.
setRepositoryPath	path: String	void	Gets the repository path.

CVSManager Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
userAccount	CVSAccountInfo	Stores the user account information to connect to CVS Server.
errorMessage	String	Stores the error message that is shown to user.

Methods of the Class:

Method Name	Parameters	Return Type	Description
getErrorMessage	void	String	Returns the error message.
connect	void	Boolean	Connects to CVS server. Returns true if successful, false otherwise.
disconnect	void	Boolean	Disconnects from CVS server. Returns true if successful, false otherwise.
checkOutFile	path: String	File	Requests the specified file from CVS Server.
commitFile	name: File	Boolean	Commits a specified file to CVS Server. Returns true if successful, false otherwise.

FTPAccountInfo Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
ftpPath	String	Stores the path of the FTP Server.

Methods of the Class:

Method Name	Parameters	Return Type	Description
getFtpPath	void	String	Sets the ftp path.
setFtpPath	path: String	void	Gets the ftp path.

FTPManager Class:

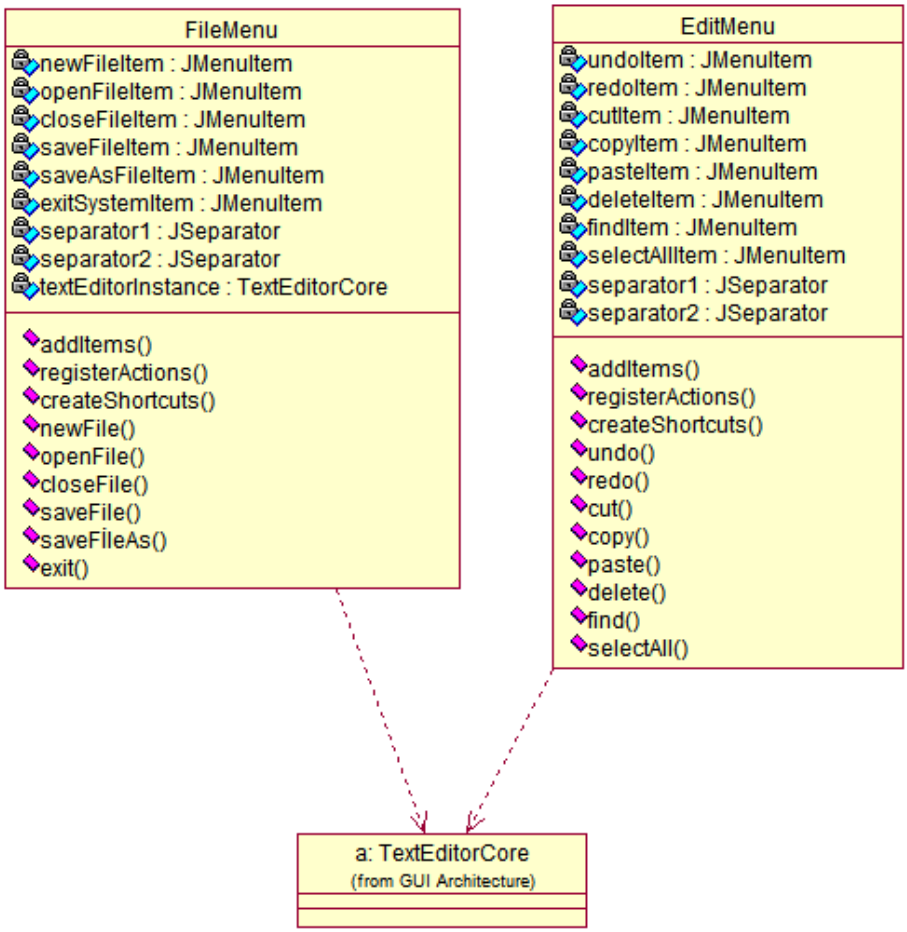
Attributes of the Class:

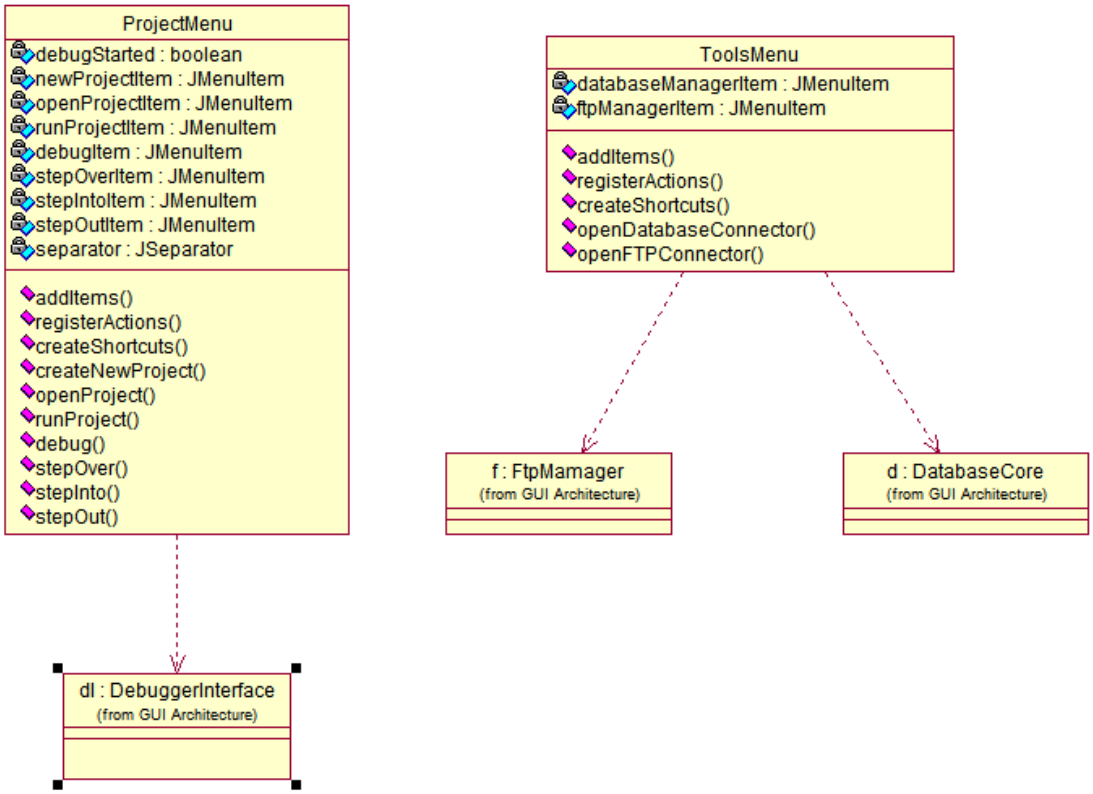
Attribute Name	Attribute Type	Description
userAccount	FTPAccountInfo	Stores the user account information to connect to an FTP Server.
errorMessage	String	Stores the error message that is shown to user.

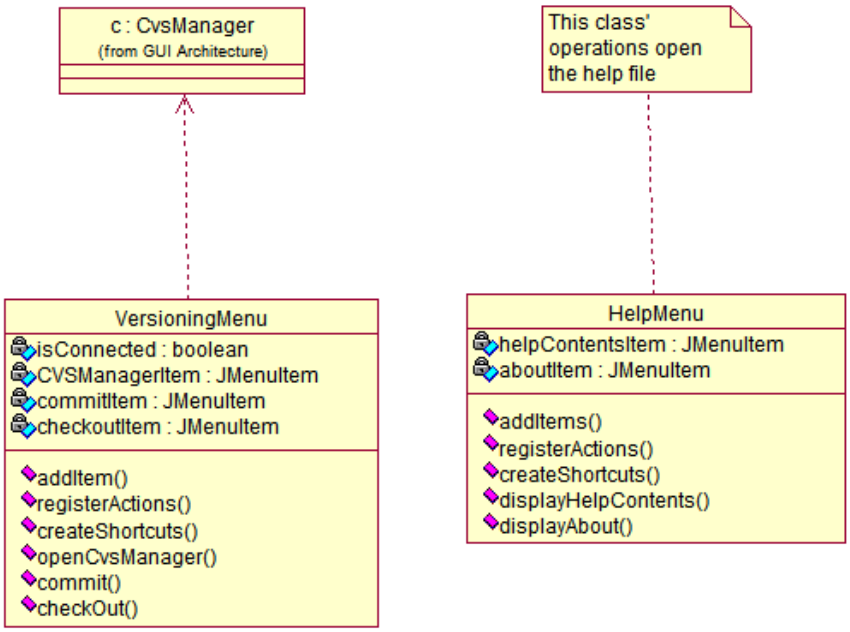
Methods of the Class:

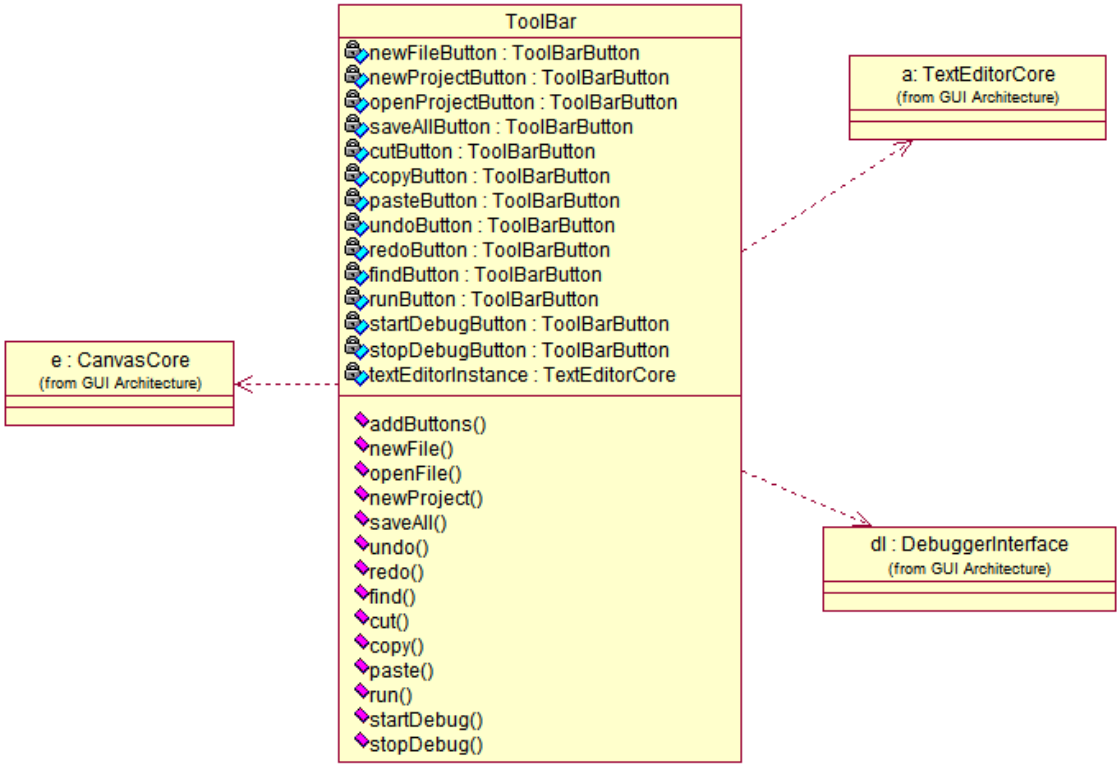
Method Name	Parameters	Return Type	Description
getErrorMessage	void	String	Returns the error message.
connect	void	Boolean	Connects to FTP server. Returns true if successful, false otherwise.
disconnect	void	Boolean	Disconnects from FTP server. Returns true if successful, false otherwise.
requestFile	path: String	File	Requests a file from FTP Server.
sendFile	name: File	Boolean	Sends a file to via FTP Server. Returns true if successful, false otherwise.
requestServerFileInfo	void	TreeModel	Gets the file information from FTP Server.

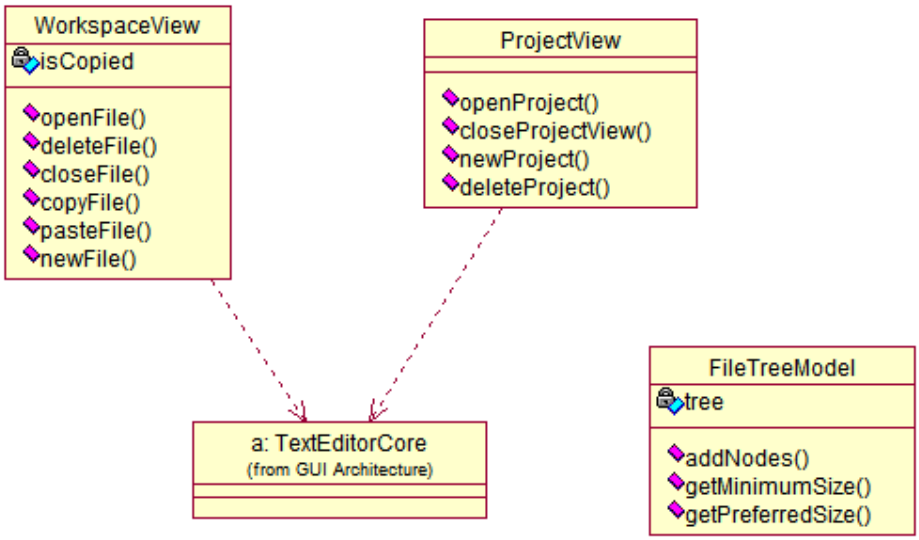
5.3.6 GUI

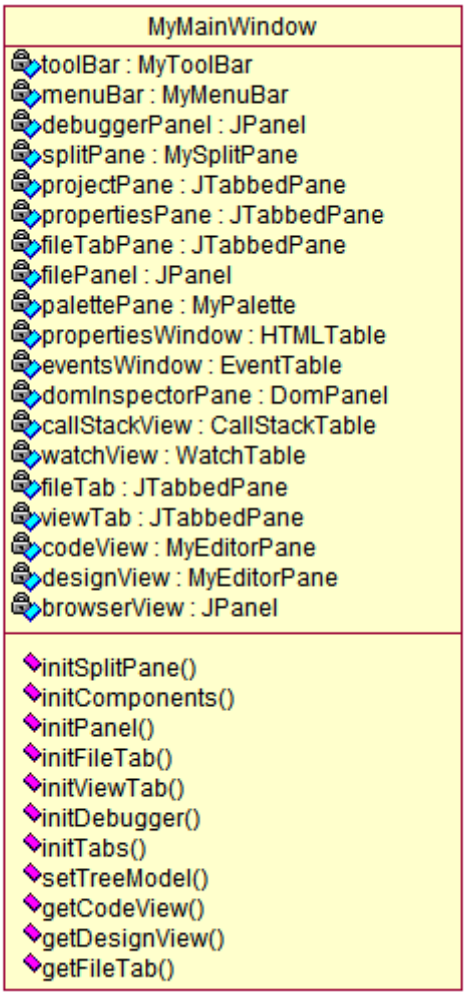

FileMenu
newFileItem : JMenuItem
openFileItem : JMenuItem
closeFileItem : JMenuItem
saveFileItem : JMenuItem
saveAsFileItem : JMenuItem
exitSystemItem : JMenuItem
separator1 : JSeparator
separator2 : JSeparator
textEditorInstance : TextEditorCore
addItems()
registerActions()
createShortcuts()
newFile()
openFile()
closeFile()
saveFile()
saveFileAs()
exit()
EditMenu
undoItem : JMenuItem
redoItem : JMenuItem
cutItem : JMenuItem
copyItem : JMenuItem
pasteItem : JMenuItem
deleteItem : JMenuItem
findItem : JMenuItem
selectAllItem : JMenuItem
separator1 : JSeparator
separator2 : JSeparator
addItems()
registerActions()
createShortcuts()
undo()
redo()
cut()
copy()
paste()
delete()
find()
selectAll()
a: TextEditorCore
(from GUI Architecture)

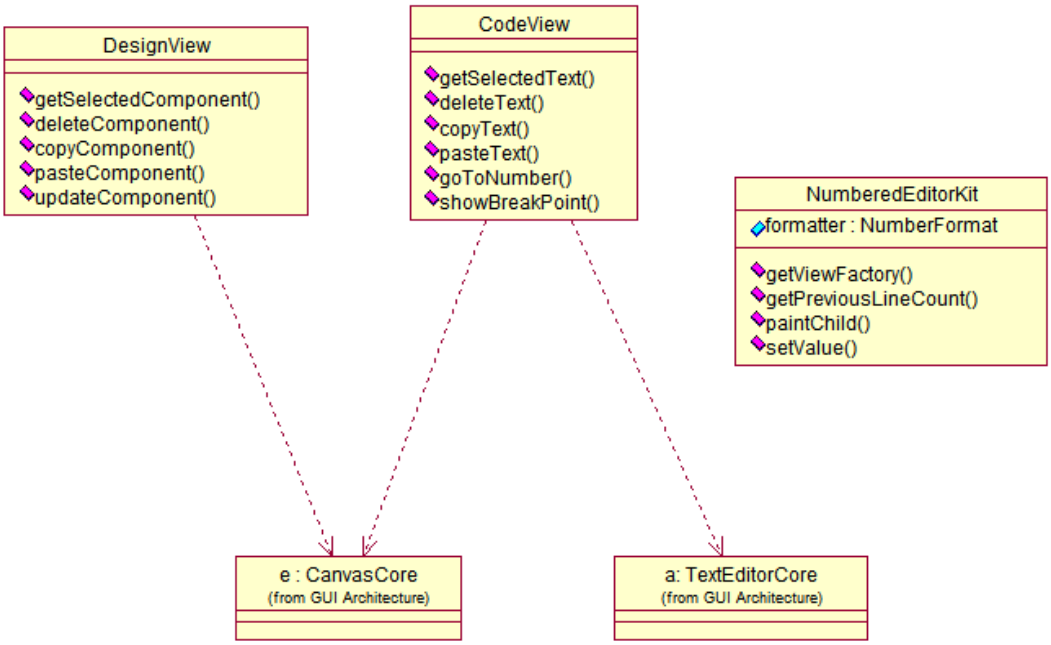

ProjectMenu
debugStarted : boolean
newProjectItem : JMenuItem
openProjectItem : JMenuItem
runProjectItem : JMenuItem
debugItem : JMenuItem
stepOverItem : JMenuItem
stepIntoItem : JMenuItem
stepOutItem : JMenuItem
separator : JSeparator
addItems()
registerActions()
createShortcuts()
createNewProject()
openProject()
runProject()
debug()
stepOver()
stepInto()
stepOut()
ToolsMenu
databaseManagerItem : JMenuItem
ftpManagerItem : JMenuItem
addItems()
registerActions()
createShortcuts()
openDatabaseConnector()
openFTPConnector()
f : FtpMamager
(from GUI Architecture)
d : DatabaseCore
(from GUI Architecture)
dl : DebuggerInterface
(from GUI Architecture)

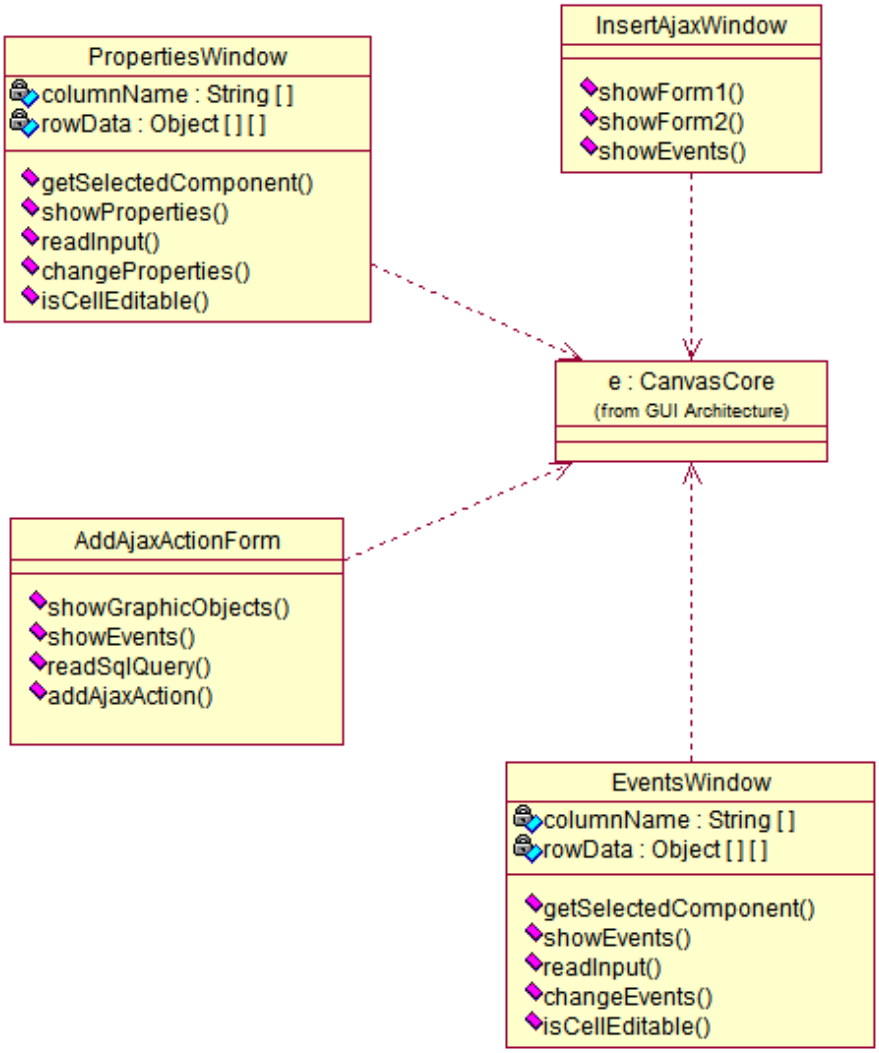

c : CvsManager
(from GUI Architecture)
This class' operations open the help file
VersioningMenu
isConnected : boolean
CVSManagerItem : JMenuItem
commitItem : JMenuItem
checkoutItem : JMenuItem
addItem()
registerActions()
createShortcuts()
openCvsManager()
commit()
checkOut()
HelpMenu
helpContentsItem : JMenuItem
aboutItem : JMenuItem
addItems()
registerActions()
createShortcuts()
displayHelpContents()
displayAbout()

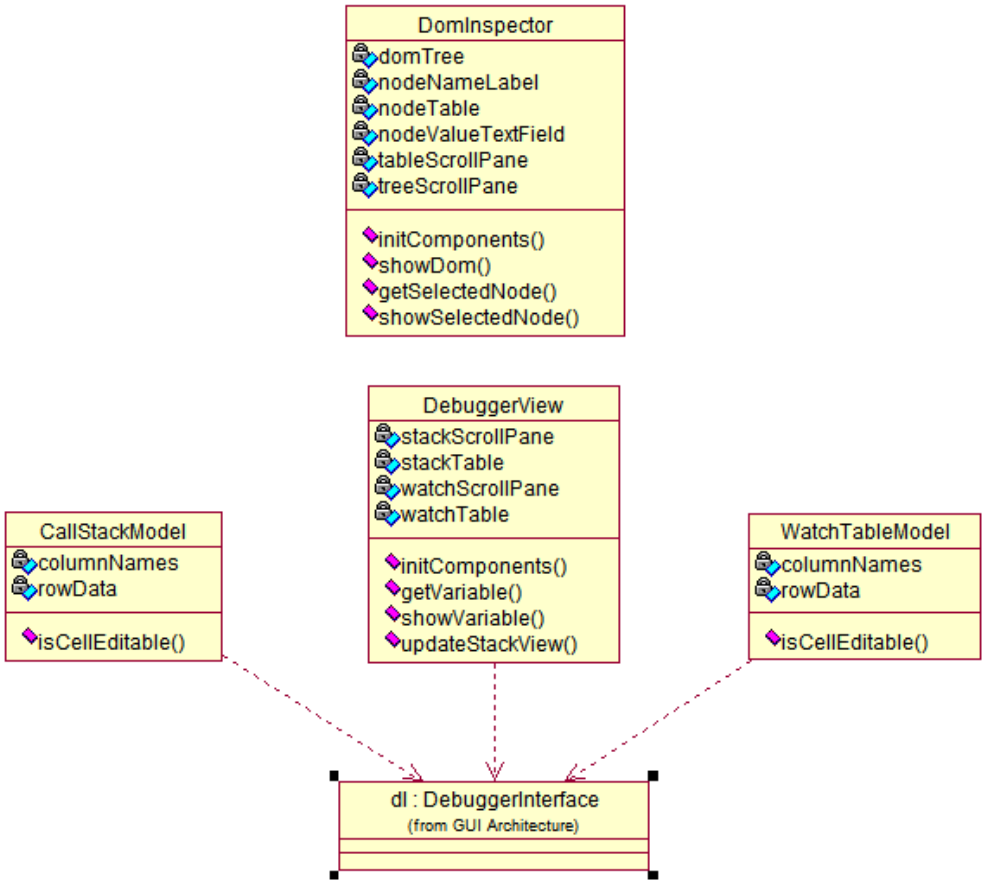

ToolBar
newFileButton : ToolBarButton
newProjectButton : ToolBarButton
openProjectButton : ToolBarButton
saveAllButton : ToolBarButton
cutButton : ToolBarButton
copyButton : ToolBarButton
pasteButton : ToolBarButton
undoButton : ToolBarButton
redoButton : ToolBarButton
findButton : ToolBarButton
runButton : ToolBarButton
startDebugButton : ToolBarButton
stopDebugButton : ToolBarButton
textEditorInstance : TextEditorCore
addButtons()
newFile()
openFile()
newProject()
saveAll()
undo()
redo()
find()
cut()
copy()
paste()
run()
startDebug()
stopDebug()
a: TextEditorCore
(from GUI Architecture)
e : CanvasCore
(from GUI Architecture)
dl : DebuggerInterface
(from GUI Architecture)

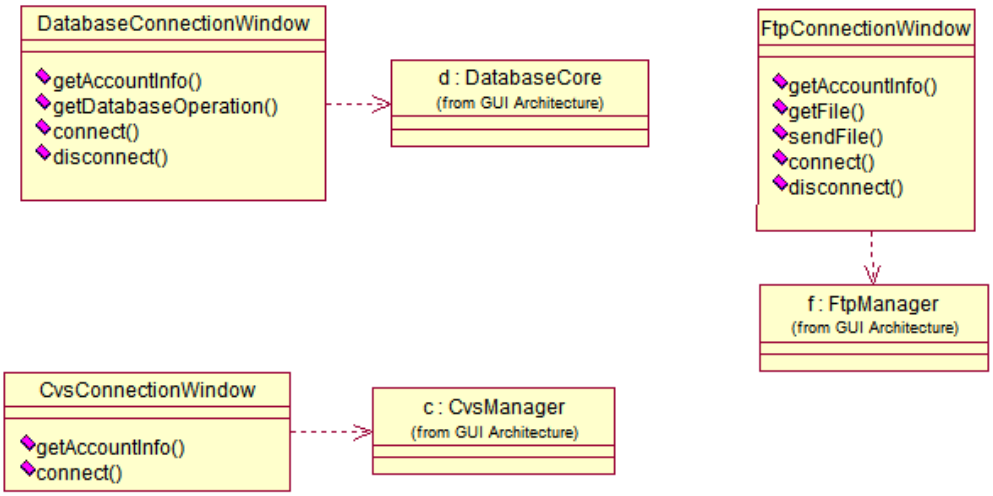

WorkspaceView
isCopied
openFile()
deleteFile()
closeFile()
copyFile()
pasteFile()
newFile()
ProjectView
openProject()
closeProjectView()
newProject()
deleteProject()
a: TextEditorCore
(from GUI Architecture)
FileTreeModel
tree
addNodes()
getMinimumSize()
getPreferredSize()


MyMainWindow
toolBar : MyToolBar
menuBar : MyMenuBar
debuggerPanel : JPanel
splitPane : MySplitPane
projectPane : JTabbedPane
propertiesPane : JTabbedPane
fileTabPane : JTabbedPane
filePanel : JPanel
palettePane : MyPalette
propertiesWindow : HTMLTable
eventsWindow : EventTable
domInspectorPane : DomPanel
callStackView : CallStackTable
watchView : WatchTable
fileTab : JTabbedPane
viewTab : JTabbedPane
codeView : MyEditorPane
designView : MyEditorPane
browserView : JPanel
initSplitPane()
initComponents()
initPanel()
initFileTab()
initViewTab()
initDebugger()
initTabs()
setTreeModel()
getCodeView()
getDesignView()
getFileTab()


DesignView
getSelectedComponent()
deleteComponent()
copyComponent()
pasteComponent()
updateComponent()
CodeView
getSelectedText()
deleteText()
copyText()
pasteText()
goToNumber()
showBreakPoint()
NumberedEditorKit
formatter : NumberFormat
getViewFactory()
getPreviousLineCount()
paintChild()
setValue()
e : CanvasCore
(from GUI Architecture)
a: TextEditorCore
(from GUI Architecture)


PropertiesWindow
columnName : String [ ]
rowData : Object [ ] [ ]
getSelectedComponent()
showProperties()
readInput()
changeProperties()
isCellEditable()
InsertAjaxWindow
showForm1()
showForm2()
showEvents()
e : CanvasCore
(from GUI Architecture)
AddAjaxActionForm
showGraphicObjects()
showEvents()
readSqlQuery()
addAjaxAction()
EventsWindow
columnName : String [ ]
rowData : Object [ ] [ ]
getSelectedComponent()
showEvents()
readInput()
changeEvents()
isCellEditable()


DomInspector
domTree
nodeNameLabel
nodeTable
nodeValueTextField
tableScrollPane
treeScrollPane
initComponents()
showDom()
getSelectedNode()
showSelectedNode()
DebuggerView
stackScrollPane
stackTable
watchScrollPane
watchTable
initComponents()
getVariable()
showVariable()
updateStackView()
CallStackModel
columnNames
rowData
isCellEditable()
WatchTableModel
columnNames
rowData
isCellEditable()
dI : DebuggerInterface
(from GUI Architecture)


DatabaseConnectionWindow
getAccountInfo()
getDatabaseOperation()
connect()
disconnect()
d : DatabaseCore
(from GUI Architecture)
FtpConnectionWindow
getAccountInfo()
getFile()
sendFile()
connect()
disconnect()
f : FtpManager
(from GUI Architecture)
CvsConnectionWindow
getAccountInfo()
connect()
c : CvsManager
(from GUI Architecture)

Class Diagram Dictionary of GUI

FileMenu

Attributes of the Class

Attribute Name	Attribute Type	Description
newFileItem	JMenuItem	New File choice of File Menu
openFileItem	JMenuItem	Open File choice of File Menu
closeFileItem	JMenuItem	Close File choice of File Menu
saveFileItem	JMenuItem	Save File choice of File Menu
saveFileAsItem	JMenuItem	Save File As choice of File Menu
exitSystemItem	JMenuItem	Exit choice of File Menu
separator1	JSeparator	Separator of File Menu
separator2	JSeparator	Separator of File Menu
textEditorInstance	TextEditorCore	Instance of Text Editor

Methods of the Class

Method Name	Parameters	Return Type	Description
addItems()	-	void	Adds items to File Menu
registerActions()	-	void	Bind actions to menu items
createShortcuts()	-	void	Create shortcuts of File Menu
newFile()	-	File	Creates a new file and invokes TextEditorCore
openFile()	-	File	Invokes TextEditorCore to open a file
closeFile()	-	void	Invokes TextEditorCore to close file
saveFile()	-	void	Invokes TextEditorCore to save file
saveFileAs()	-	File	Invokes TextEditorCore to save file as another file
exit()	-	void	Invoke exit function of system

EditMenu

Attributes of the Class

Attribute Name	Attribute Type	Description
undoItem	JMenuItem	Undo choice of Edit Menu
redoItem	JMenuItem	Redo choice of Edit Menu
cutItem	JMenuItem	Cut choice of Edit Menu
copyItem	JMenuItem	Copy choice of Edit Menu
pasteItem	JMenuItem	Paste choice of Edit Menu
deleteItem	JMenuItem	Delete choice of Edit Menu
selectAllItem	JMenuItem	Select All choice of Edit Menu
findItem	JMenuItem	Find choice of Edit Menu
separator1	JSeparator	Separator of Edit Menu
separator2	JSeparator	Separator of Edit Menu

Methods of the Class

Method Name	Parameters	Return Type	Description
addItems()	-	void	Adds items to Edit Menu
registerActions()	-	void	Bind actions to menu items
createShortcuts()	-	void	Create shortcuts of Edit Menu
undo()	-	void	Invokes Text Editor's undo function
redo()	-	void	Invokes Text Editor's redo function
cut()	-	void	Invokes Text Editor's cut function
copy()	-	void	Invokes Text Editor's copy function
paste()	-	void	Invokes Text Editor's paste function
delete()		void	Invokes Text Editor's delete function
find()	-	void	Invokes Text Editor's find function
selectAll()	-	void	Invokes Text Editor's selectAll funct

ProjectMenu

Attributes of the Class

Attribute Name	Attribute Type	Description
newProjectItem	JMenuItem	New Project choice of Project Menu
openProjectItem	JMenuItem	Open Project choice of Project Menu
runProjectItem	JMenuItem	Run Project choice of Project Menu
debugItem	JMenuItem	Debug choice of Project Menu
stepOverItem	JMenuItem	Step Over choice of Project Menu
stepIntoItem	JMenuItem	Step Into choice of Project Menu
stepOutItem	JMenuItem	Step Out choice of Project Menu
separator	JSeparator	Separator of Project Menu
debugStarted()	int	Stores debugging information.

Methods of the Class

Method Name	Parameters	Return Type	Description
addItems()	-	void	Adds items to Project Menu
registerActions()	-	void	Bind actions to menu items
createShortcuts()	-	void	Create shortcuts of Project Menu
createNewProject()	-	Project	Invokes related function to create project
openProject()	-	Project	Invokes related function to open a project
runProject()	-	void	Invokes related function to run project
startDebug()	-	void	Invokes DebuggerInterface to debug
stepOver()	-	void	Invokes DebuggerInterface to step over
stepInto()		void	Invokes DebuggerInterface to step into
stepOut()	-	void	Invokes DebuggerInterface to stepout

ToolsMenu

Attributes of the Class

Attribute Name	Attribute Type	Description
databaseManagerItem	JMenuItem	Database Editor choice of Tools Menu
ftpManagerItem	JMenuItem	FTP Connector choice of Tools Menu

Methods of the Class

Method Name	Parameters	Return Type	Description
addItems()	-	void	Adds items to Tools Menu
registerActions()	-	void	Bind actions to menu items
createShortcuts()	-	void	Create shortcuts of Tools Menu
openDatabaseConnector()	-	void	Shows Database connection console and invoke DatabaseCore
openFTPConnector()	-	void	Shows FTP connection console and invoke FTPManager

HelpMenu

Attributes of the Class

Attribute Name	Attribute Type	Description
helpContentsItem	JMenuItem	Help Contents choice of Help Menu
aboutItem	JMenuItem	About choice of Help Menu

Methods of the Class

Method Name	Parameters	Return Type	Description
addItems()	-	void	Adds items to Help Menu
registerActions()	-	void	Bind actions to menu items
createShortcuts()	-	void	Create shortcuts of Help Menu
displayHelpContents()	-	void	Shows help contents
displayAbout()	-	void	Shows information about IDE

VersioningMenu

Attributes of the Class

Attribute Name	Attribute Type	Description
CVSManagerItem	JMenuItem	CVS Manager choice of Versioning Menu
commitItem	JMenuItem	Commit choice of Versioning Menu
checkoutItem	JMenuItem	Checkout choice of Versioning Menu
isConnected()	boolean	Stores CVS connection information.

Methods of the Class

Method Name	Parameters	Return Type	Description
addItems()	-	void	Adds items to Versioning Menu
registerActions()	-	void	Bind actions to menu items
createShortcuts()	-	void	Create shortcuts of Versioning Menu
openCVSManager()	-	void	Invokes CVS Manager to open CVS window
commit()	-	void	Invokes CVS Manager to commit file
checkout()	-	void	Invokes CVS Manager to checkout file

ToolBar

Attributes of the Class

Attribute Name	Attribute Type	Description
newFileButton	ToolBarButton	New File button of Tool Bar
newProjectButton	ToolBarButton	New Project button of Tool Bar
openProjectButton	ToolBarButton	Open Project button of Tool Bar
saveAllButton	ToolBarButton	Save All button of Tool Bar
cutButton	ToolBarButton	Cut button of Tool Bar
copyButton	ToolBarButton	Copy button of Tool Bar
pasteButton	ToolBarButton	Paste button of Tool Bar
undoButton	ToolBarButton	Undo button of Tool Bar
redoButton	ToolBarButton	Redo button of Tool Bar
findButton	ToolBarButton	Find button of Tool Bar
runButton	ToolBarButton	Run button of Tool Bar
startDebugButton	ToolBarButton	Start Debug button of Tool Bar
stopDebugButton	ToolBarButton	Stop Debug button of Tool Bar
textEditorInstance	TextEditorCore	Instance of Text Editor

Methods of the Class

Method Name	Parameters	Return Type	Description
addButtons()	-	void	Add items to Tool Bar and bind actions.
newFile()	-	void	Creates a new file and invokes TextEditorCore.
newProject()	-	void	Creates a new project and invokes related function.
openFile()	-	void	Invokes TextEditorCore to open file.
saveAll()	-	void	Invokes TextEditorCore to save files
undo()	-	void	Invokes Text Editor’s undo function
redo()	-	void	Invokes Text Editor’s redo function
find()	-	void	Invokes Text Editor’s find function.
cut()	-	void	Invokes Text Editor’s cut function
copy()	-	void	Invokes Text Editor’s copy function
paste()	-	void	Invokes Text Editor’s paste function

run()	-	void	Invokes related function to run project.
startDebugging()	-	void	Invokes DebuggerInterface to start debugging.
stopDebugging()	-	void	Invokes DebuggerInterface to stop debugging.

MyMainWindow

Attributes of the Class

Attribute Name	Attribute Type	Description
toolBar	MyToolBar	Tool Bar part of GUI
menuBar	MyMenuBar	Menu Bar part of GUI
debuggerPanel	JPanel	Debugger part of GUI
splitPane	MySplitPane	Split Pane part of GUI
splitPaneLeft	MySplitPane	Split Pane part of GUI
splitPaneRight	MySplitPane	Split Pane part of GUI
middle	MySplitPane	Split Pane part of GUI
right	MySplitPane	Split Pane part of GUI
debuggerSplitPane	MySplitPane	Split Pane part of GUI
projectPane	JTabbedPane	Project part of GUI
propertiesPane	JTabbedPane	Properties part of GUI
fileTabPane	JTabbedPane	File view of GUI
filePanel	JPanel	File Panel of GUI
palettePane	MyPalette	Palette part of GUI
propertiesWindow	HTMLTable	HTML table of Properties part
eventsWindow	EventTable	Event part of GUI
domInspectorPane	DomPanel	Dom Inspector part of GUI
callStackView	CallStackTable	Call Stack View part of GUI
watchView	WatchTable	Watch View part of GUI
fileTab	JTabbedPane	Tabbed File View of GUI
viewTab	JTabbedPane	Code-Design-Browser Views
codeView	MyEditorPane	Code View of GUI
designView	MyEditorPane	Design View of GUI
browserView	JPanel	Browser View of GUI

Methods of the Class

Method Name	Parameters	Return Type	Description
initSplitPane()	-	void	Initializes main window and split it using split panes.
initComponents()	-	void	Initializes and place all GUI components in main window.
initPanel()	-	void	Initializes File and View tabs of Code-Design-Browser part.
initFileTab()	-	void	Initializes File Tab of Code-Design-Browser part

initViewTab()	-	void	Initializes View Tab of Code-Design-Browser part
initDebugger()	-	void	Initializes debugger view of GUI
initTabs()	-	void	Initializes tabs of Project-Workspace and Properties-Event
setTreeModel()	URL : string	JScrollPane	Provides a tree structure for Project and Workspace view.
getCodeView()	-	MyEditorPane	Places Code View for Text Editor
getDesignView()	-	MyEditorPane	Places Design View
getFileTab()	-	JTabbedPane	Places File Tab of Code-Design-Browser part

DesignView

Methods of the Class

Method Name	Parameters	Return Type	Description
getSelectedComponent()	-	void	send selected component to CanvasCore.
deleteComponent()	-	void	Invokes CanvasCore to delete a component.
copyComponent()	-	void	Invokes CanvasCore to copy a component.
pasteComponent()	-	void	Invokes CanvasCore to paste a component.
updateComponent()	-	void	Invokes CanvasCore to update a component.

CodeView

Methods of the Class

Method Name	Parameters	Return Type	Description
getSelectedText()	-	void	send selected text to TextEditorCore.
deleteText ()	-	void	Invokes TextEditorCore to delete text.
copyText ()	-	void	Invokes TextEditorCore to copy text.
pasteText ()	-	void	Invokes TextEditorCore to paste text.
goToNumber()	-	void	Invokes TextEditorCore to go selected line.
showBreakPoint()	-	void	shows breakpoints of Debugger

NumberedEditorKit

Attributes of the Class

Attribute Name	Attribute Type	Description
formatter	NumberFormat	Describes numbers of line of Code View

Methods of the Class

Method Name	Parameters	Return Type	Description
getViewFactory()	-	ViewFactory	Creates a new NumberedViewFactory instance
getPreviousLineCount()	-	int	Counts previous lines
paintChild()	g : Graphics r :Rectangle n : int	void	Writes the line number of Code View
setValue()	value:Object	void	Set formatter's value

PropertiesWindow

Attributes of the Class

Attribute Name	Attribute Type	Description
columnName	String [ ]	Describes column name of Properties Table
rowData	Object [ ] [ ]	Describes row data of Properties Table

Methods of the Class

Method Name	Parameters	Return Type	Description
getSelectedComponent()	-	void	send selected component to CanvasCore.
showProperties()	-	void	show properties of component.
readInput()	-	void	gets input from user.
changeProperties()	-	void	update properties of selected component.
isCellEditable	row : int column : int	Boolean	Returns if cell is editable or not

EventsWindow

Attributes of the Class

Attribute Name	Attribute Type	Description
columnName	String [ ]	Describes column name of Events Table
rowData	Object [ ] [ ]	Describes row data of Events Table

Methods of the Class

Method Name	Parameters	Return Type	Description
getSelectedComponent()	-	void	Get selected component CanvasCore.
showEvents()	-	void	show events of component.
readInput()	-	void	gets input from user.
changeEvents()	-	void	update events of selected component.
isCellEditable	row : int column : int	Boolean	Returns if cell is editable or not

InsertAjaxWindow

Methods of the Class

Method Name	Parameters	Return Type	Description
showForm1()	-	void	shows the form to insert object1.
showForm2()	-	void	shows the form to insert object2
showEvents()	-	void	shows events of inserted objects.

AddAjaxActionForm

Methods of the Class

Method Name	Parameters	Return Type	Description
showGraphicObjects()	-	void	shows the list of objects.
showEvents()	-	void	shows events of inserted objects.
readSqlQuery()	-	void	gets the SQL query input of user.
addAjaxAction()	-	void	invokes system to add new AJAX object.

ProjectView

Methods of the Class

Method Name	Parameters	Return Type	Description
openProject ()	-	void	Invokes related function to open a project.
closeProject ()	-	void	Invokes related function to close a project.
newProject()	-	void	Creates a new project and invokes related function.
deleteProject ()	-	void	Invokes related function to delete project.

WorkspaceView

Attributes of the Class

Attribute Name	Attribute Type	Description
isCopied	boolean	Describes if selected project is copied to workspace or not

Methods of the Class

Method Name	Parameters	Return Type	Description
openFile()	-	void	Invokes TextEditorCore to open a file.
deleteFile()	-	void	Invokes TextEditorCore to delete a file.
closeFile()	-	void	Invokes TextEditorCore to close a file.
copyFile()	-	void	Invokes TextEditorCore to copy a file.
pasteFile()	-	void	Invokes TextEditorCore to paste a file.
newFile()	-	void	Creates a new file and invokes TextEditorCore.

FileTreeModel

Attributes of the Class

Attribute Name	Attribute Type	Description
tree	JTree	tree structure of Files in WorkSpace

Methods of the Class

Method Name	Parameters	Return Type	Description
addNodes()	currentTop : DefaultMutableTreeNode dir : File	void	Adds new files to file tree
getMinimumSize()	-	Dimension	Returns min size of tree
getPreferredSize()	-	Dimension	Returns preferred size of tree

DebuggerView

Attributes of the Class

Attribute Name	Attribute Type	Description
stackScrollPane	JScrollPane	Scroll pane of call stack view
stackTable	JTable	Call stack view table
watchScrollPane	JScrollPane	Scroll pane of watch view
watchTable	JTable	Watch view table

Methods of the Class

Method Name	Parameters	Return Type	Description
initComponents()	-	void	Initialize Debugger components
getVariable()	-	void	Gets the entered varible information
showVariable()	-	void	Shows information of variable.
updateStackView()	-	void	Invokes DebuggerInterface to update program stack.

CallStackModel

Attributes of the Class

Attribute Name	Attribute Type	Description
columnName	String [ ]	Describes column name of CallStack Table
rowData	Object [ ] [ ]	Describes row data of CallStack Table

Methods of the Class

Method Name	Parameters	Return Type	Description
isCellEditable	row : int column : int	Boolean	Returns if cell is editable or not

WatchTableModel

Attributes of the Class

Attribute Name	Attribute Type	Description
columnName	String [ ]	Describes column name of Watch Table
rowData	Object [ ] [ ]	Describes row data of Watch Table

Methods of the Class

Method Name	Parameters	Return Type	Description
isCellEditable	row : int column : int	Boolean	Returns if cell is editable or not

DomInspector

Attributes of the Class

Attribute Name	Attribute Type	Description
DomTree	JTree	Tree structure of DOM inspector
nodeNameLabel	JLabel	Node Name label of DOM inspector
nodeTable	JTable	Table of nodes in DOM inspector
nodeValueTextField	JTextField	Text field that shows value
tableScrollPane	JScrollPane	Scroll pane for table of DOM inspector
treeScrollPane	JScrollPane	Scroll pane for tree of DOM inspector

Methods of the Class

Method Name	Parameters	Return Type	Description
initComponents ()	-	void	Initialize DOM inspector components
showDom()	tree : DefaultMutableTreeNode	void	Bind actions to menu items
getSelectedNode()	-	void	Gets the selected node information
showSelectedNode()	-	void	Shows information of selected node.

CvsConnectionWindow

Methods of the Class

Method Name	Parameters	Return Type	Description
getAccountInfo()	-	void	Gets input of connection information from user.
connect()	-	void	Invokes CVSManager to connect.

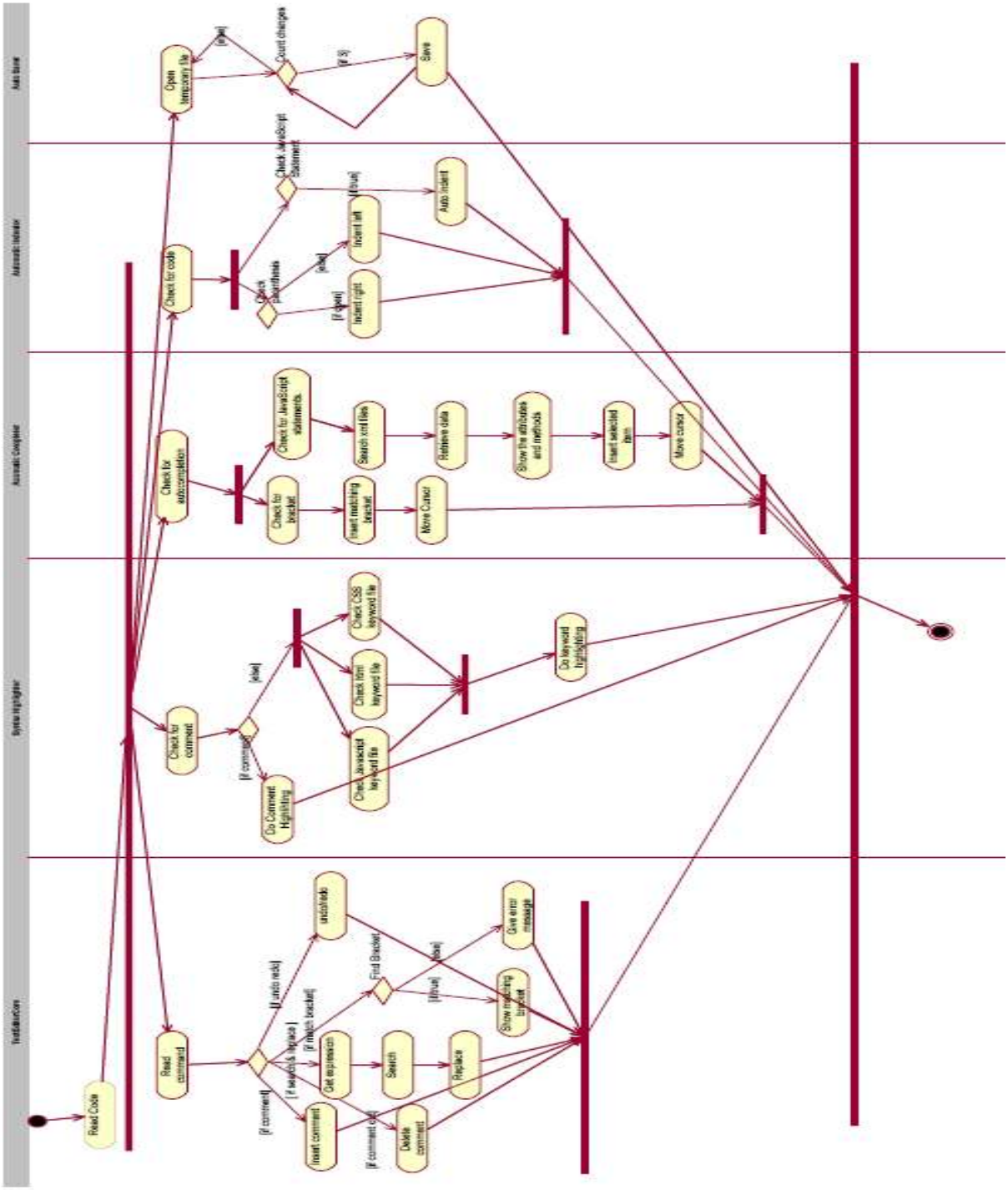
FtpConnectionWindow

Methods of the Class

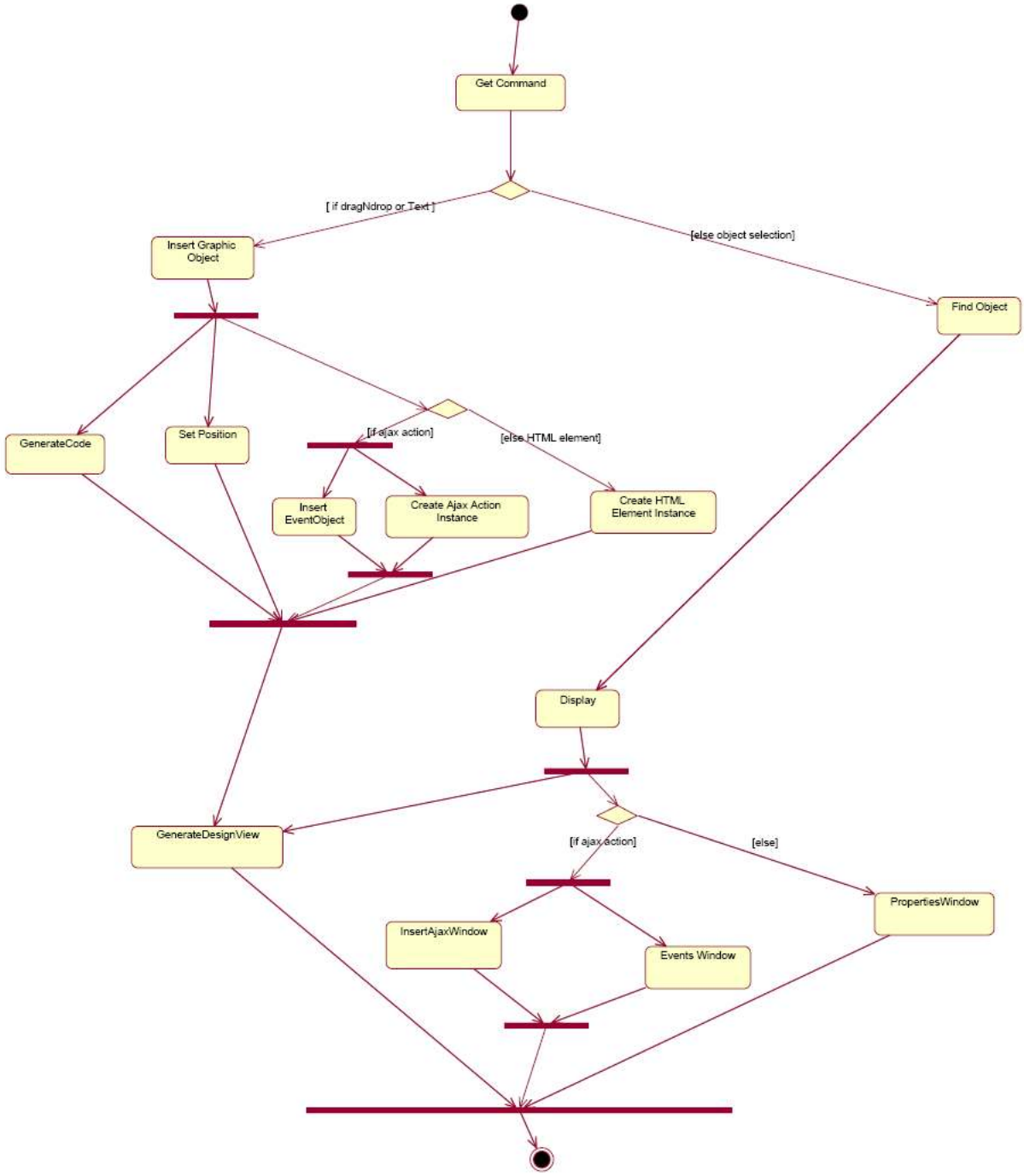
Method Name	Parameters	Return Type	Description
getAccountInfo()	-	void	gets input of connection information from user.
getFile()	-	void	gets file information from user and invokes FtpManager to get file.
sendFile()	-	void	gets file information from user and invokes FtpManager to send file..
connect()	-	void	Invokes FtpManager to connect
disconnect()	-	void	Invokes FtpManager to disconnect.

5.4 Activity Diagrams

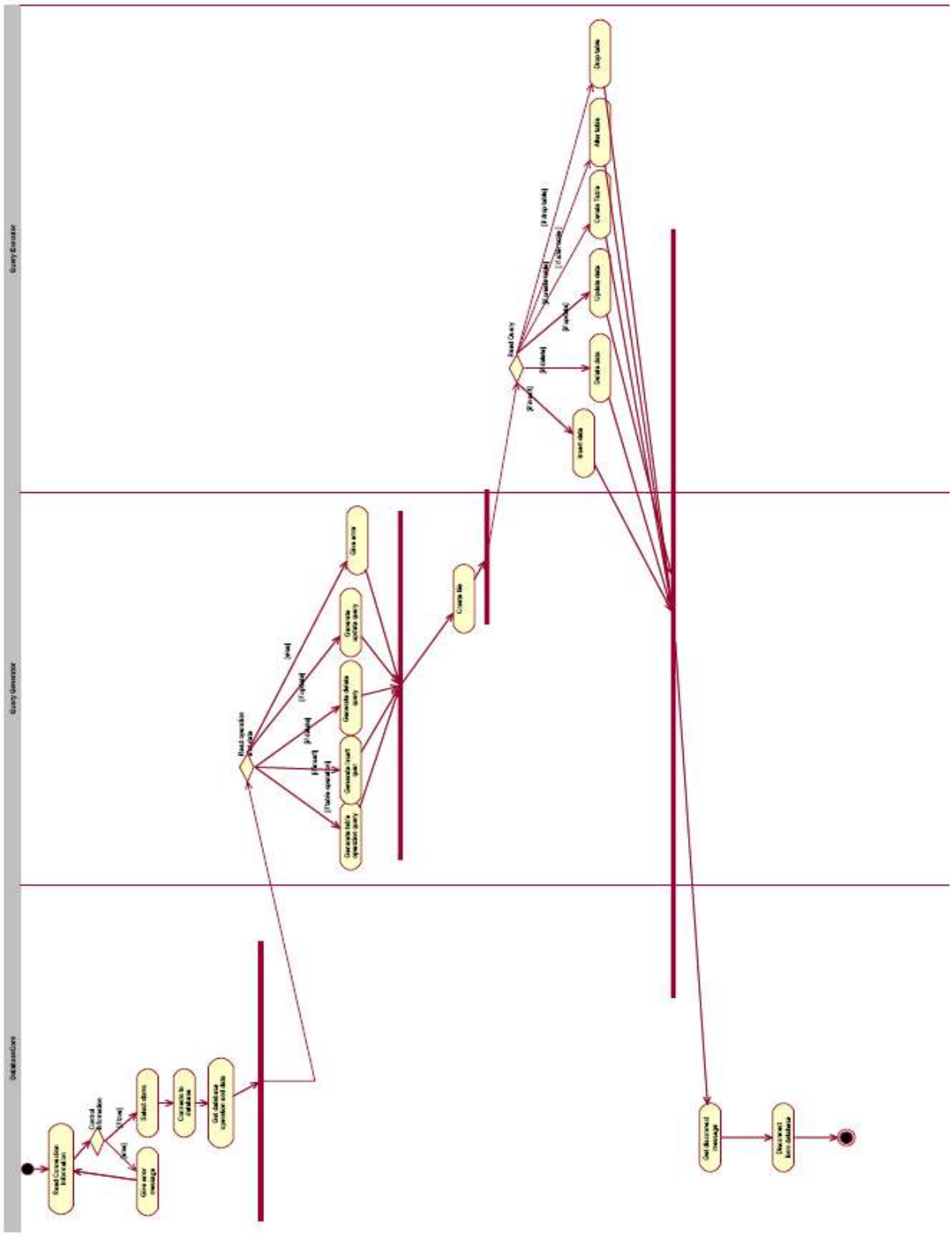
5.4.1 Text Editor



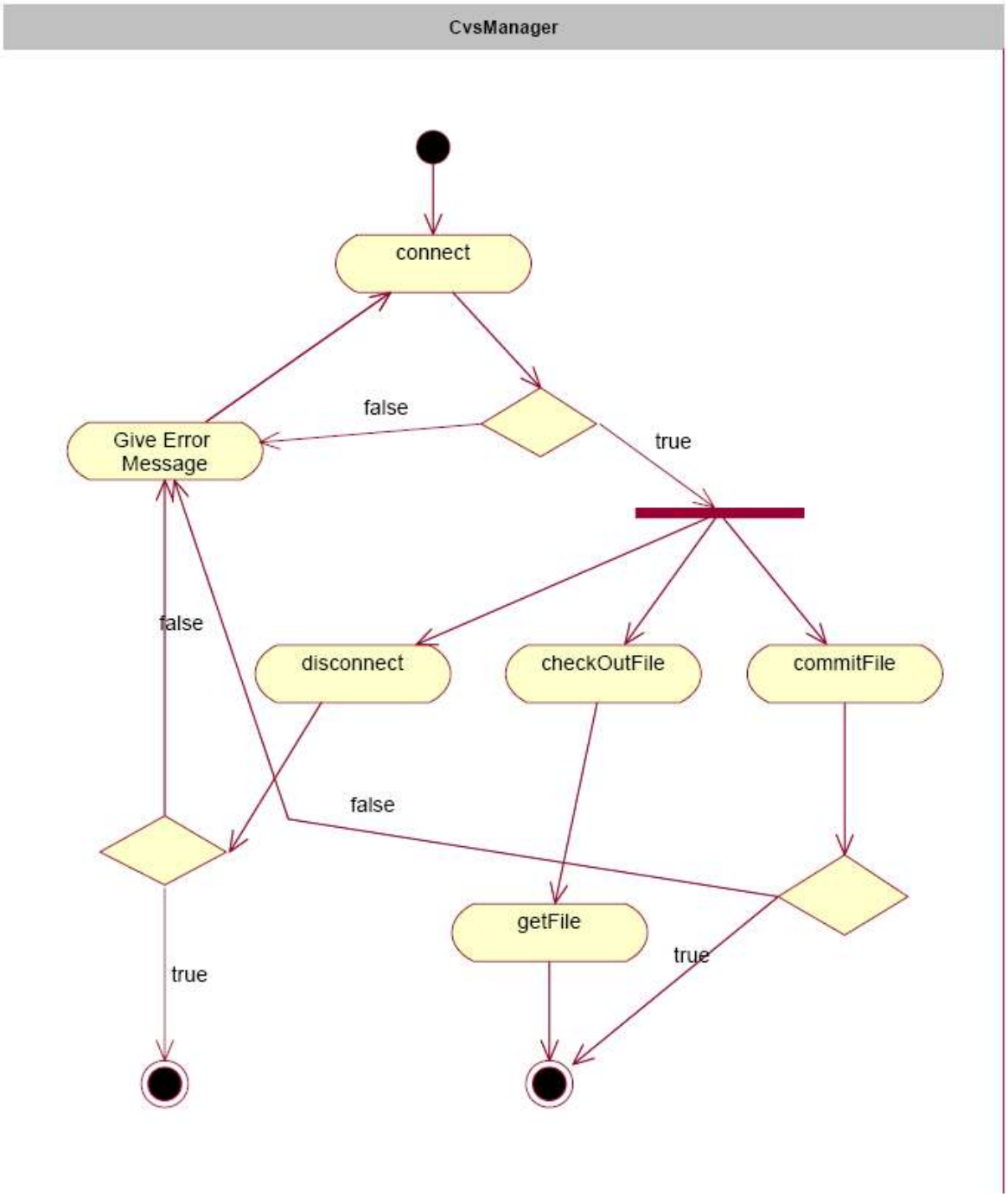
5.4.2 WYSIWYG Editor

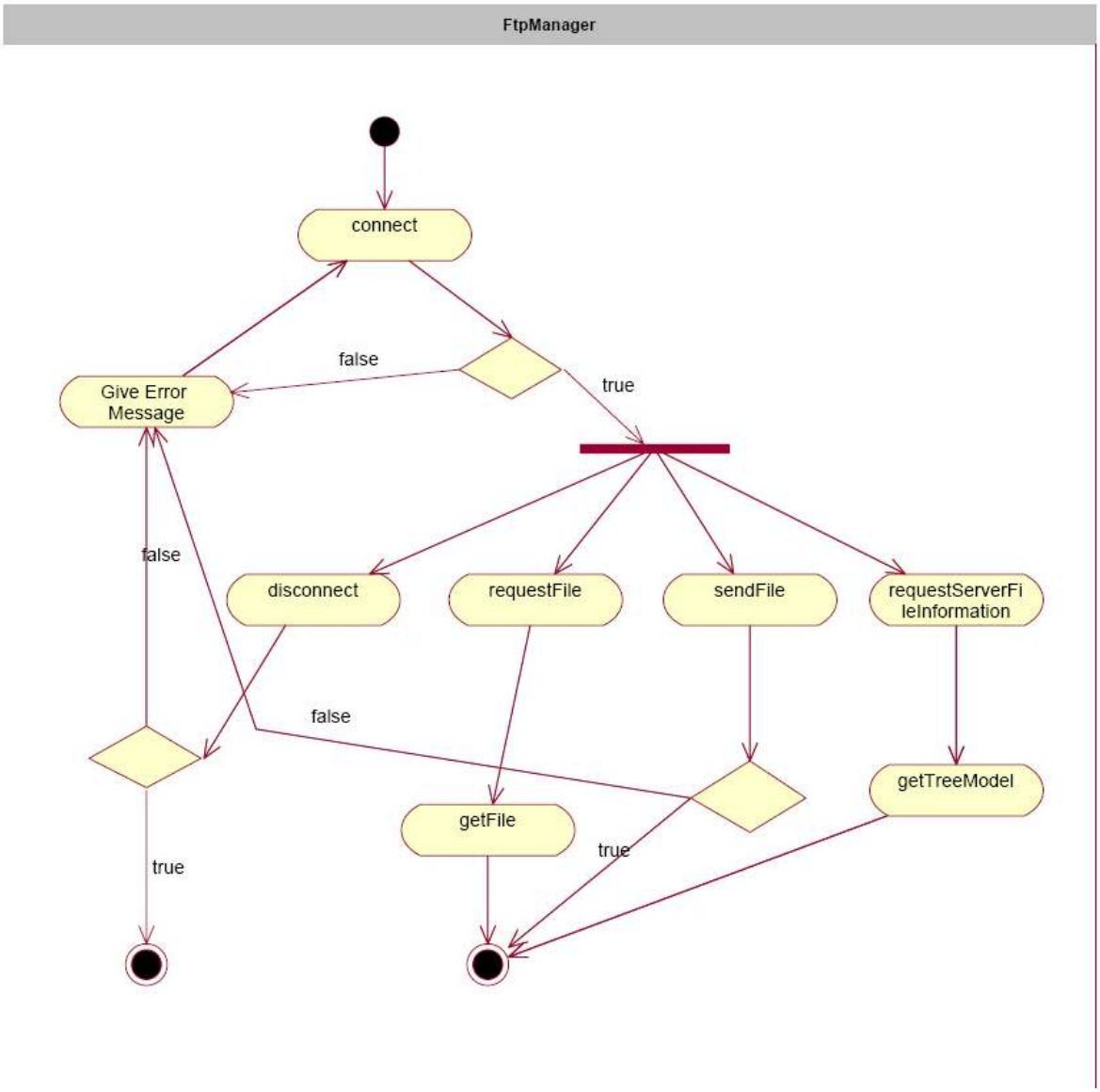


5.4.3 Database Editor



5.4.4 CVS – FTP


CvsManager
connect
false
Give Error Message
true
false
disconnect
checkOutFile
commitFile
false
getFile
true
true


FtpManager
connect
false
true
Give Error Message
false
disconnect
requestFile
sendFile
requestServerFileInformation
false
getTreeModel
getFile
true
true

6. GUI DESIGN

6.1. Overview of GUI

"GUI Design" is one of the most important parts of our project because it provides the permanent interaction of user with Integrated Development Environment. "Sihirbaz" has to provide developers a user-friendly environment which they can create interactive and rich web applications especially using AJAX actions. We designed a GUI that supports all features of our IDE in a user-friendly way and also view of our IDE should be nice-looking. We have investigated existing Development Environments such as "Aptana", "Tibco" and "JSE8" to be able to identify our design as an applicable combination of these well-designed tools. As stated before, we have already determined our detailed GUI functional requirements mainly in "Initial Design Report". We revised again our GUI to provide users more usability before writing the "Final Design Report".

Consequently, we have started to design and coding GUI of our development environment and it has almost finished. We have implemented nearly all functionalities that we stated in "Initial Design Report". We have also started to implement "Database Editor" module of our project so GUI of "Database Connection" window and "Database Editor" have already finished. During implementation our project, according to our needs, we have made and also we will probably make some refinements about GUI of "SiHiRBAZ".

As we decided to implement our project by using JAVA, we have used "`JAVA Swing`" package while implementing GUI.

6.2 GUI Requirements

- User can see “Code”, “Design” and “Browser” views in the middle of main window, each one will be placed in a different tab. S/he will be able to switch between these tabs.

- When user chooses “Code” tab, s/he will be able to write his/her source code with the help of a featured text editor.
    - If user right clicks in the “Code” view, s/he will be able to perform “Undo”, “Redo”, “Save” and “Cut”, “Copy”, “Paste” actions.

- When user chooses “Design” tab, s/he will create graphical design of his/her project by using a WYSIWYG editor.
    - If user right clicks in the “Design” view, s/he will be able to perform “Undo”, “Redo”, “Save” and “Cut”, “Copy”, “Paste” actions.

- When user chooses “Browser” tab, s/he will be able to see his/her application in an embedded browser.

- User can see “Project” and “Workspace” view at the left of “Code/Design” view, in tabbed structure.

- When user chooses “Project” tab, s/he will see all projects of development environment and select by double clicking any of them. If user selects one of these projects, that project will be set as current project and appears in “Workspace” view.
    - If user right clicks in the “Project” view, s/he will be able to perform “New”, “Open”, “Edit” and “Delete” actions.
    - User will be able to expand and enclose the hierarchical tree structure of projects.

• When user chooses “Workspace” tab, s/he will see current project and its files that s/he creates and will probably run. If user selects one of these files by double clicking on it, that file will be ready for editing or running and appears in “Code” view. User will also be able to see JavaScript variables and functions of classes of files.

- If user right clicks in the “Workspace” view, s/he will be able to perform “New”, “Open”, “Edit” and “Delete” actions for current project’s files.

• User will be able to see “DOM Inspector” view (Outline) just below the “Project / Workspace” view.

- When user chooses “DOM Inspector” view, s/he will see and reach all nodes which are tags of HTML/XML document of current project. If user chooses one of components by double clicking on it, that component's appearances will be highlighted in editor.
- There is also a table that shows “Node Name” and “Node Value” in “DOM Inspector” part of GUI.

• User will be able to see “Palette” view at the right of the “Code/Design” view. There are HTML and JavaScript components and AJAX Actions that are created before for the ease of user in this view.

- If user selects one of these components by clicking the icon of component and put it on the “Design” view (drag and drop), that component will be added to design and also its source code will be added to the file in “Code” view.
- If user wants to add a new AJAX action to the palette (the one that s/he creates or benefits from another source), s/he will click “Add New AJAX Action” button, and a window will be open for user to write the source code of action to be added.
- After making required connection and configurations about action, user will clicks “Add” button on window and new AJAX component will be added to palette.

• User will be able to see “Properties” and “Events” views that are in table structure just below the “Palette” view in tabbed structure.

- User will define his/her component’s properties (name, type, width, height, action etc.) by using “Properties” table.
- User will define his/her component’s events (handlers, actions) by using “Events” table.
- If user will click any cell of “Properties” or “Events” table, that cell will be ready to edit or update.

• User will be able to see “Debugger” view at the bottom of main window, with two tables which are “Call Stack” and “Variables” views.

- In “Call Stack” view, user will be able to see variables and functions currently placed in program stack.
- In “Variables” view, user will be click a cell, write name of the variable that s/he want to trace, and s/he will be able to see value of it during program flow.
- User will be able to add breakpoints at the line which is just left of “Code” view.

• User will be able to see “Database” view if s/he clicks to “Database Editor” from Tools submenu of “Menu Bar” and connects his/her database without any problem.

- When user clicks “Database Editor”, “Database Connector” dialog opens and gets information from user. Needed information is type of database (MySQL or Oracle), Server Host, Port, Username, Password and Schema.
- There are “Connect”, “Clear” and “Cancel” buttons in “Database Connector” dialog. After filling required fields, user clicks “Connect” button to connect stated database and schema. S/he also can use “Clear” button to clear all form.
- If request is accepted by DBMS, “Database Editor” view is shown to user to interact with his/her database.
- If request is denied system shows an error message and request account information again.

- After user connects to a database, schemas in that database will be shown to user at the left of the page. User can select a schema among the list.
- After a schema is selected its tables are shown as selectable items in tree view. User can select a table to view or modify.
- After a table is selected its rows and columns are shown at the screen in table view at just right of the schema view.
- User can select any row or column (attributes) in the tables by clicking on.
- If cell is empty user can write new value for that attribute, if it has a value, s/he can change it by using “Edit” icon, or discard the change by clicking “Discard” icon. Finally changes are applied by clicking “Apply” icon
- If user wants to delete an entry in a cell, s/he can right click and select “Delete” option to clear the selected cell.
- There is also “Refresh” button to refresh the tables after applying the recent changes.
- If user wants to execute his/her query by using the query window on the top of “Database Editor” view, s/he will write query and click “Execute” button to get the result of query.

• User will be able to see “Menu Bar” on the top of the main window.

- If user selects “File” submenu of “Menu Bar”, s/he can perform “New File”, “Open File”, “Close File”, “Save File”, “Save File As” and “Exit”.
- If user selects “Edit” submenu of “Menu Bar”, s/he can perform “Undo”, “Redo”, “Cut”, “Copy”, “Paste”, “Delete”, “Select All” and “Find” actions.
- If user selects “Project” submenu of “Menu Bar”, s/he can perform “New Project”, “Open Project”, “Run Project”, “Start Debugging” and “Step Over”, “Step Into”, “Step Out” actions.
- If user selects “Tools” submenu of “Menu Bar”, s/he can use “Database Editor” to connect database or send his/her files by using “FTP Connection” option.
- If user selects “Versioning” submenu of “Menu Bar”, s/he can use “CVS Manager”. User can easily “Commit” or “Check-out” his/her files.

- If user selects “Help” submenu of “Menu Bar”, s/he can choose “Help Contents” or “About”.

• User can to see “Toolbar” on the top of the main window, just below the Menu Bar.

- If user clicks any icon on the toolbar, s/he can perform the action of that icon.
- Possible icons that are shown on the toolbar are, “New File”, “New Project”, “Open”, “Save File”, “Cut”, “Copy”, “Paste”, “Undo”, “Redo”, “Find” “Run”, “Start Debug” and “Stop Debug”.

• If user runs his/her application or chooses “Preview in selected browser” option, s/he will also be able to see application in an external browser.

• Efficient keyboard shortcuts are provided for user.

- New (CTRL + N)
- Open (CTRL + O)
- Save (CTRL + S)
- Save As (CTRL + Shift + S)
- Find (CTRL + F)
- Cut (CTRL + X)
- Copy (CTRL + C)
- Paste (CTRL + V)
- Select all (CTRL + A)
- Undo (CTRL + Z)
- Redo (CTRL + Y)

- Keyboard shortcuts for pause, resume, step in/over/out, break will be provided.
    - Break (Pause)
    - Go (F5)
    - Step into (F11)
    - Step over (F7)
    - Step out (F8)

- Powerful keyboard navigation in the file system browser is allowed.
    - User will press 'ALT' and the file menu fill be opened.
    - User will use arrow keys to navigate on the menu.

6.3. *Screenshots of GUI*

In this part, screenshots of all GUI modules are shown.

6.3.1 “Code”, “Design” and “Browser” views

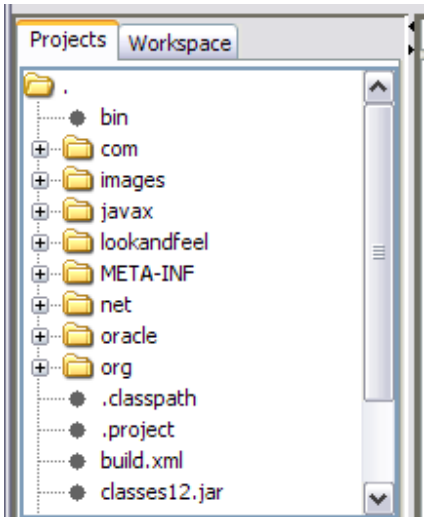
deneme.html | kodadi.html

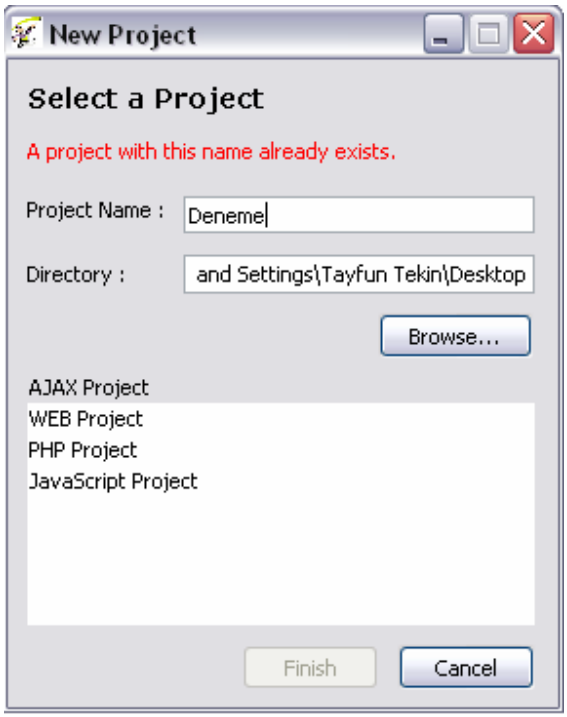
```
1  <html>
2  <head>
3  <title>Student</title>
4  </head>
5  <font face = "comicsans ms" color = "pink">
6  <body bgcolor = "purple">
7  <h1 align="center">This is heading 1</h1>
8  <p>
9  <a href="mypage.html">
10 This text</a> is a link to a page on
11 this Web site.
12 </p>

14 <h4>Table headers:</h4>
15 <table border="1"
16 bgcolor = "white">
17 <tr>
18   <th>Name</th>
19   <th>Telephone</th>
20   <th>Telephone</th>
21 </tr>
22 <tr>
23   <td>fulya</td>
24   <td>1347756</td>
25   <td>bos</td>
```

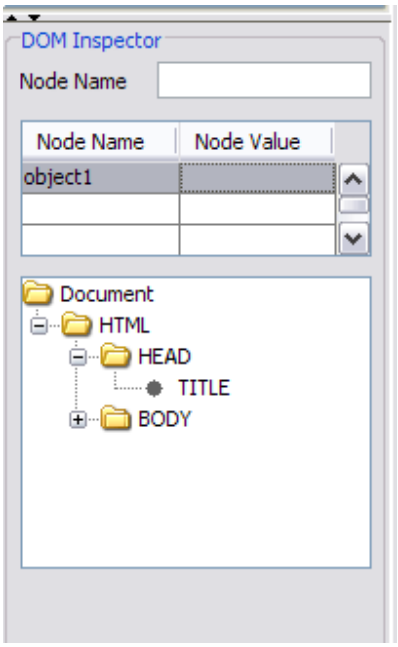
Code View | Design View | Browser View

6.3.2 “Project” and “Workspace” views



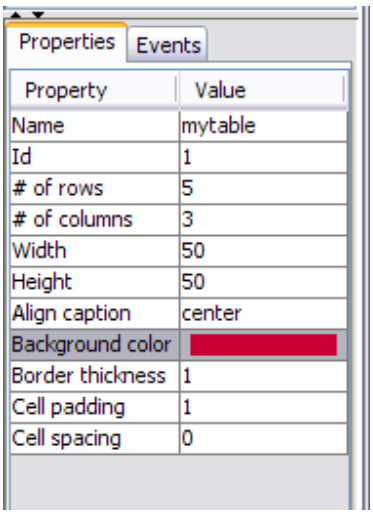


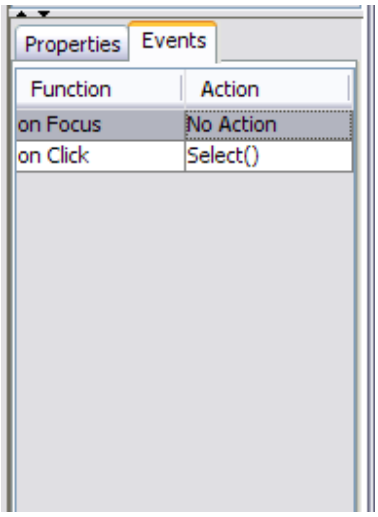
6.3.3 “DOM Inspector” view



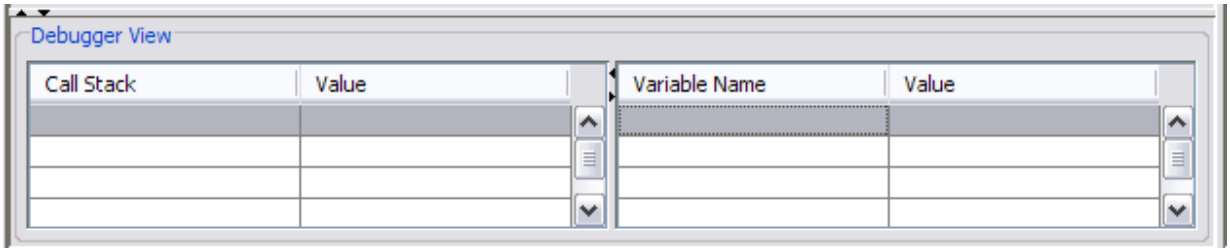
6.3.4 “Palette” view

6.3.5 “Properties” and “Events” views



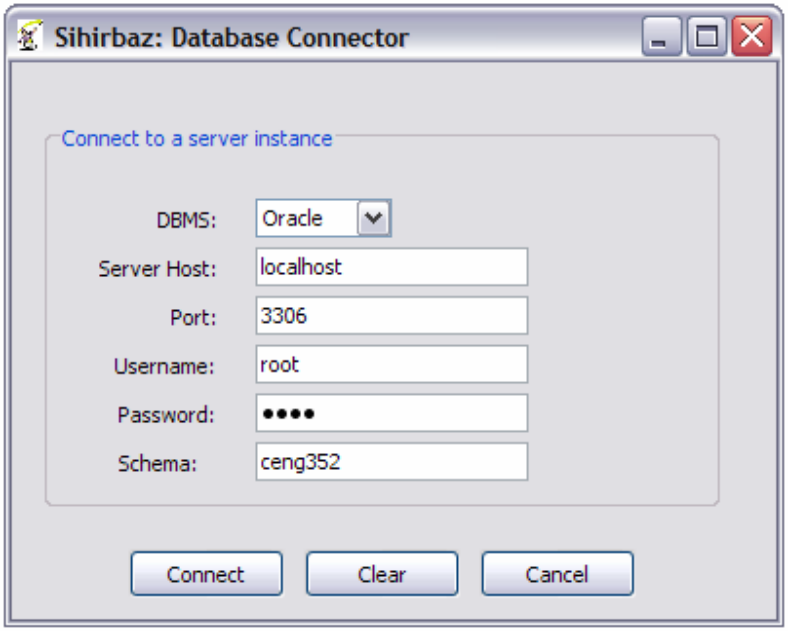


6.3.6 “Debugger” view

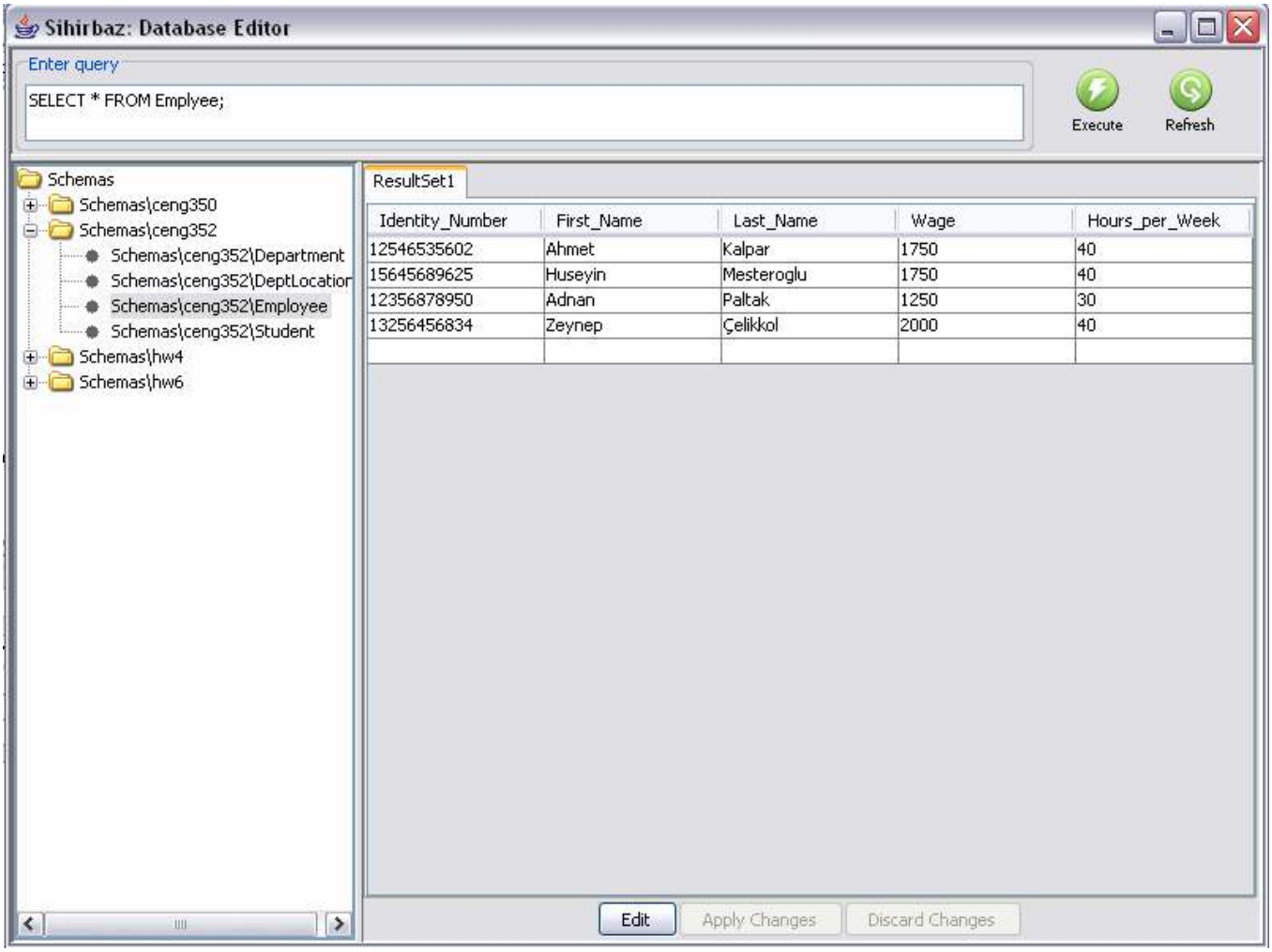


6.3.7 “Menu Bar” & “Tool Bar”

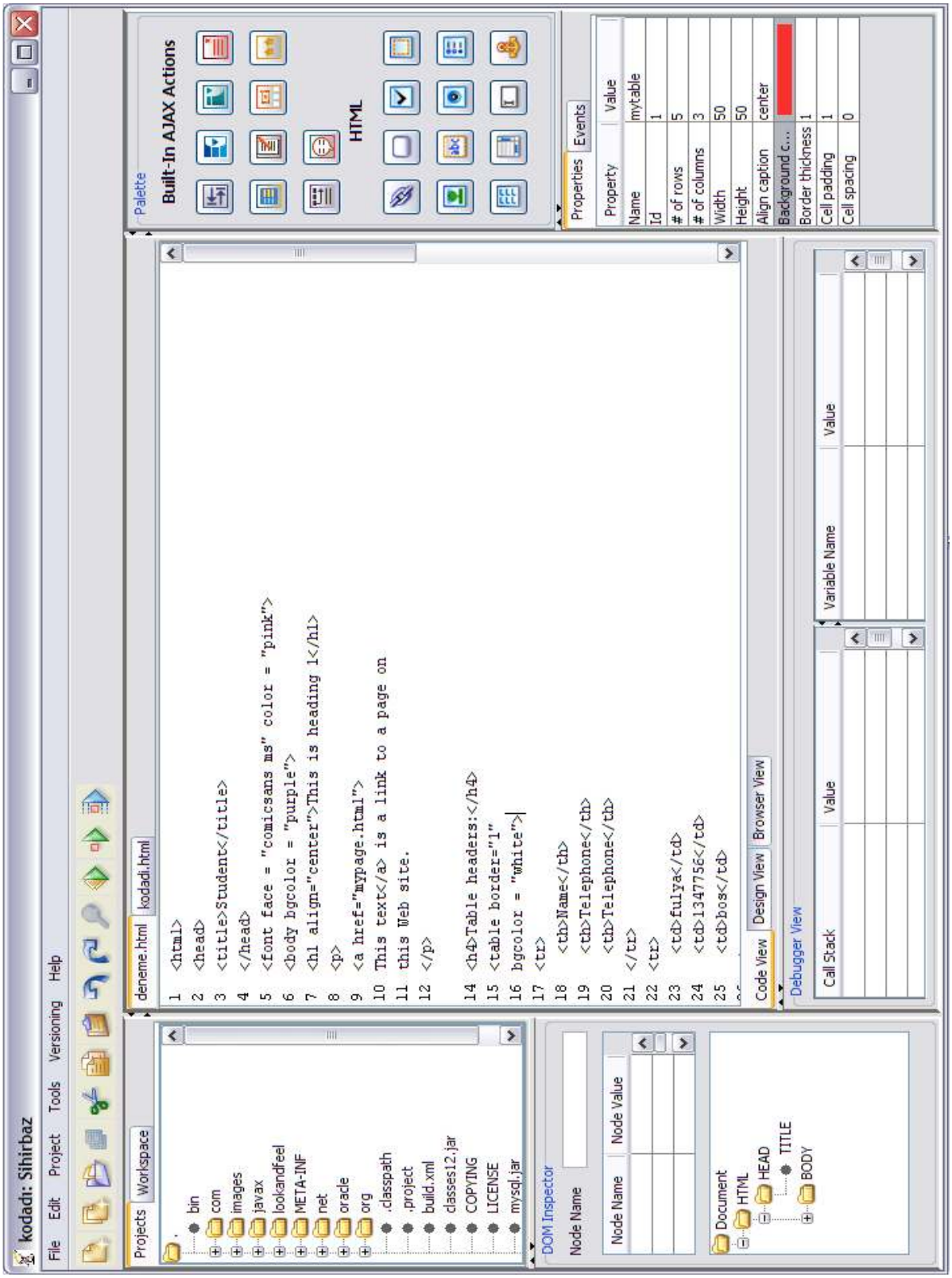
6.3.8 “Database Connector”



6.3.9 “Database Editor”



6.3.10 Final view of GUI



7. OFF-THE-SHELF COMPONENTS

7.1 Debugger

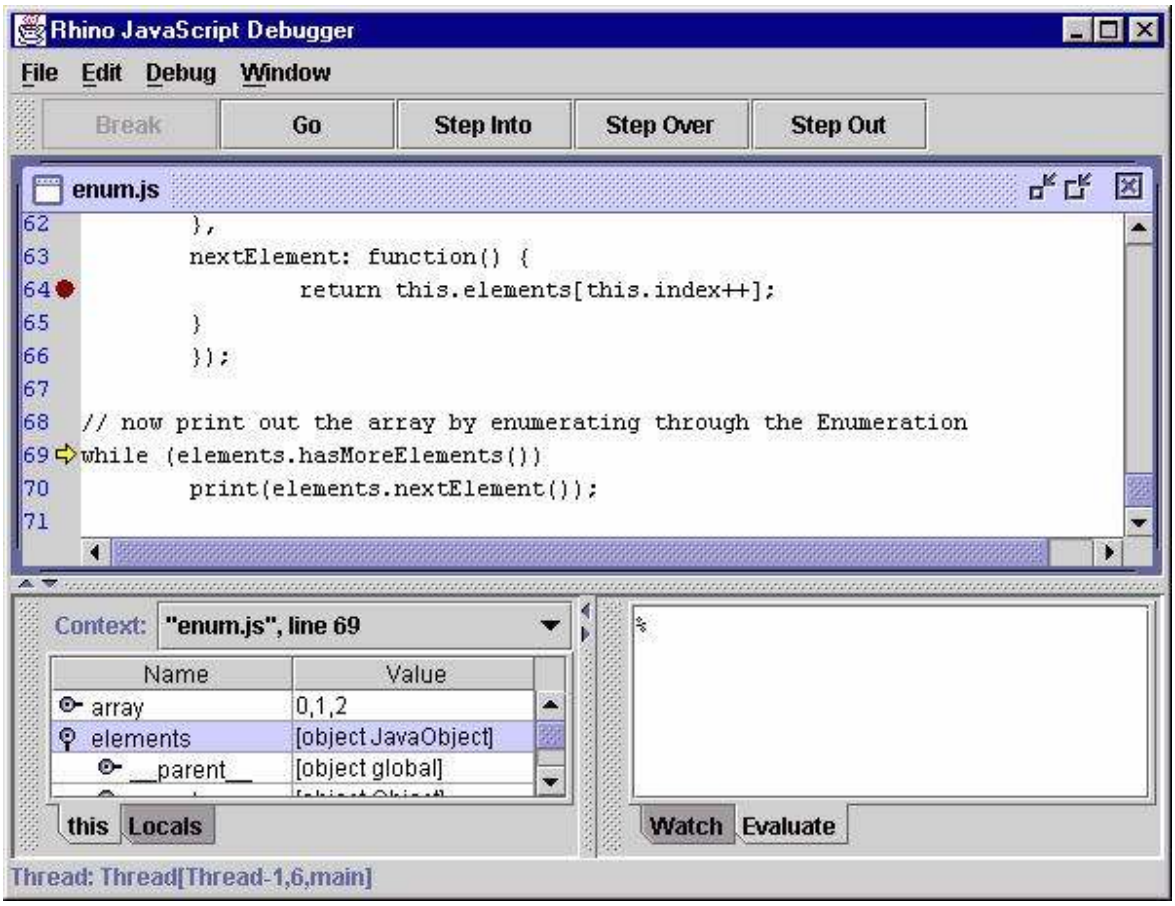
We are planning to use an open-source JavaScript Debugger. Rhino is an open-source implementation of JavaScript written entirely in Java. It is typically embedded into Java applications to provide scripting to end users.

All the features of JavaScript 1.5 which conforms to Edition 3 of the by Standard ECMA-262 *ECMAScript.*

- Documentation of Mozilla Rhino:
  - http://www.mozilla.org/rhino/doc.html

- Mozilla Rhino API Documents:
  - http://www.mozilla.org/rhino/apidocs/

Screenshot of the Rhino JavaScript Debugger :



7.2 Embedded Browser

We are planning to use JRex(powered by mozdev.org) which is a Java Browser Component with set of API's for embedding Mozilla GECKO within a Java Application. To embedded Mozilla GECKO to our project, first we need to build it.

- Mozilla GECKO is built by the instructions in these sites.
  - http://developer.mozilla.org/en/docs/Build_and_Install
  - http://gemal.dk/mozilla/build.html

- Documentation of JRex:
  - http://jrex.mozdev.org/docs.html

- JRex API Documents:
  - http://jrex.mozdev.org/docs/api/index.html

Features of JRex:

- Embedded Java Browser based on Mozilla GECKO.
- Event capturing like InputEvents (Mouse & keyboard), History, ContextMenu, ContentUrlListener, Observer, Progress, ToolTip.
- Compatible with AWT and Swing.
- Build in support for window and event management.
- Easy to use, developer need not know much of Mozilla details. The effective line of code for simple use is not more than 3 lines.
- Easy to use and easily extendable API's.
- Compatible with windows and *nix (Having GTK support).
- Compatible with Mozilla Gecko 1.4 and above. Has been tested with Mozilla Gecko 1.4 and 1.6 and 1.7.7
- Supports Tabbed and Java Internal Pane browser windows.
- Support for Profile & preferences.
- Support for Persist, Find & BroswerSetup (to enable/disable plug-in, image etc.) preferences.

- Support for accessing DOM objects of rendered page.
- JRex also implements DOM HTML2 for manipulating loaded HTML Document.
- In built support for Java WebStart deployment.
- In built support for LiveConnect which helps in communication between javascript and Java.
    - Can be used for communication between XUL and JVM in which JRex is running.

Screenshot of the JRex embedded into a Java Application :

8. SPECIFICATIONS

8.1 Syntax Specifications

Variable names: If a variable name consists of more than one word, first letter of each word except the first one will be capitalized: control, requestReturnData

Function names: Functions will be named with the same rule as variables: getControl, check

Class names: The first letter of every word in a class name will be capitalized: HomeIndoorArea, Student

Class members and methods will be written in the following order:

1. Private members
2. Protected members
3. Public members
4. Private methods
5. Protected methods
6. Public methods

There will be one empty line between function bodies. Only one member can be written on a line. There will be two empty lines after member declarations. Members in a same visibility will be grouped according to their data types. Example:

```
public class Student {

    private String name;
    private String surname;
    private int studentNumber;
    public char studentType;
```

```
        Student() {

                //body
        }

        protected int getStudentNumber() {

                //body
        }

        public String getName() {

                //body
        }
}
```

Functions: When writing a function the opening and closing brackets of functions will be on individual lines. Local variables in a function will be declared on top and local variables with the same data type will be grouped. Only one local variable can be declared on a line. After the declaration of local variables there will be two empty lines before starting to code. Example:

```
Bool checkDoorCollision(void) {
        int control;
        int index;
        float distance;
        Position cameraPos;



        //code starts here.
}
```

Conditionals and loops: Opening and closing brackets of conditional and loops will be on individual lines. Example:

```
if( ) {
        // condition body
}
```

Comments:

- At the beginning of every file, the author of it, the date file is created and the date file was last modified will be written in a comment with the following syntax.

```
/**
        @author: Fulya Oktay
        Created 01.12.2006
        Modified 01.12.2006
*/
```

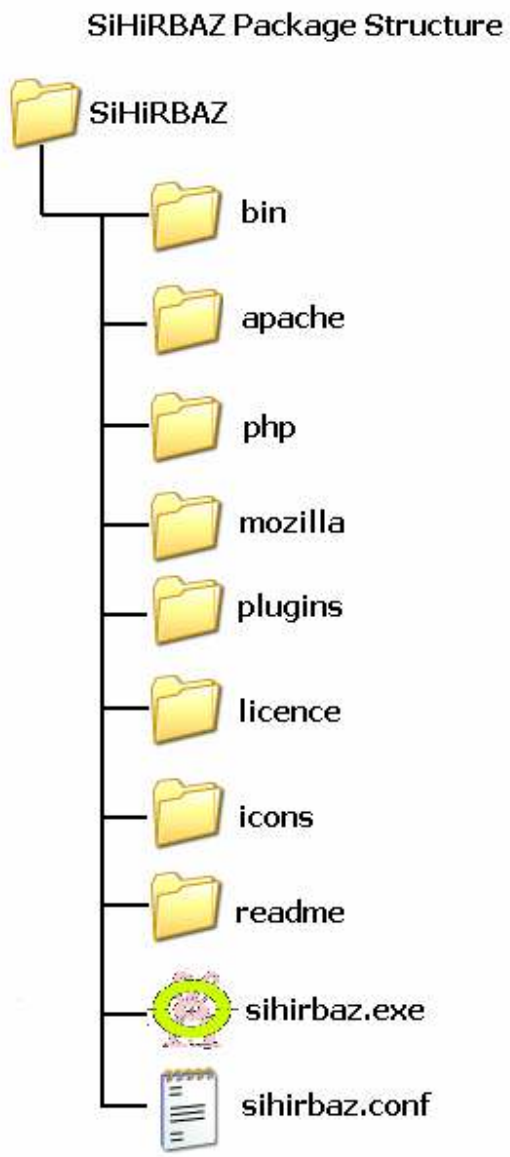
- Before the 'if' and 'for' expressions, the purpose of them will be stated in a comment.
- Before every class definition, the component which the class specifies will be stated in a comment.
- Before every function definition, the functionality will be stated in a comment.
- For every attribute of the class, an explanation will be provided in a comment.

Syntax for comments is

```
/**
        Comment
*/
```

8.2 Project Management Specifications

8.2.1 SiHiRBAZ Package Structure

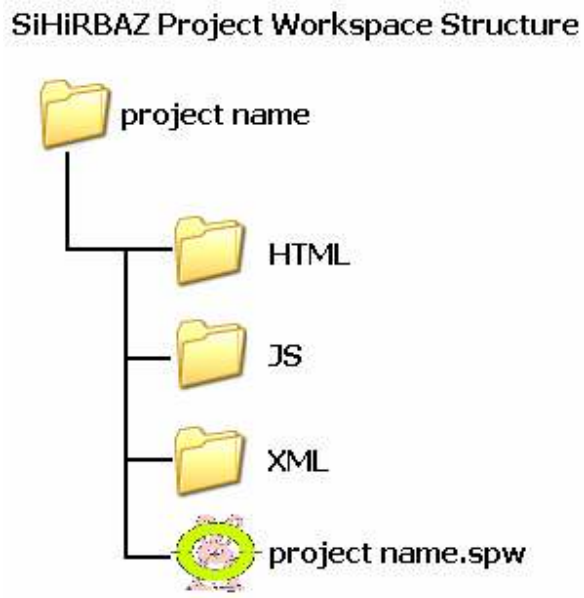
	* Our Product will be existing in SiHiRBAZ directory after installation. SiHiRBAZ Package Structure consists of some necessary directories and files. * In the bin directory, there exists binary project files. * In the apache directory, there exists apache web server. * In the php directory, there exists php software.(php and apache directories may not exist in the package if the user has already installed them on his/her pc.) * In the mozilla directory, there exists open-source Mozilla web browser for embedded browser support of our product. * In the plugins directory, there exists plugin files if avaliable. * In the licence directory, there exists necessary licence files. * In the icons directory, there exists project icons image files. * In the readme directory, there exists all necessary help and readme files of our product. * There exists a **sihirbaz.exe** file which is an executable file to run our product on Windows. There will be Linux executable for Linux Package. * There exists **sihirbaz.conf** file. This is a XML file. Our product first read this file. Necessary information is existed about the all project the user have created.

A sample sihirbaz.conf file will be like this :

- **<sihirbaz>** is a root tag, it contains **<projects>** tag and **<java_jdk>** tag.
- **<projects>** contains **<project>** tag which provides necessary information about projects. There is an attribute **current** for currently opened project in the workspace view of the product.
- Between the **<project> </project>** tags there exists there exists **<name>** and **<path>** tags.
- In the **<java_jdk>**, there exists a path for installed Java JDK.

```
<?xml version="1.0" encoding="UTF-8"?>
<sihirbaz>
    <projects current="Deneme">
        <project type="AJAX Project">
            <name>Deneme</name>
            <path>C:\Documents and Settings\Tayfun Tekin\Desktop</path>
        </project>
        <project type="AJAX Project">
            <name>Sihirbaz</name>
            <path>C:\Documents and Settings\Tayfun Tekin\Desktop</path>
        </project>
    </projects>
<java_jdk>
    <path>C:\Program Files\Java\jdk1.5.0_06</path>
</java_jdk>
</sihirbaz>
```

8.2.2 SiHiRBAZ Project Workspace Structure

	* A project created by the user will be saved in to “project name” directory. Location of the project directory is anywhere in the local drive of the computer chosen by the user. * There exists three directories for the created project files which are HTML, JavaScript and XML files. * There exists a **project name.spw** (SiHiRBAZ Project Workspace) file which will be in XML format. User will able to run our product with this project by clicking this file.

The contents of the file sample .spw file will be as follows:

This file includes information about the project which are its name, path and type.

```
<?xml version="1.0" encoding="UTF-8"?>
<project type="AJAX Project">
        <name>Deneme</name>
        <path>C:\Documents and Settings\Tayfun Tekin\Desktop</path>
</project>
```

9. TESTING ISSUES

9.1 Testing Plan and Strategy

In order to present an error-free and defect-free product we need to make some tests. For this purpose, we have decided on some testing strategies and built a testing plan during our design interval. Since we will have very little time for testing, we tried to simplify our strategy and concentrate on an efficient strategy rather than trying to do all real software test methods.

To see how easily our software can be tested we check our project with according to several characteristics:

Operability: From the beginning we will try to work carefully and eliminate errors. This will help us to test our product easily. Several modules and tasks will be prepared in order to perform efficient tests and obtain better results.

Observability: We will prepare distinct error and warning messages.
Controllability: We decomposed a job into several units in order to control the actions easily.
Decomposability: Several modules and tasks will help to uncover errors.
Simplicity: We'll also try to code as efficiently as possible.
Stability: Separate modules will help us for the stability. Past tests won't be invalid. Function and module dependencies, architecture are all understood clearly by group members and this will help in testing. Also our large document archive will help this process.

We will test

- User interaction
- Data manipulation
- Display processing and generation

Below are the methods we will use.

9.1.1 Unit Testing

In the unit test case we will test each module separately. White box testing will be used to both detect the errors and correct them. We will test the components by passing data through it and we will be monitoring data to find the errors.

We will make sure that all the components work correctly and efficiently. The test will be done primarily by the programmer who designed and implemented the module. If necessary, the other programmer will do the second testing for the same module.

All the important paths will be tested with a white box method. Rather than the complete program, all of the modules will be tested individually. Below are the modules:

- **GUI Testing**
- **Text Editor**
- **Database Editor**
- **WYSIWYSG Editor**
- **JavaScript Debugger**
- **CVS Support**
- **FTP Support**

9.1.2 Integration Testing

Although we can find errors in modules by unit test, we must also make an integration test in order to find errors due to integration of the modules. We will examine the product from the user's perspective for making integration test. We are planning to use an incremental integration for this manner. Smoke testing may be the most suitable because of the time interval however we won't have time to test or product daily. This is unrealistic. We will probably use bottom-up integration.

We will be looking whether all the modules work correctly, i.e. is data correctly managed, are interface features easy to understand and use, does the product really do the job we want, is there any confusion where more than one person uses the product, etc. All of these tests will be implemented from the perspective of a user. However it will not be possible to see all the errors, and there may probably be defects. Some other tests are still needed.

9.1.3 Validation Testing

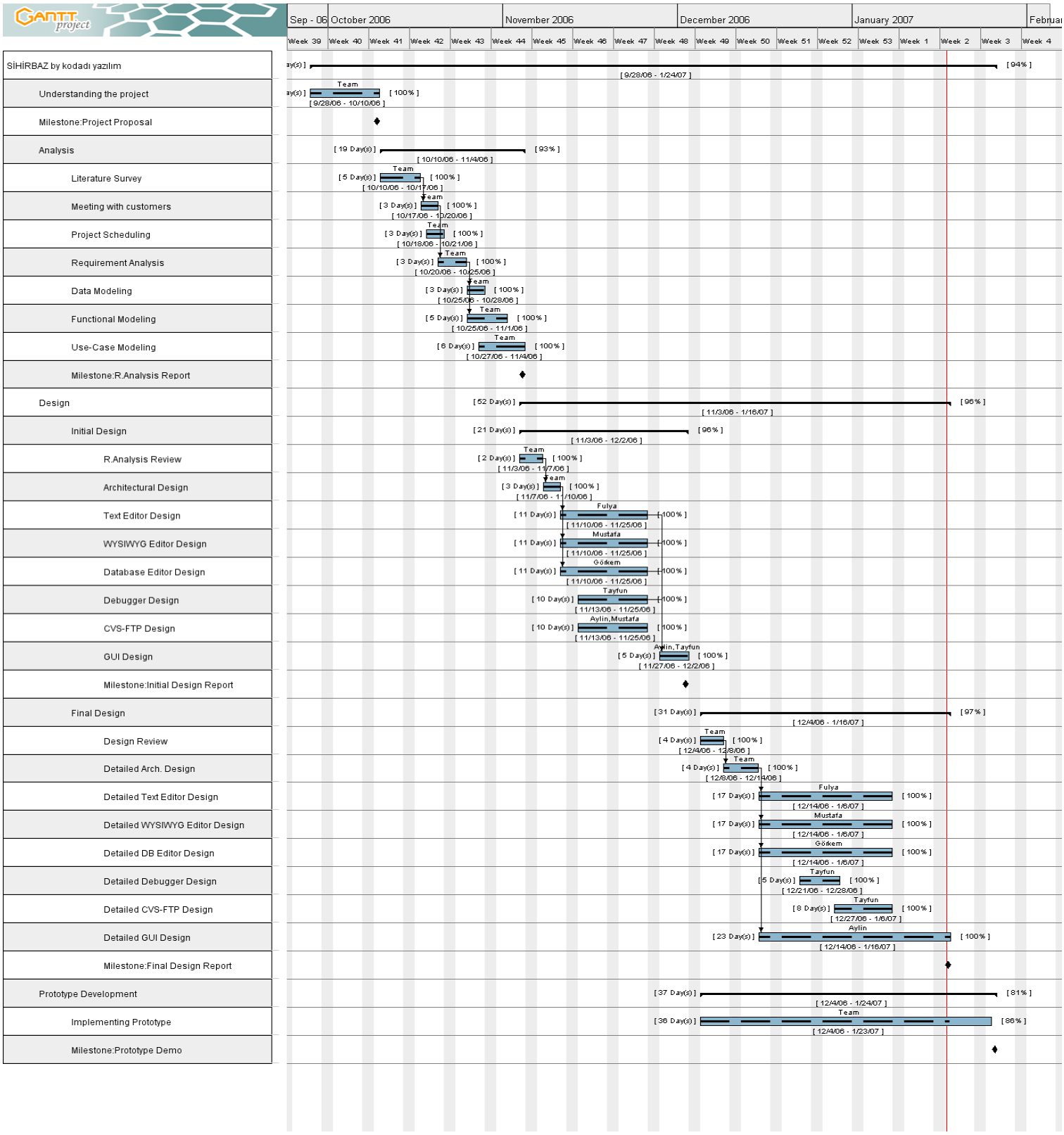
Validation asks: "Are we building the right product". And the answer specifies whether our program will be preferred by the web developers or not. Therefore validation is important.
We will perform a black box testing too. Use cases will be used in order to specify all the needed requirements and obtain possible errors.

Beta Testing: It is virtually impossible for us to foresee how the customer will use our program. We are especially interested in alpha testing. Therefore we will release an alpha version of the product before the demo deadline. Since our customers are web developers, we believe we will obtain some error reports from our friends who have experience in web developments and Ajax actions. In addition, we are planning to put our product on web site and do advertisement in some communities and forums related with Ajax applications.

10. CONCLUSION

This is the Detailed Design Report of "SiHiRBAZ" project. During preparing this report, we have tried to decide on the way we will implement our product. We have reviewed our initial design report and made some refinements. In addition the diagrams we have drawn before, we provide activity diagrams for this report.

11. APPENDIX


GANTT project
Sep - 06
October 2006
November 2006
December 2006
January 2007
Week 39
Week 40
Week 41
Week 42
Week 43
Week 44
Week 45
Week 46
Week 47
Week 48
Week 49
Week 50
Week 51
Week 52
Week 53
Week 1
Week 2
Week 3
Week 4
SİHİRBAZ by kodadı yazılım
Understanding the project
Milestone:Project Proposal
Analysis
Literature Survey
Meeting with customers
Project Scheduling
Requirement Analysis
Data Modeling
Functional Modeling
Use-Case Modeling
Milestone:R.Analysis Report
Design
Initial Design
R.Analysis Review
Architectural Design
Text Editor Design
WYSIWYG Editor Design
Database Editor Design
Debugger Design
CVS-FTP Design
GUI Design
Milestone:Initial Design Report
Final Design
Design Review
Detailed Arch. Design
Detailed Text Editor Design
Detailed WYSIWYG Editor Design
Detailed DB Editor Design
Detailed Debugger Design
Detailed CVS-FTP Design
Detailed GUI Design
Milestone:Final Design Report
Prototype Development
Implementing Prototype
Milestone:Prototype Demo