MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

SENIOR PROJECT
FALL 2006

INITIAL DESIGN REPORT

03.12.2006

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose of the Document

This is an initial design report for our project *"kodadı: AJAXDEV"*. The purpose of this document is to express the initial 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 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 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 gannt chart to the report to show the progress of our project.

1.2 Scope

AJAXDEV project consists of mainly 5 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 planning to

write 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 version control system to the user and ftp 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 will implement 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 decided to find 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

Gannt Chart can be found in Appendix.

4. SYSTEM MODULES

4.1 Text Editor

We are planning to write an HTML text editor 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.
    - When the user has written a separate word, 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 redouable.
        - 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.

- It will provide commands for commenting and commenting out code.
  - Enable user to select multiple rows.
  - Understand what language the selected code belongs to.
  - Comment the unselected code by putting the related comment item to it.
  - Comment out the selected commented code by removing the comment items on it.

- 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 cursor position, related code and operation type from WYSIWYG editor.
    - If the operation type is insertion, the related code will be added to the cursor position.
    - If the operation type is deletion, the related code will be deleted from the cursor position.
    - If the operation type is update, the code that will be deleted will be deleted and the code that will be inserted will be added to the cursor position.

- Automatic save is provided to prevent user from loosing data.
    - Count the modifications the user has made.
    - When this count is 3 save the current entry to the temporary 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

- User can drag & drop CSS into his design.
    - If user inserts a CSS item open dialog box.
    - Prompt user to enter a site for CSS.
    - 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".

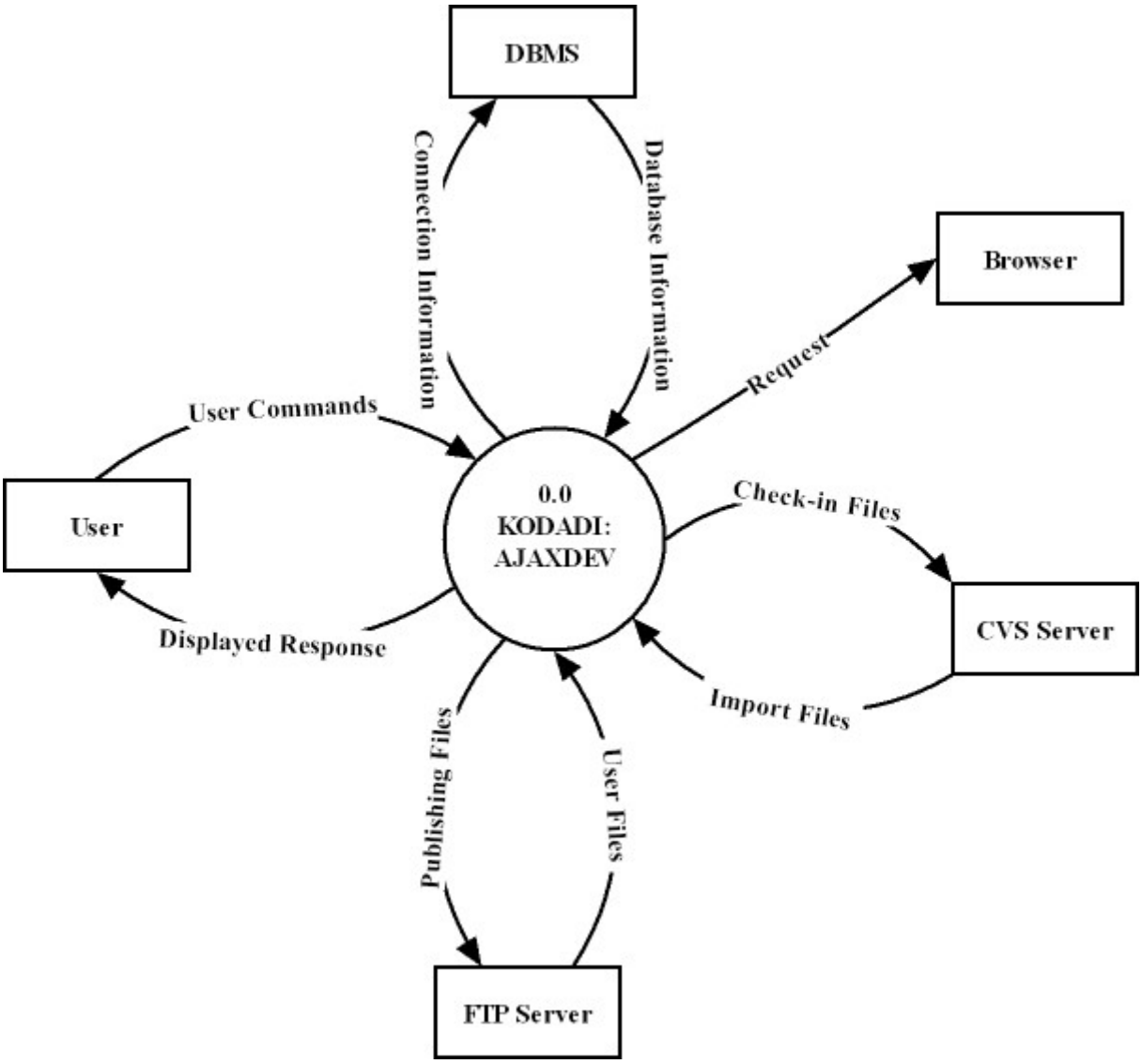
- User selects a file and clicks "CVS Commit".
    - If request can be done user will be able to create a new revision of the file, containing his/her changes, into the repository.
    - If a file commit is not allowed by server, an error is shown to user.

- 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.

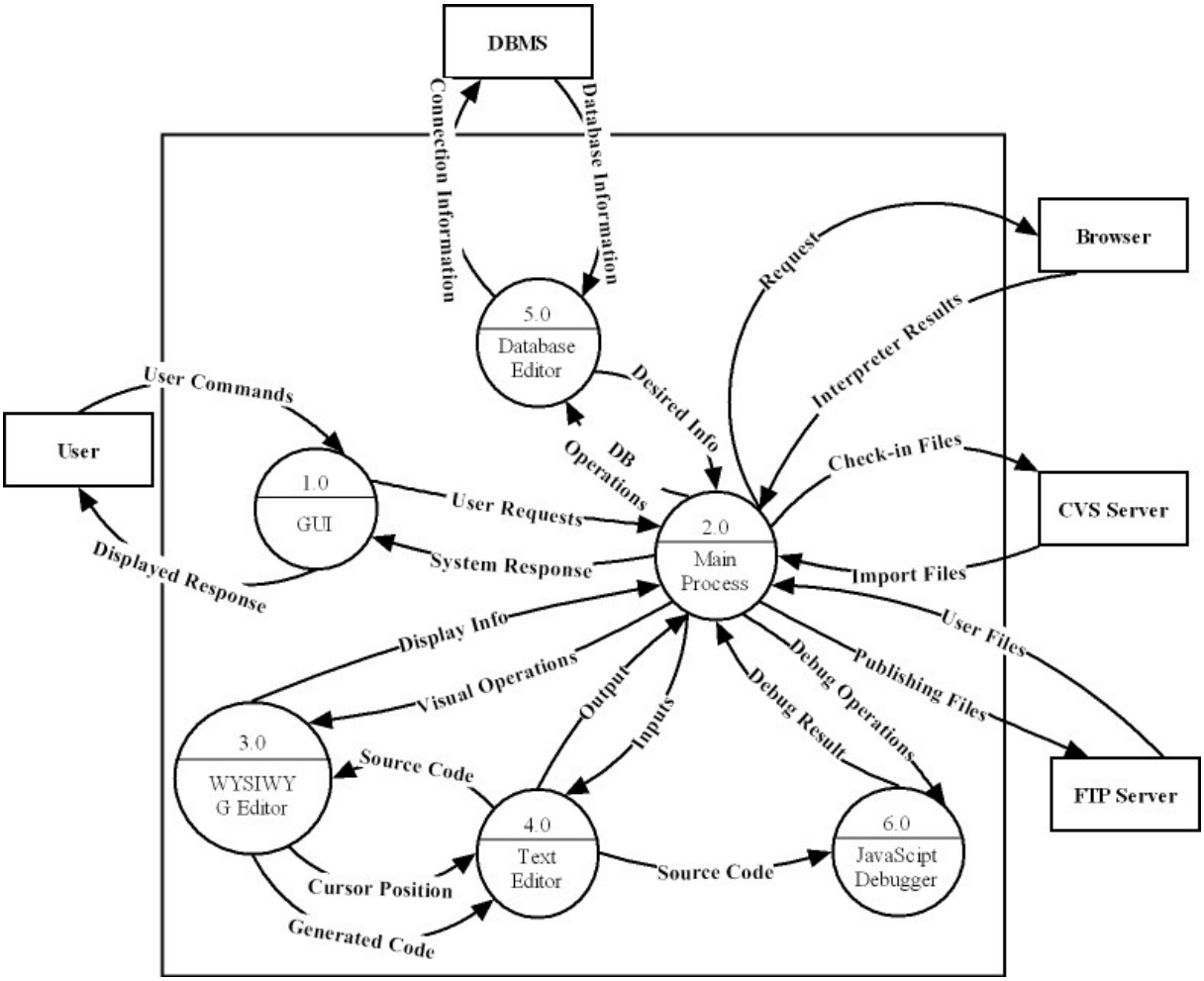
- User can close connection by pressing a button.
    - If user requests a connection close, a close signal is sent to server.

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

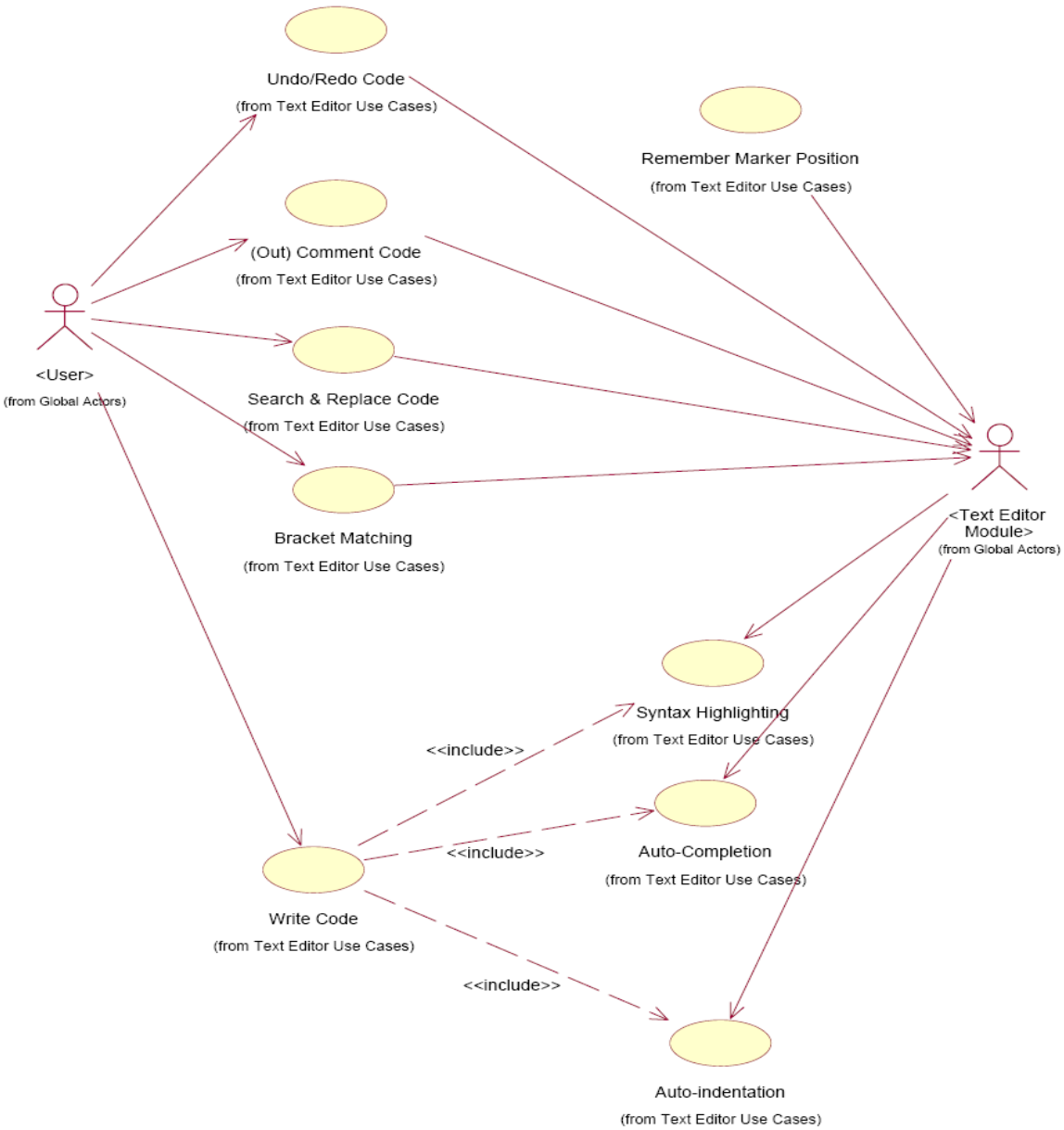
name:	Database Operations
where used / how used:	Database Editor (5.0) *input* Main Process (2.0) output
description:	User requests on database

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 Case Diagrams and 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.

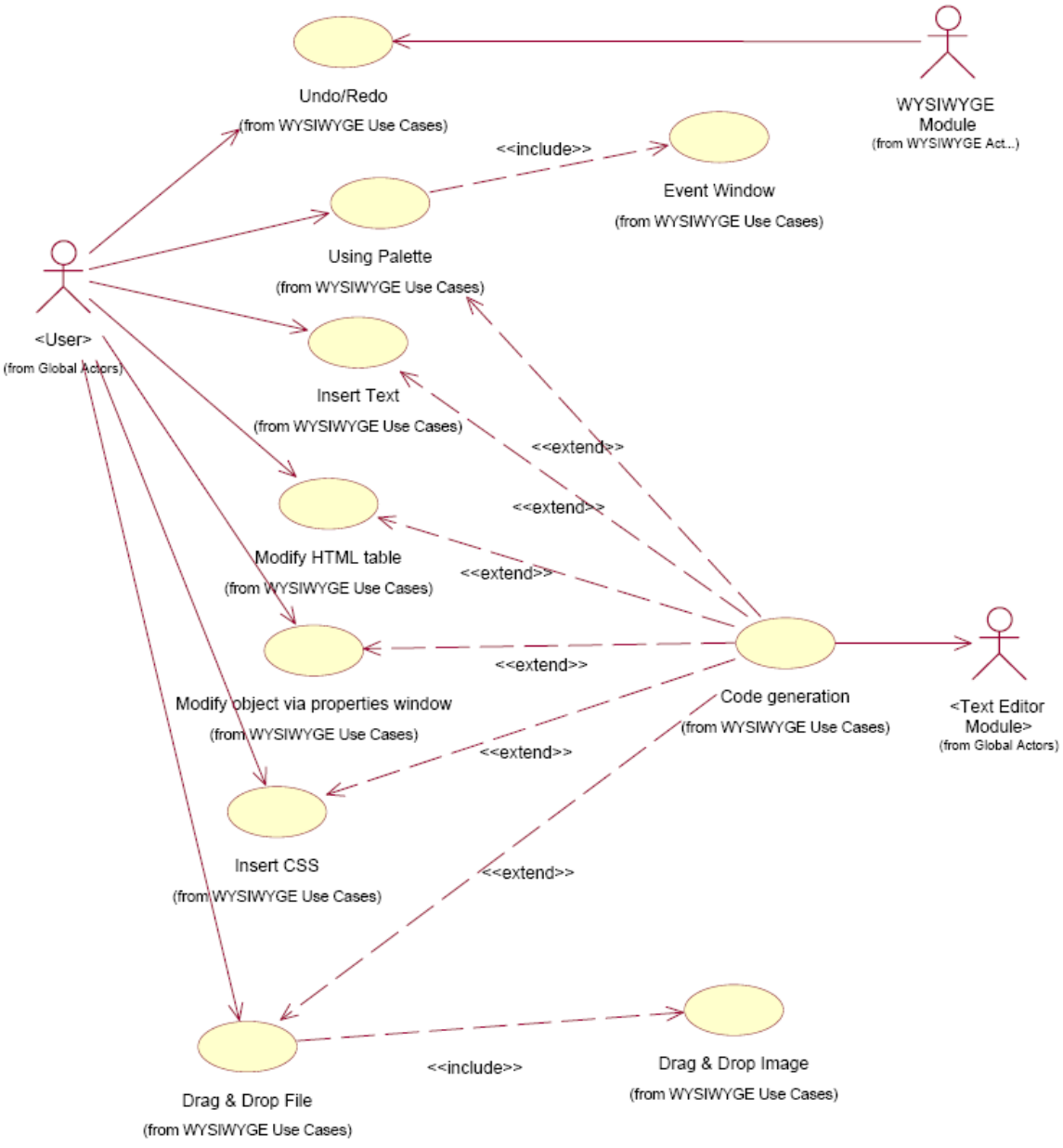
Link Checking: When the user has entered a link, editor will automatically highlight it as a link.

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.

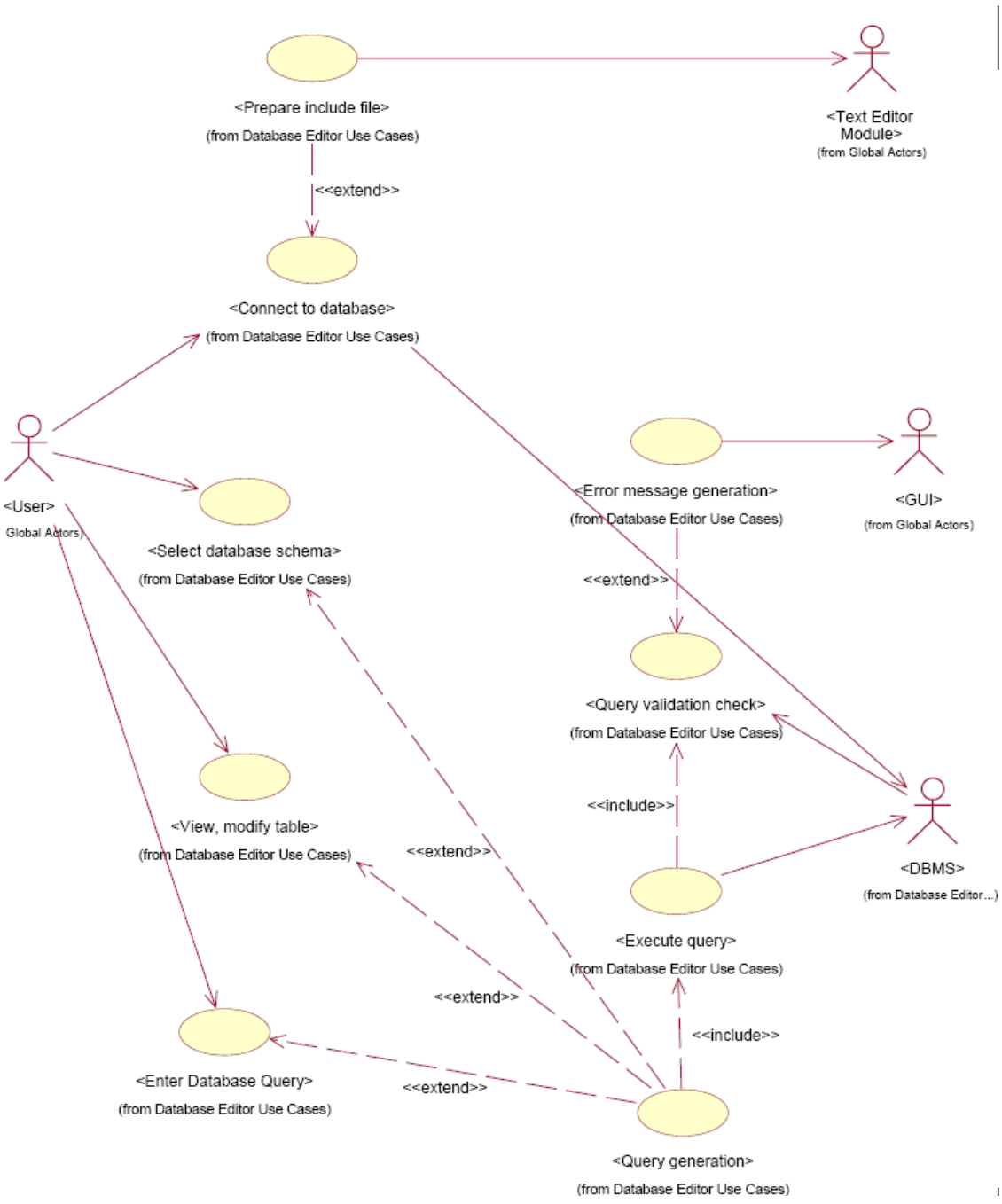
Customizing the properties of element on properties editor: User will arrange the desired properties of elements.

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



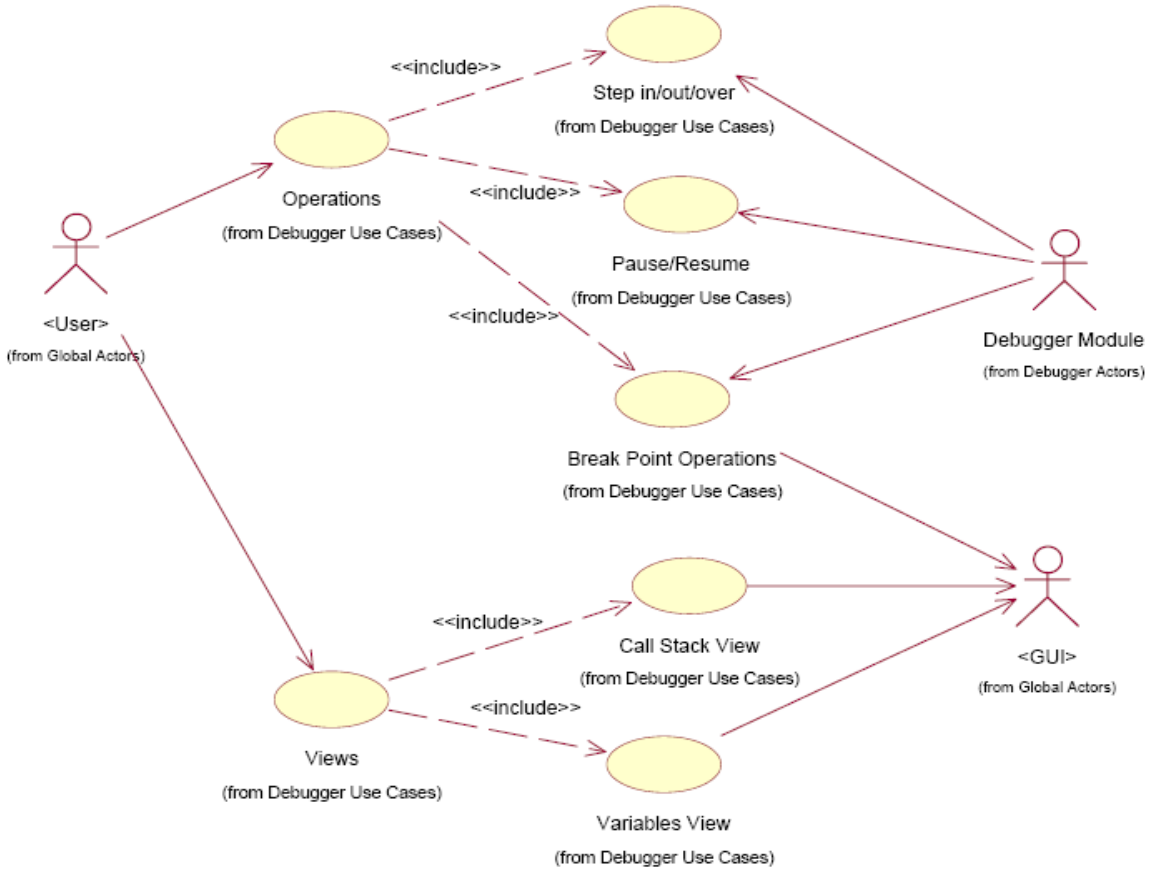
Connect to Database: User will press connect button. Then user interface will bring up connection dialog and waits for the user to enter connection info. After user enters connection info, user interface will send it to DBMS. If the connection info is correct, DBMS will return database info and user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

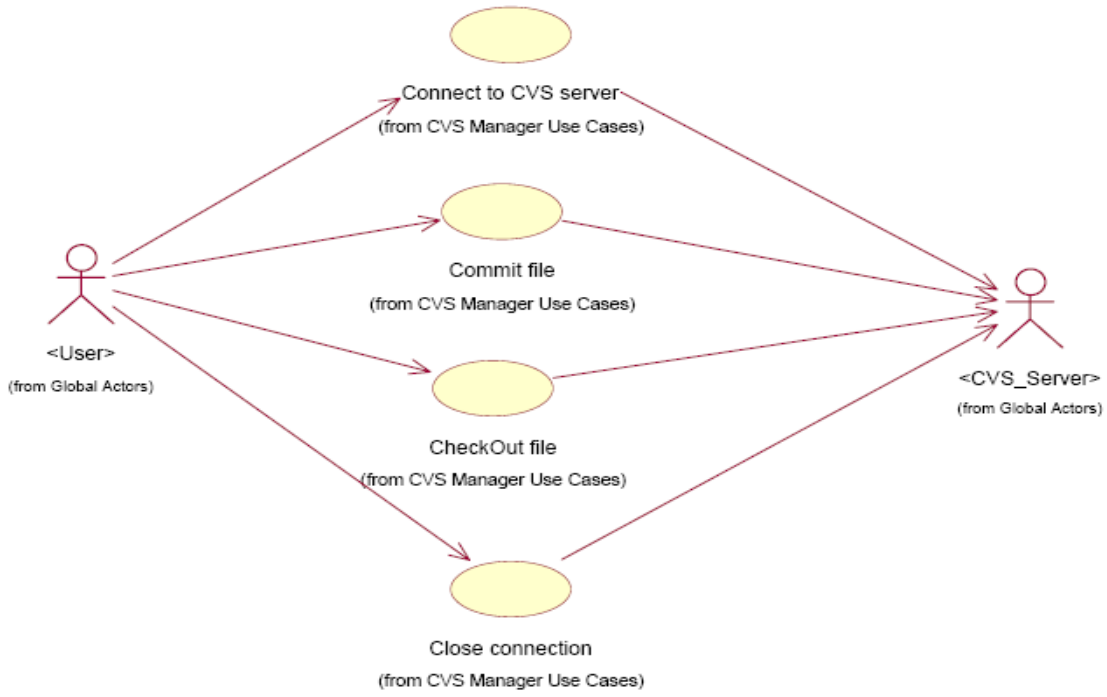
Step in/over/out: User will press the Step in/over/out button. The debugging engine will go in/over/out the execution step of the scripts its debugging.

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.

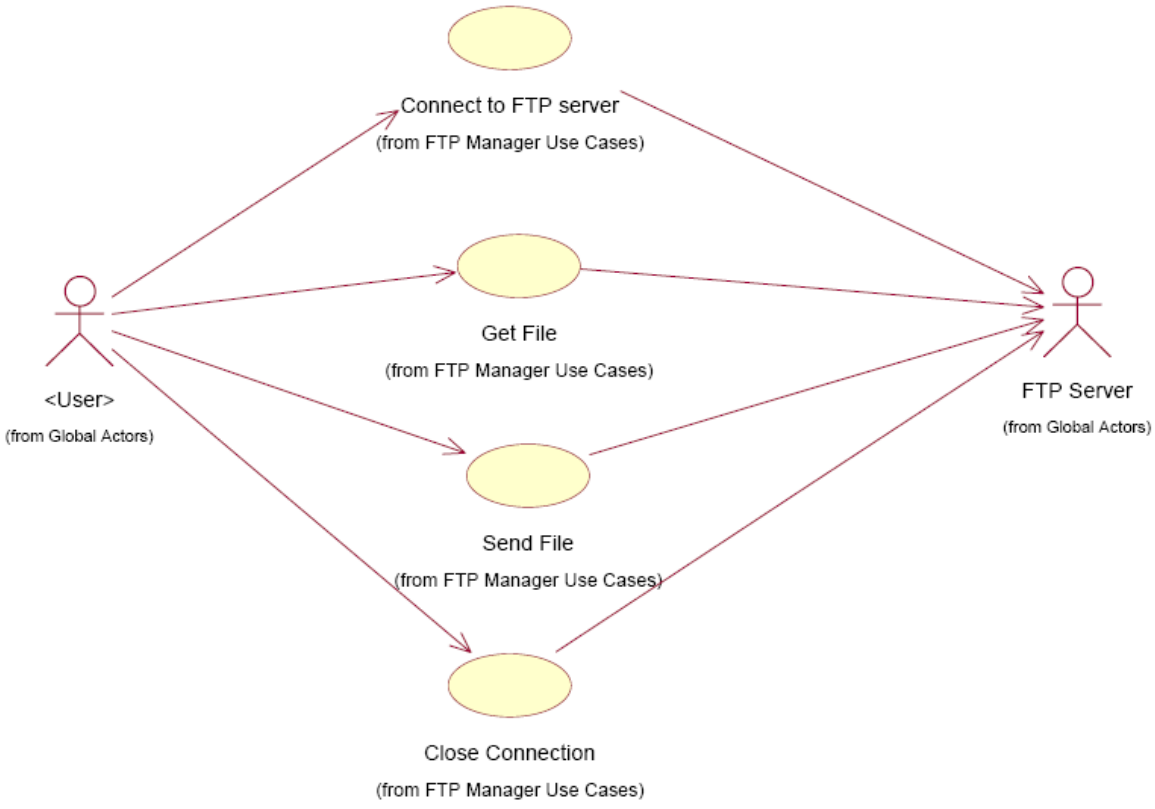
Call Stack view: When the debugger is stopped, the Call Stack view displays the list of active functions.

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

5.1.5 CVS Manager

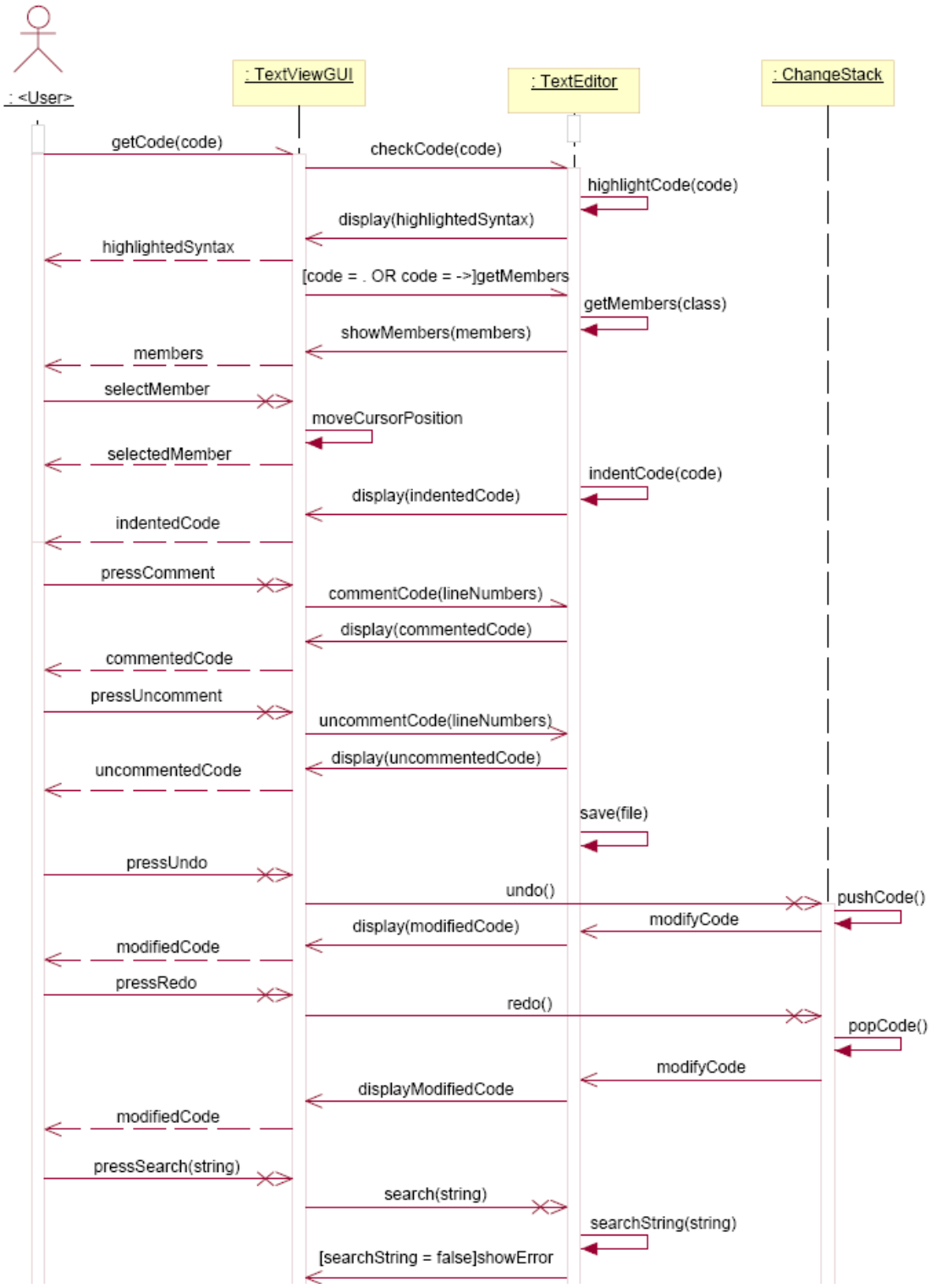


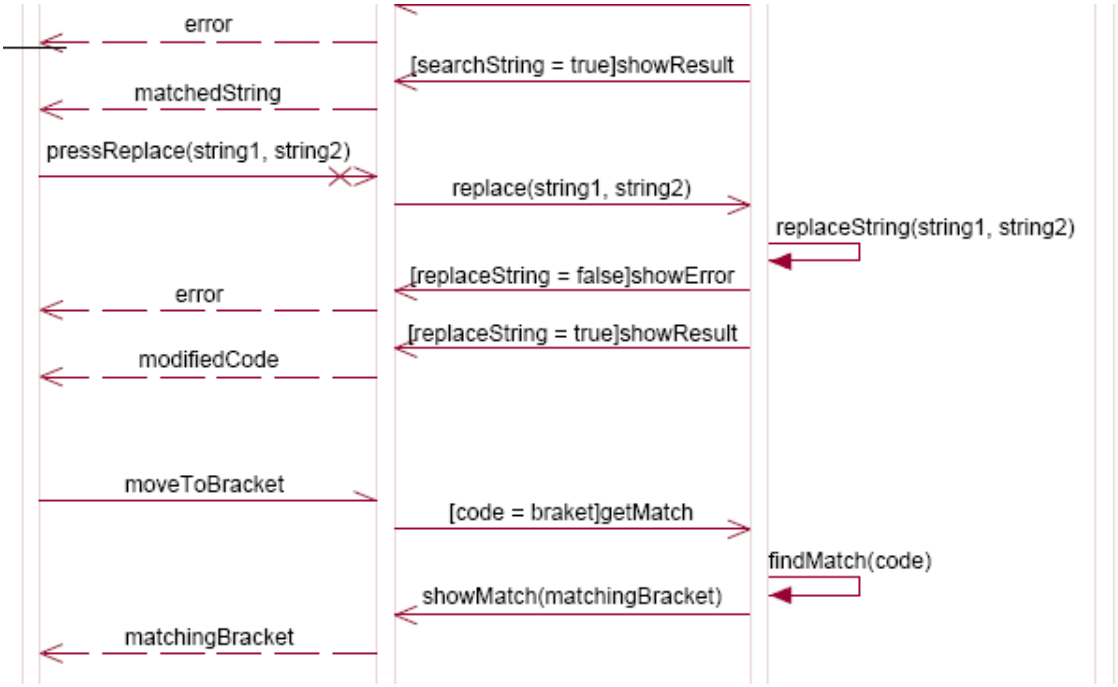
5.1.6 FTP Manager



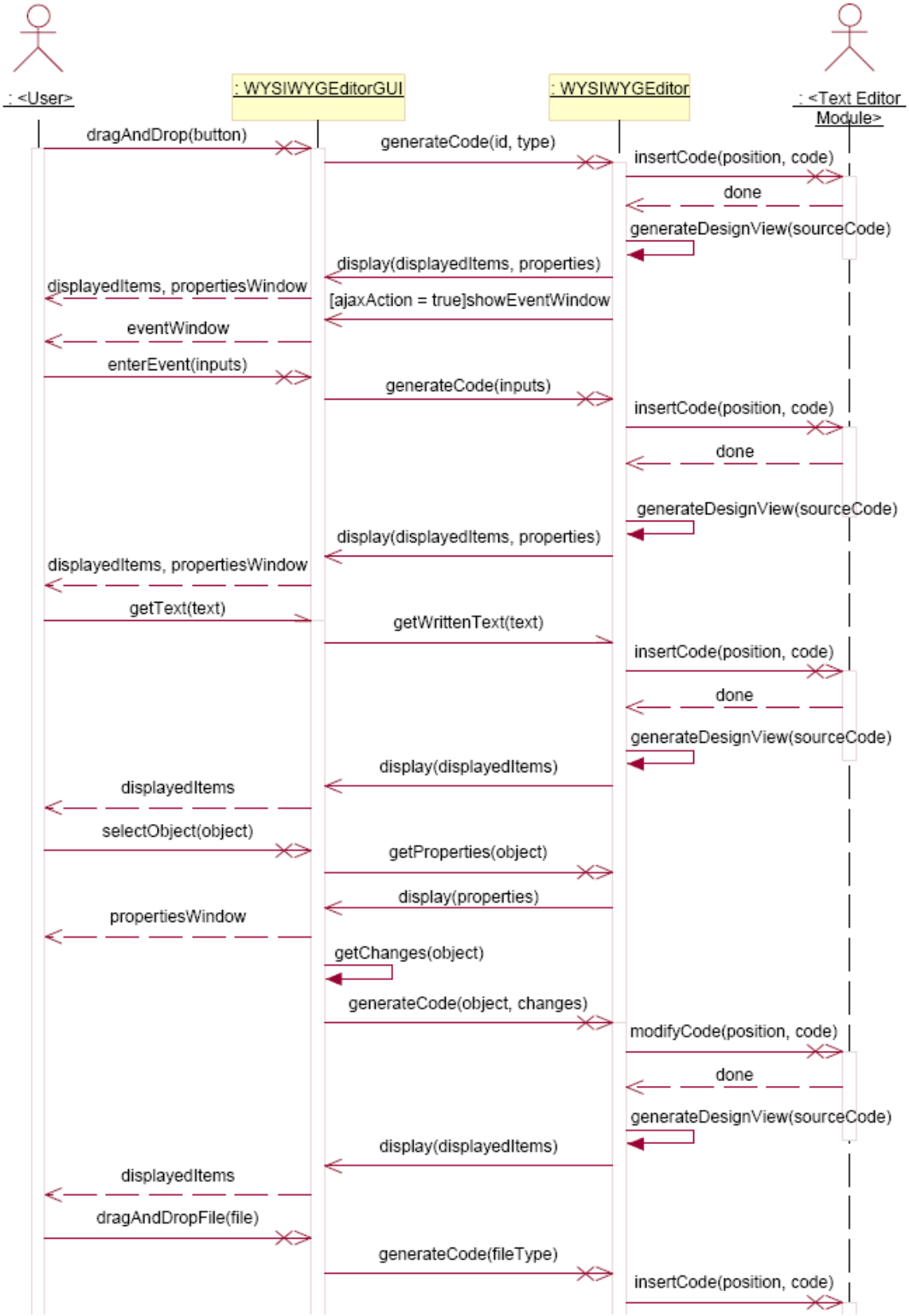
5.2 Sequence Diagrams

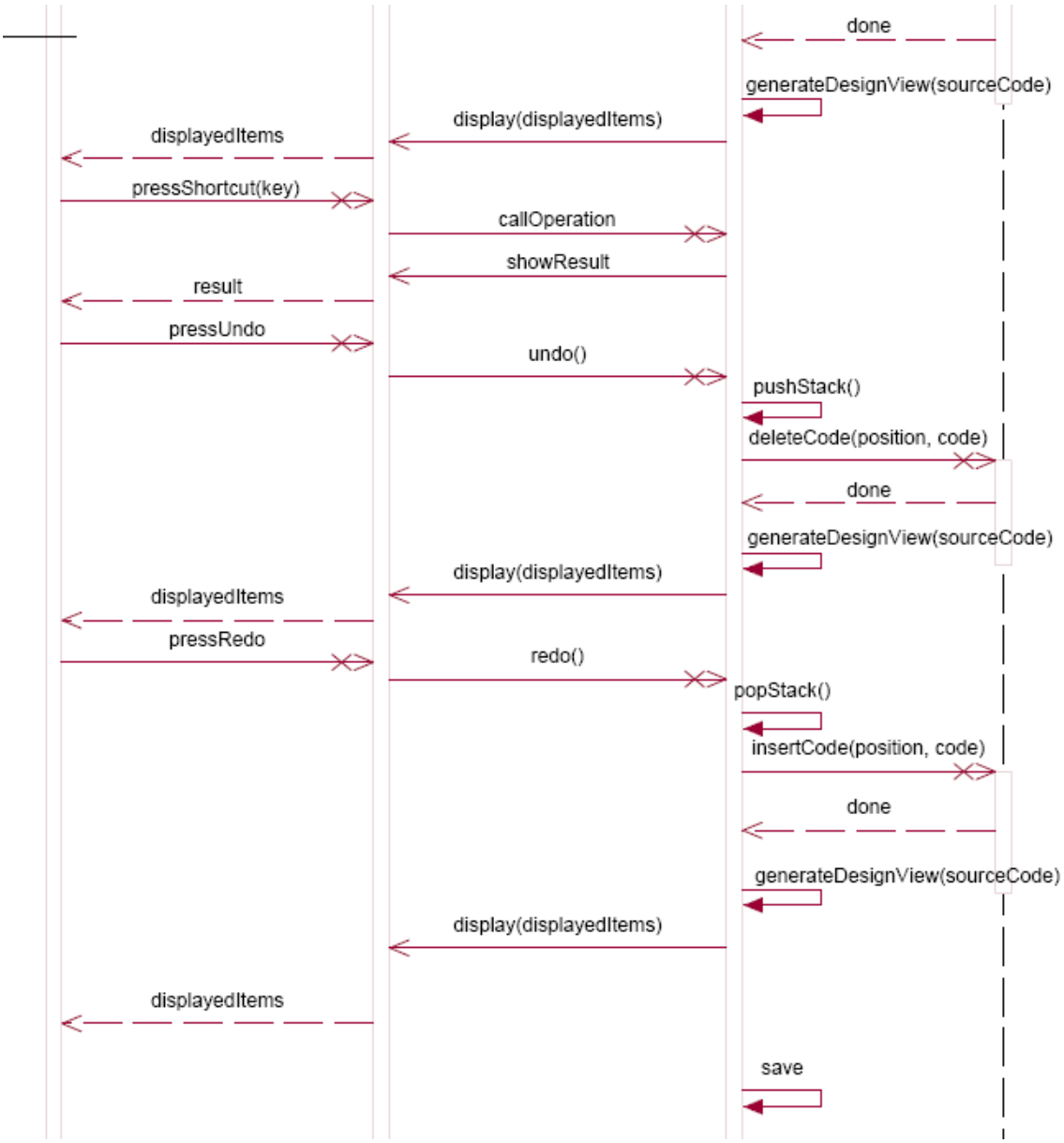
5.2.1 Text Editor



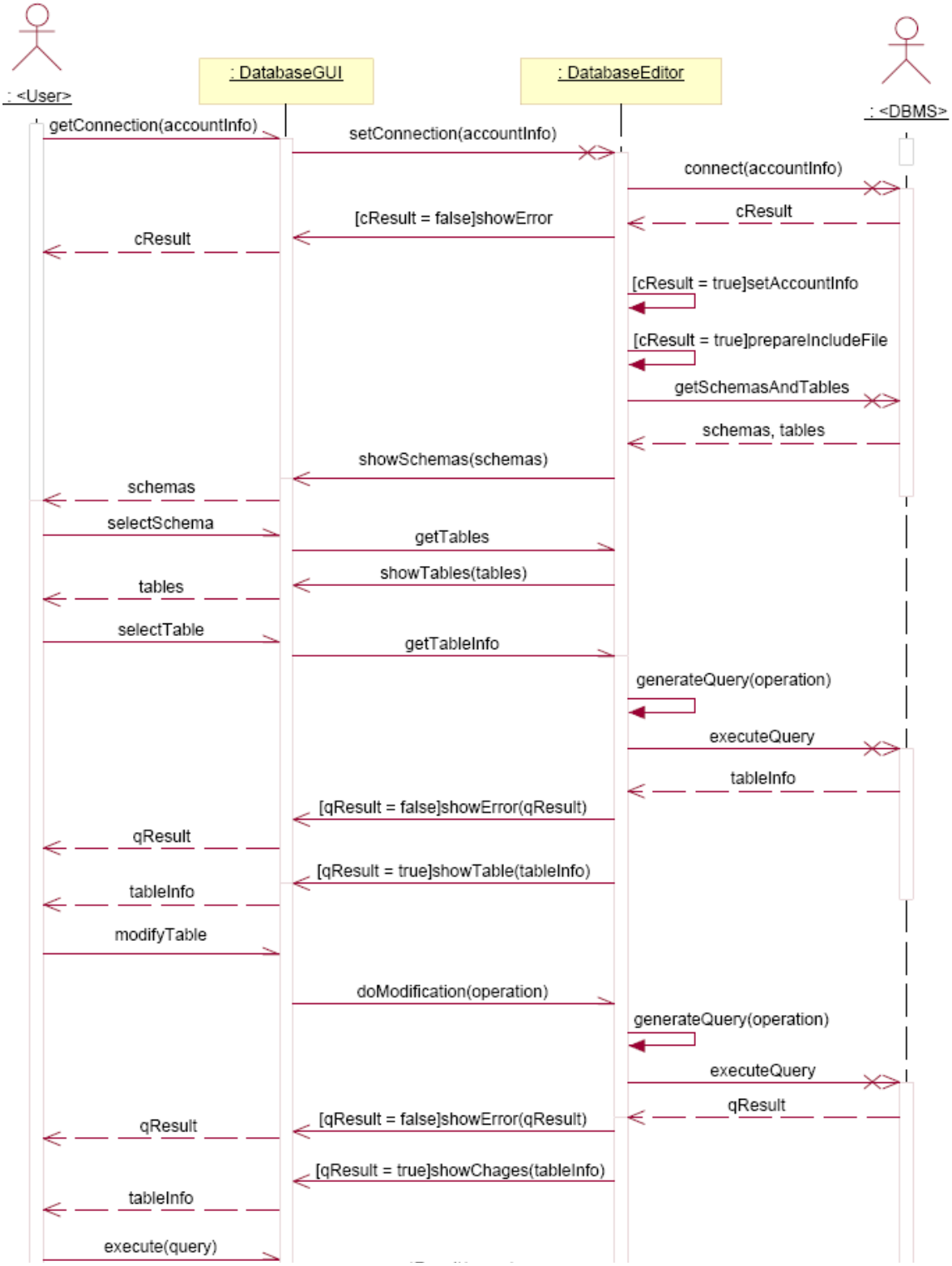

error
[searchString = true]showResult
matchedString
pressReplace(string1, string2)
replace(string1, string2)
replaceString(string1, string2)
[replaceString = false]showError
error
[replaceString = true]showResult
modifiedCode
moveToBracket
[code = braket]getMatch
findMatch(code)
showMatch(matchingBracket)
matchingBracket

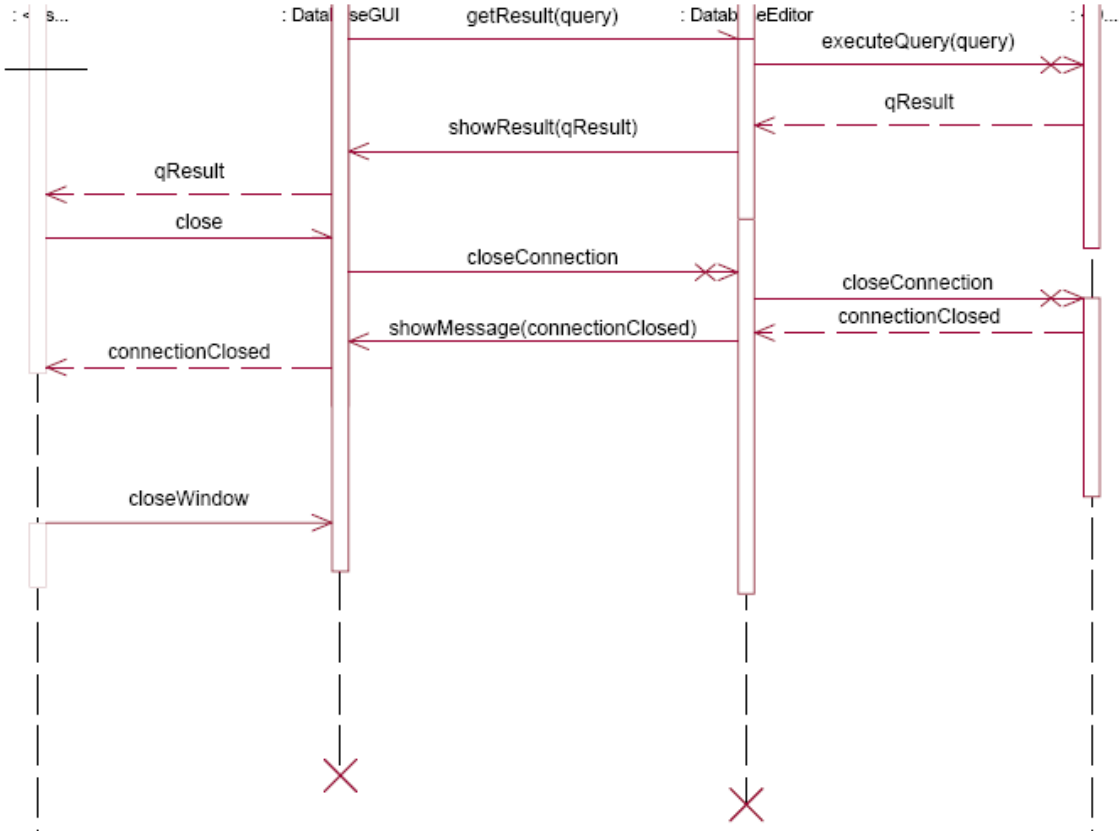
5.2.2 WYSIWYG Editor



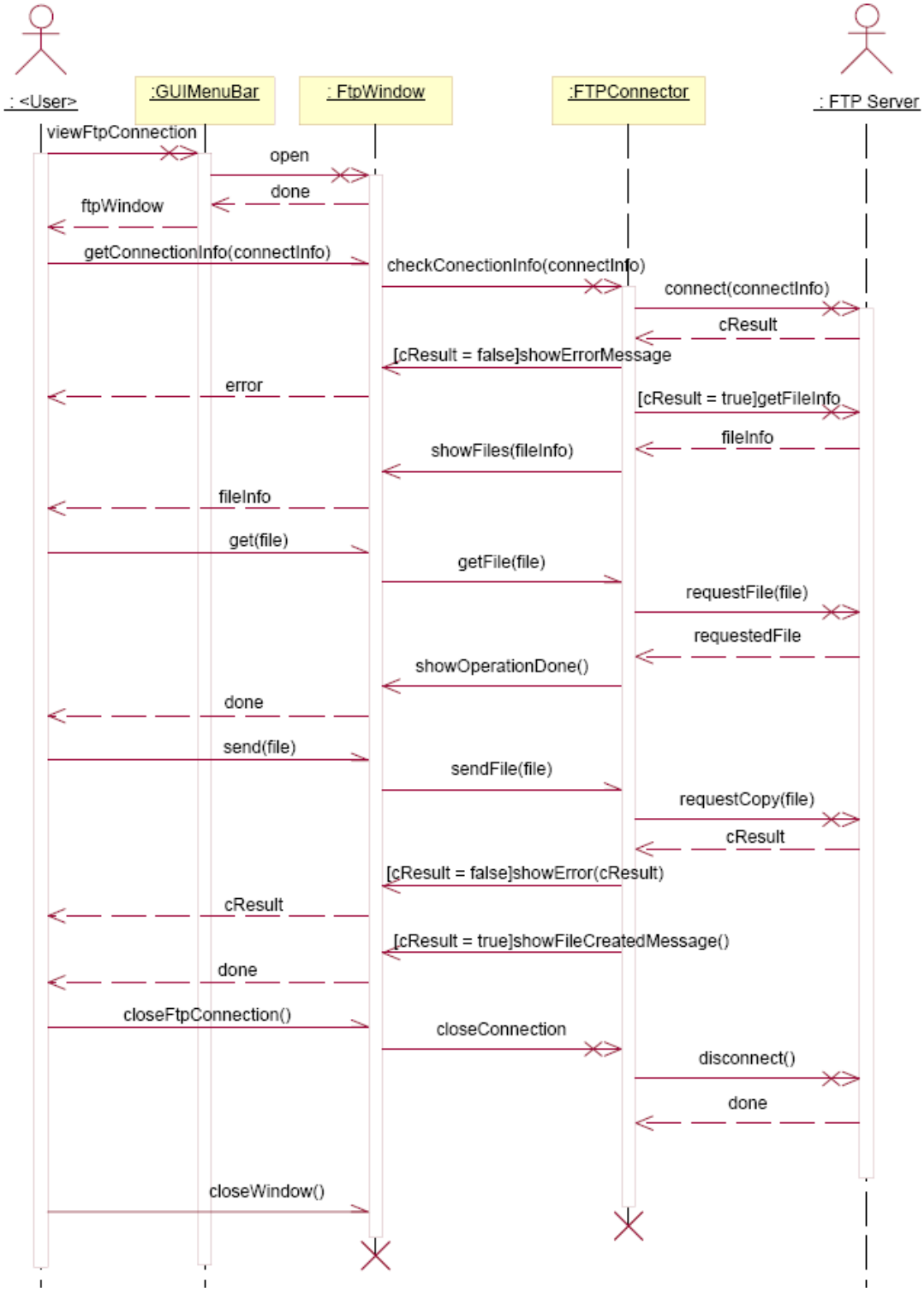

done
generateDesignView(sourceCode)
display(displayedItems)
displayedItems
pressShortcut(key)
callOperation
showResult
result
pressUndo
undo()
pushStack()
deleteCode(position, code)
done
generateDesignView(sourceCode)
display(displayedItems)
displayedItems
pressRedo
redo()
popStack()
insertCode(position, code)
done
generateDesignView(sourceCode)
display(displayedItems)
displayedItems
save

5.2.3 Database Editor

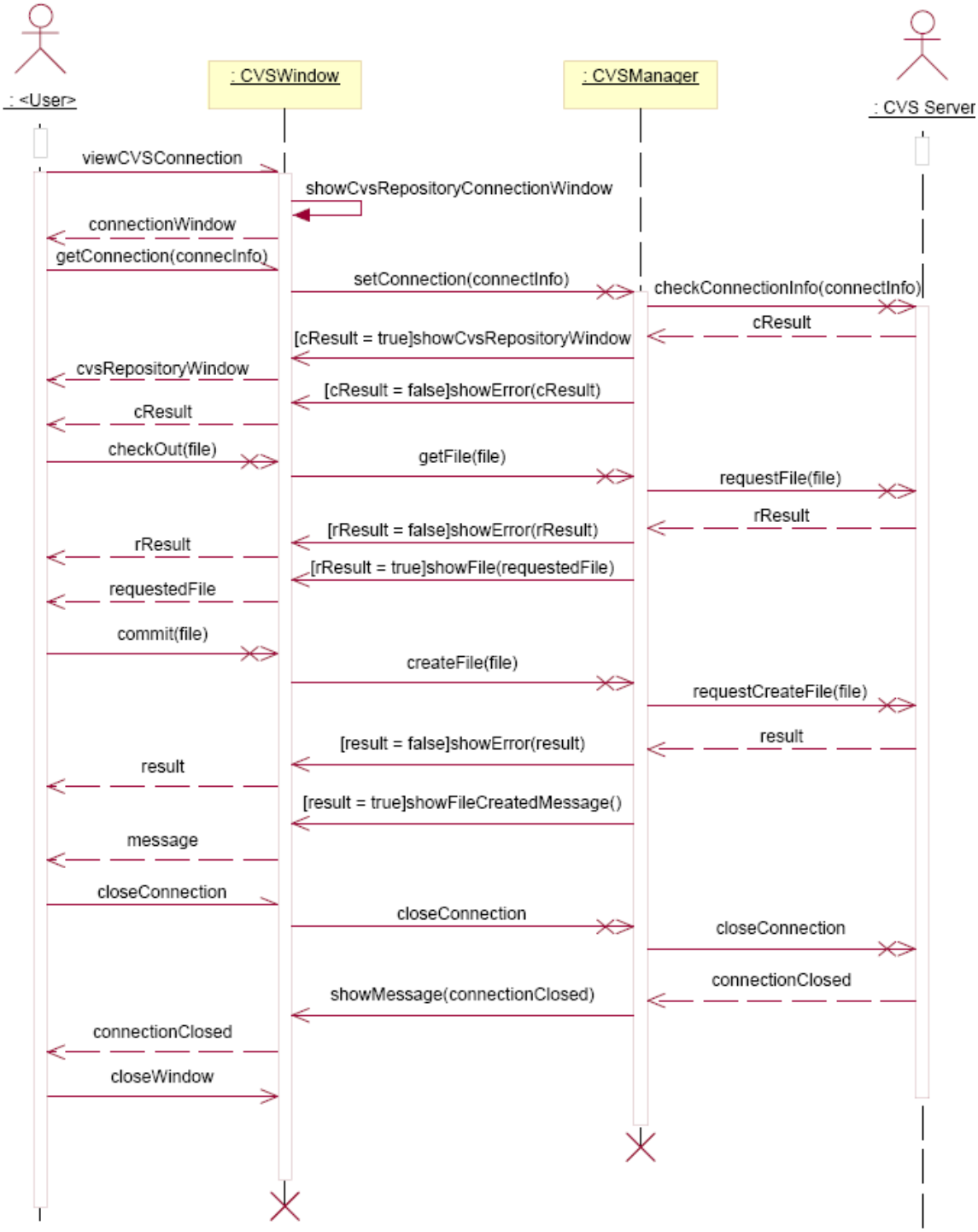



: < s...
: Data
seGUI
getResult(query)
: Datab
eEditor
executeQuery(query)
qResult
showResult(qResult)
qResult
close
closeConnection
closeConnection
connectionClosed
showMessage(connectionClosed)
connectionClosed
closeWindow

5.2.4 FTP Manager

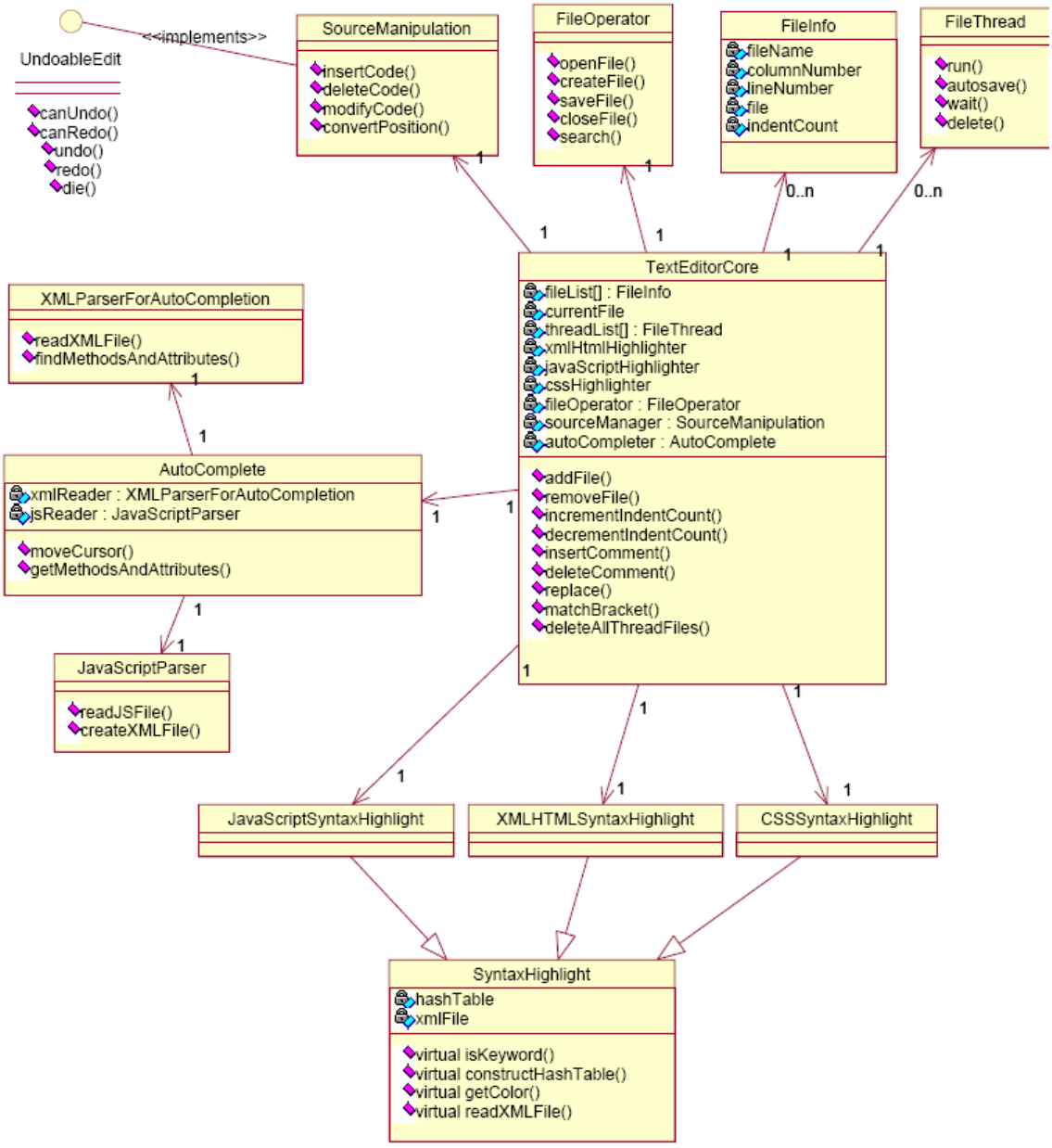


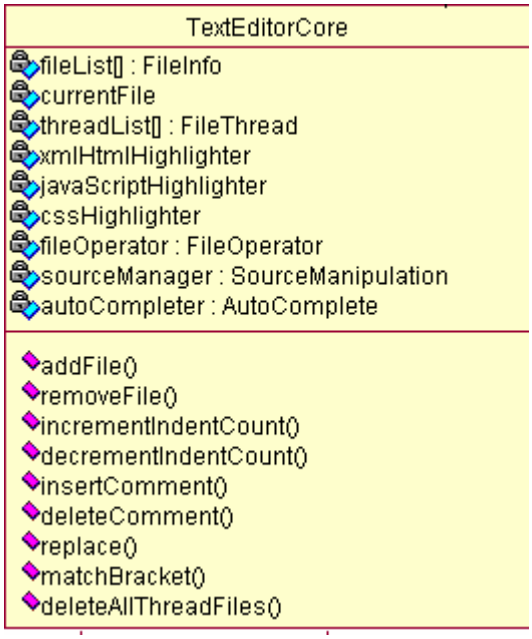
5.2.5 CVS Manager



5.3 Class Diagrams

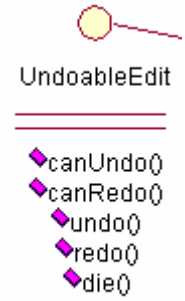
5.3.1 Text Editor

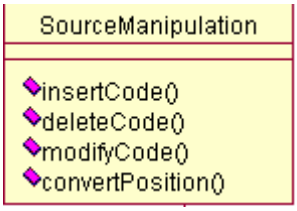


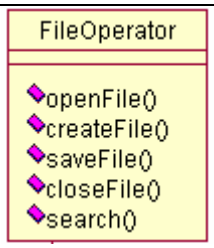


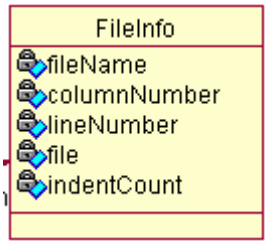
- **fileList[] : fileInfo** : Stores the information about the file and also the information needed to manipulate them.
- **currentFile :** Stores the identity of current file.
- **threadList[] : FileThread :** Stores the current list of the threads that work for functionalities of every file.
- **xmlHtmlHighlighter :** It is the instance of the XMLHTMLSyntaxHighlight class. It provides syntax highlighting for XML and HTML.
- **javaScriptHighlighter :** It is the instance of the javaScriptSyntaxHighlight class. It provides syntax highlighting for JavaScript.
- **cssHighlighter :** It is the instance of the cssSyntaxHighlight class. It provides syntax highlighting for css.
- **fileOperator: FileOperator :** It manages the file operations which will be discussed later.
- **sourceManager : SourceManipulation :** It manages the code writing operation which will be discussed later.
- **autoCompleter :** AutoComplete : It manages auto completion.
- **addFile() :** It receives input from GUI and adds a new file to the filelist.
- **removeFile() :** It receives input from GUI and removes a file from the filelist.
- **incrementIndentCount() :** Increments the indent number of the current file.
- **decrementIndentCount() :** Decrements the indent number of the current file.
- **insertComment() :** Inserts comment line to the beginning of

	every selected line. • **deleteComment() :** Deletes comment lines from the beginning of every selected line. • **replace() :** It replaces a selected word with the given word. • **matchBracket() :** It find the match bracket of a selected bracket. • **deleteAllThreadFiles() :** Removes all the threads.

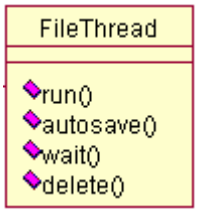
	This is an interface which Java already has. • **canUndo() :** Checks whether it is an undoable action. • **canRedo() :** Checks whether it is a redouble action. • **undo() :** Performs undo action. • **redo() :** Performs redo action.

	• **insertCode() :** Takes the code sent from WYSIWYG editor and inserts it to the correct position. • **deleteCode() :** Takes the code position sent from WYSIWYG editor and deletes it from the correct position. • **modifyCode() :** Takes the code and code position sent from WYSIWYG editor and inserts it to the correct position. • **convertPosition() :** Converts the cursor position of WYSIWYG editor to the cursor position of text editor.

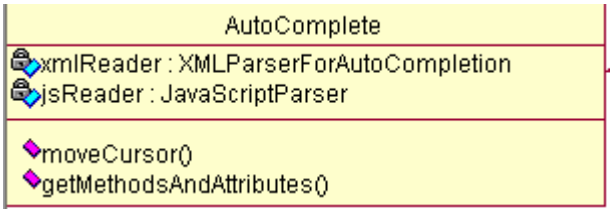
	• **openFile() :** It opens the file specified with GUI command. • **createFile() :** It creates the file specified with GUI command. • **saveFile() :** It saves the current file. • **closeFile() :** It closes the current file. • **search() :** It searches the user specified word or phrase.



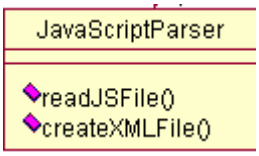
- **fileName :** It stores the name of the file.
- **columnNumber :** It stores the column number of the cursor position.
- **lineNumber :** It stores the row number of the cursor position.
- **file :** It stores a file pointer.
- **indentCount :** It stores the indent count number of the file.



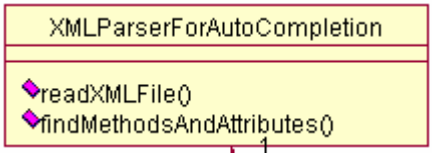
- **run() :** It starts the thread.
- **autosave() :** It saves the user changes to the temporary file created by the system.
- **wait() :** It tells the thread to wait.
- **delete() :** It removes the temporary file.



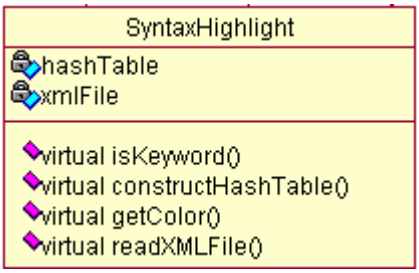
- **xmlReader :** It reads the XML files for auto completion.
- **jsReader :** It reads the JavaScript files for auto completion.
- **moveCursor() :** It moves the cursor after inserting an attribute and a command for a JavaScript Class.
- **getMethodsAndAttributes() :** It gets the methods and attributes which belong to the class user is currently dealing with.



- **readJSFile() :** It reads the javascript file the user has created.
- **createCMLFile() :** It creates an XML file for storing the class, method and attribute information of a user defined JavaScript class.



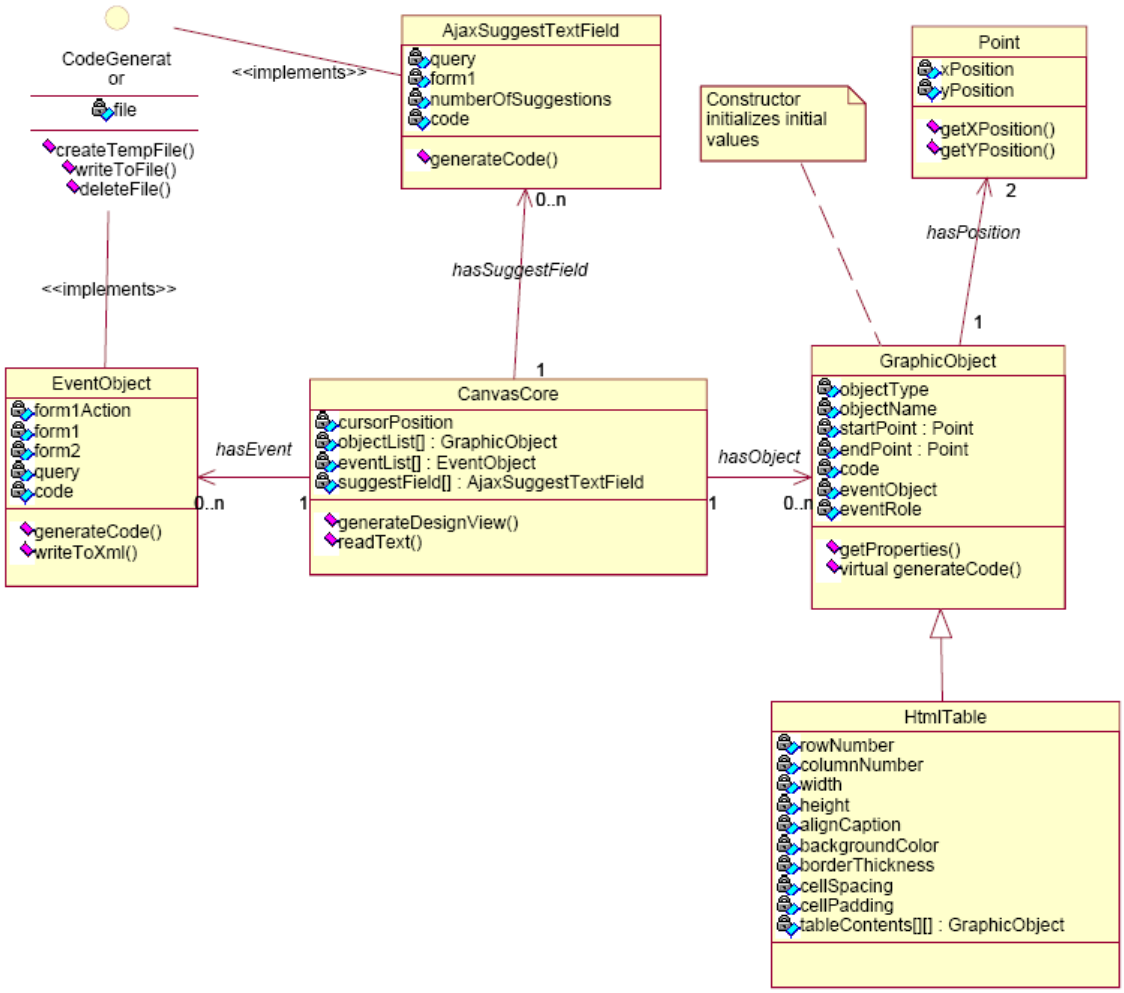
- **readXMLFile() :** It reads the XML files that are created by a JavaScriptParser object.
- **findMethodsAndAttributes() :** It find the methods and attributes of a JavaScript function.

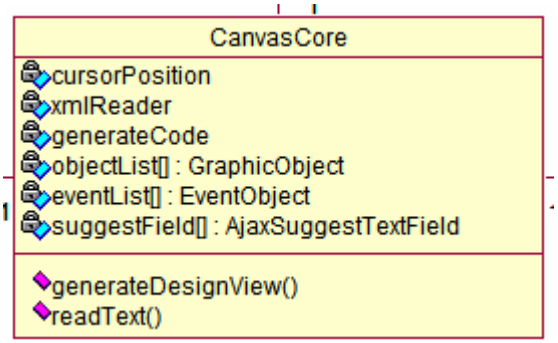


This class is a base class for JavaScriptSyntaxHighlight, XMLHTMLSyntaxHighlight, CSSSyntaxHighlight classes.

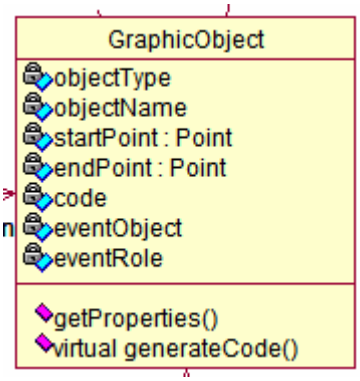
- **hasTable :** It is the hash table of a programming language which stores the keywords and their predefined colors.
- **xmlFile :** It reads the XML files to insert the keywords to the hash table.
- **isKeyword() :** It checks whether the written word is a keyword or not.
- **getColor() :** It gets the color of the keyword which the user has entered from the hash table.
- **readXMLFile() :** It reads the XML file to find the keywords of the current file type.

5.3.2 WYSIWYG Editor

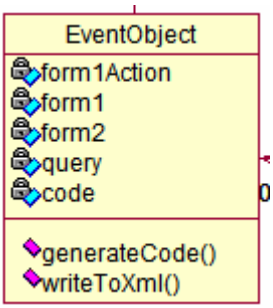




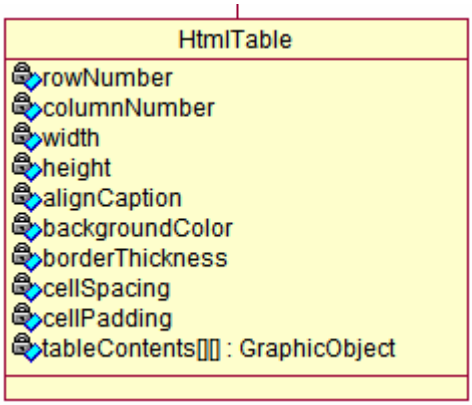
- **CanvasCorecursorPosition :** Stores the current position of cursor.
- **objectList[]**: The List of GraphicalObject class instances.
- **eventList[]:**The List of EventObject class instances.
- **suggestField[]:**The List of AjaxSuggestTextField class instances.
- **generateDesignView() :** Read the source code and generate Design view.
- **readText():** get the text input from user.



- **objectType :** Stores the type of object
- **objectName:** Stores the name of object
- **startPoint:** Stores the start point coordinates of object
- **endPoint:** Stores the end point coordinates object
- **code:** Stores related code for generating the Design view of object
- **eventObject:** if object is an ajax action stores the eventObject of ajax action
- **eventRole:** if object is an ajax action stores the role of ajax action
- **getProperties():** return the properties of Graphical Object
- **virtual generateCode():** virtual function for creating code according to the properties.



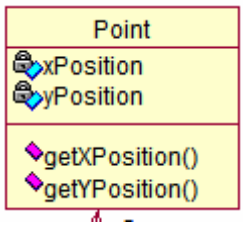
- **form1Action:** Stores the action type for ajax action
- **form1:** Stores the info of first related form
- **form2:** Stores the info of second related form
- **query:** Stores desired query for custom ajax action
- **code:** Stores code for custom ajax action
- **generateCode():** generates code for required for that custom ajax action.
- **writeToXml():** writes the required information of custom ajax action for reuse.



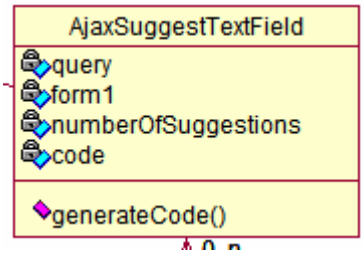
- **rowNumber:** Stores the row number of html table
- **columnNumber:** Stores the column number of html table
- **width:** Stores the row number of html table
- **height:** Stores the row number of html table
- **alignCaption:** Stores the align info of html table
- **backgroundColor:** Stores the bg color info of html table
- **borderThickness:** Stores the border thickness info of html table
- **cellSpacing:** Stores the cell spacing info

of html table

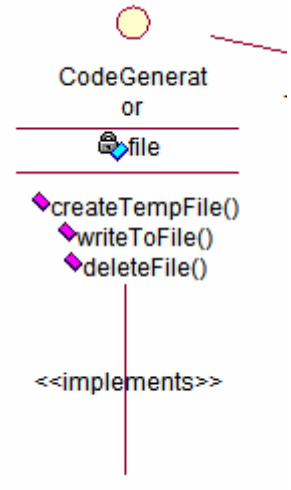
- **cellPadding:** Stores the cell padding info of html table
- **tableContents[][]:**Stores the contents of rows and column



- **xPosition**: stores the line number information as x point
- **yPosition**: stores the character information as x point
- **getXpositon():** returns the xPosition
- **getYposition():**returns the yPosition

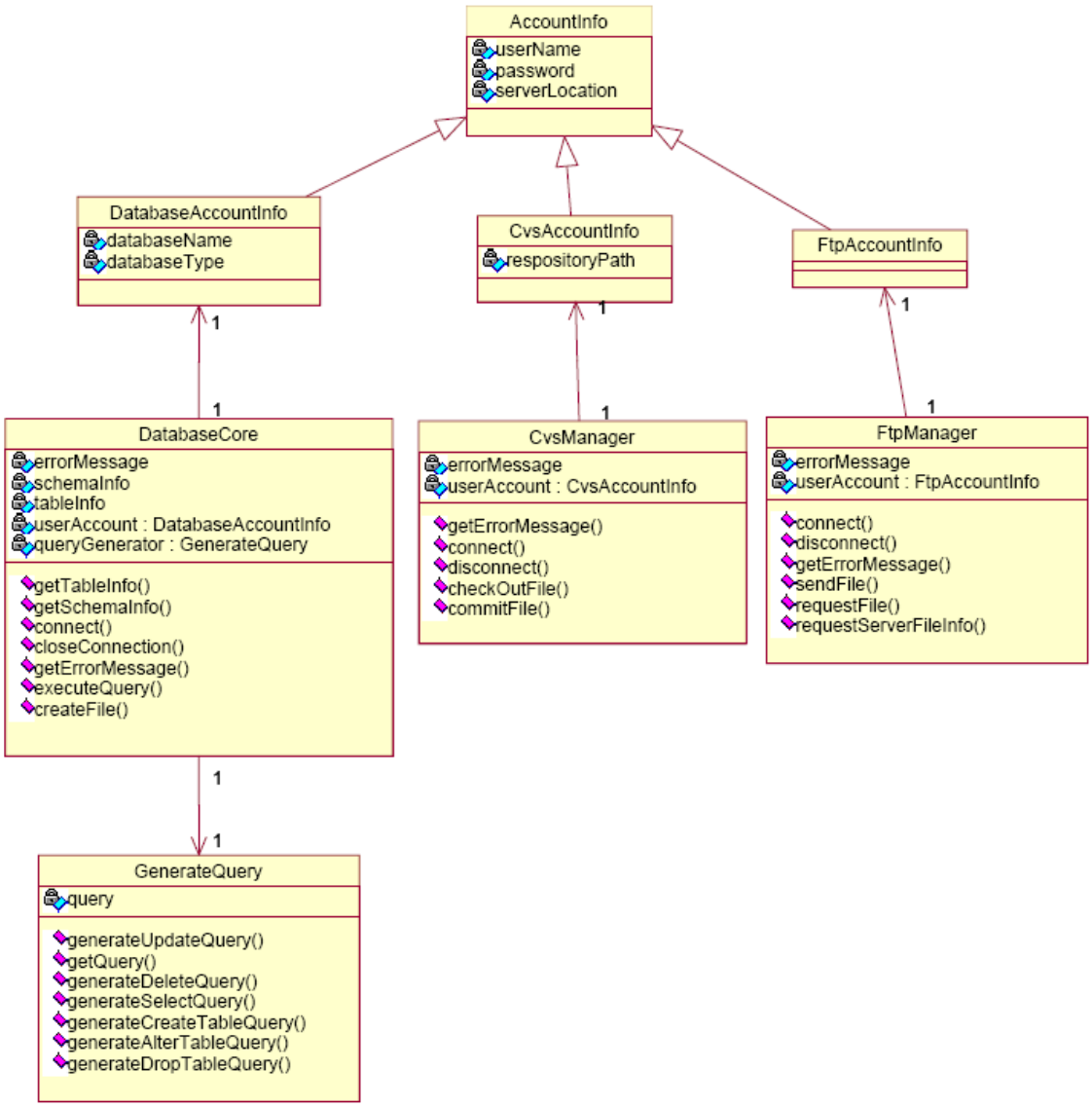


- **query** : It stores the sql query which the user has entered to use the AJAX application.
- **form1** : It stores the information of which element the suggestion will show.
- **numberOfSuggestions** : It stores the number of suggestions.
- **code** : It stores the JavaScript code related to the AJAX action.
- **generateCode()** : It generates the necessary code for the action.



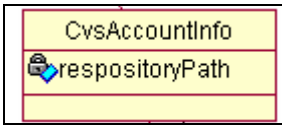
- **createTempFile():** creates a temporary file for sending codes to Text Editor
- **writeToFile():** writes the codes
- **deleteFile():** delete the temporary file after Text Editor gets the codes.

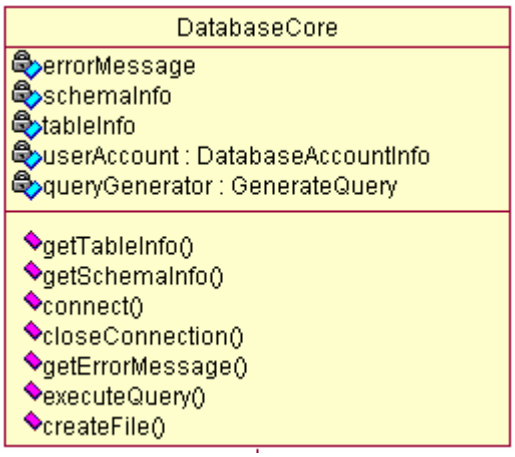
5.3.3 Database Editor, CVS Manager and FTP Manager

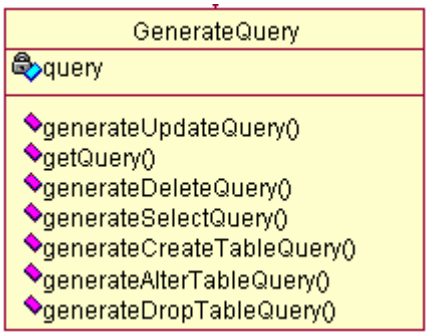


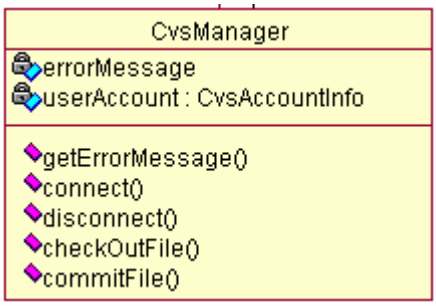
AccountInfo userName password serverLocation	• **userName** : Stores username for connection. • **password** : Stores password for connection. • **serverLocation** : Stores the location of the server.

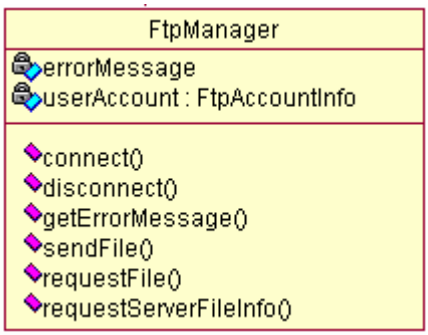
DatabaseAccountInfo databaseName databaseType	• **databaseName** : Stores the name of the database that is connected to. • **databaseType** : Stores the type of the database (MySQL or Oracle) that is connected to.

	• **respositoryPath**: Stores the path of the reprository in the CVS server.

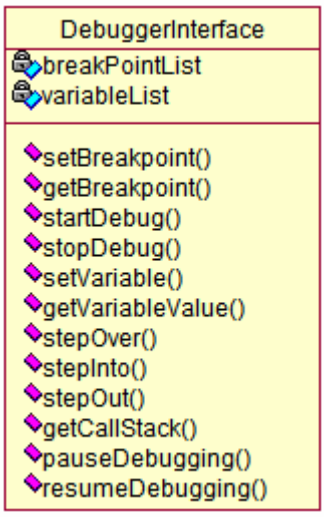
	• **errorMessage :** Stores the error message that is shown to user. • **schemaInfo :** Stores the schema info of the database that is connected to. • **tableInfo :** Stores the table info of a selected schema. • **userAccount :** DatabaseAccountInfo : Stores the account information for a database. • **queryGenerator :** GenerateQuery : Instance of QueryGenerator class which is responsible for generating queries. • **getTableInfo :** Returns tableInfo. • **getSchemaInfo :** Returns schemaInfo. • **connect :** Connects to a database specified by userAccount. • **closeConnection :** Disconnects from database. • **getErrorMessage :** Returns error. • **executeQuery :** Sends a query to DBMS. • **createFile :** Prepares an include file with database account information.

	• **query :** Stores the query generated. • **getQuery :** Returns the query. • **generateUpdateQuery :** Generates an update query. • **generateDeleteQuery :** Generates a delete query. • **generateSelectQuery :** Generates a select query. • **generateCreateTableQuery :** Generates a create table query. • **generateAlterTableQuery :** Generates an alter table query. • **generateDropTableQuery :** Generates a drop table query.

	• **errorMessage :** Stores the error message that is shown to user. • **userAccount :** Stores the user account information to connect to CVS server. • **getErrorMessage :** Returns the error message. • **connect :** Connects to CVS server. • **disconnect :** Disconnects to CVS server. • **commitFile :** Commits a specified file to CVS server. • **checkOutFile :** Requests the specified file from CVS server.

	• **errorMessage :** Stores the error message that is shown the user. • **userAccount :** Stores the account information to connect to an FTP server. • **connect :** Connects to an FTP server. • **disconnect :** Disconnects from an FTP server. • **getErrorMessage :** Returns an error message. • **sendFile :** Sends a file via ftp. • **requestFile :** Requests a file via ftp. • **requestServerFileInfo :** Gets the file information of server.

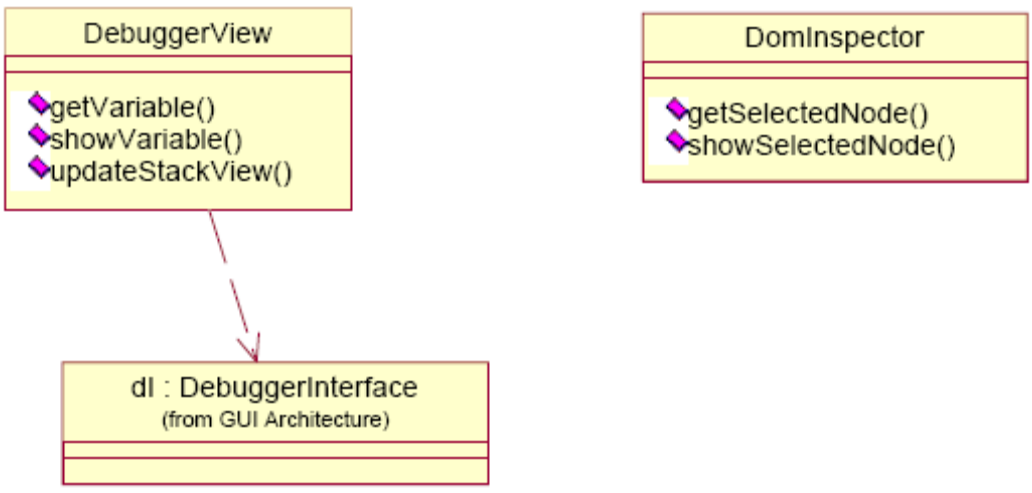
5.3.4 Debugger

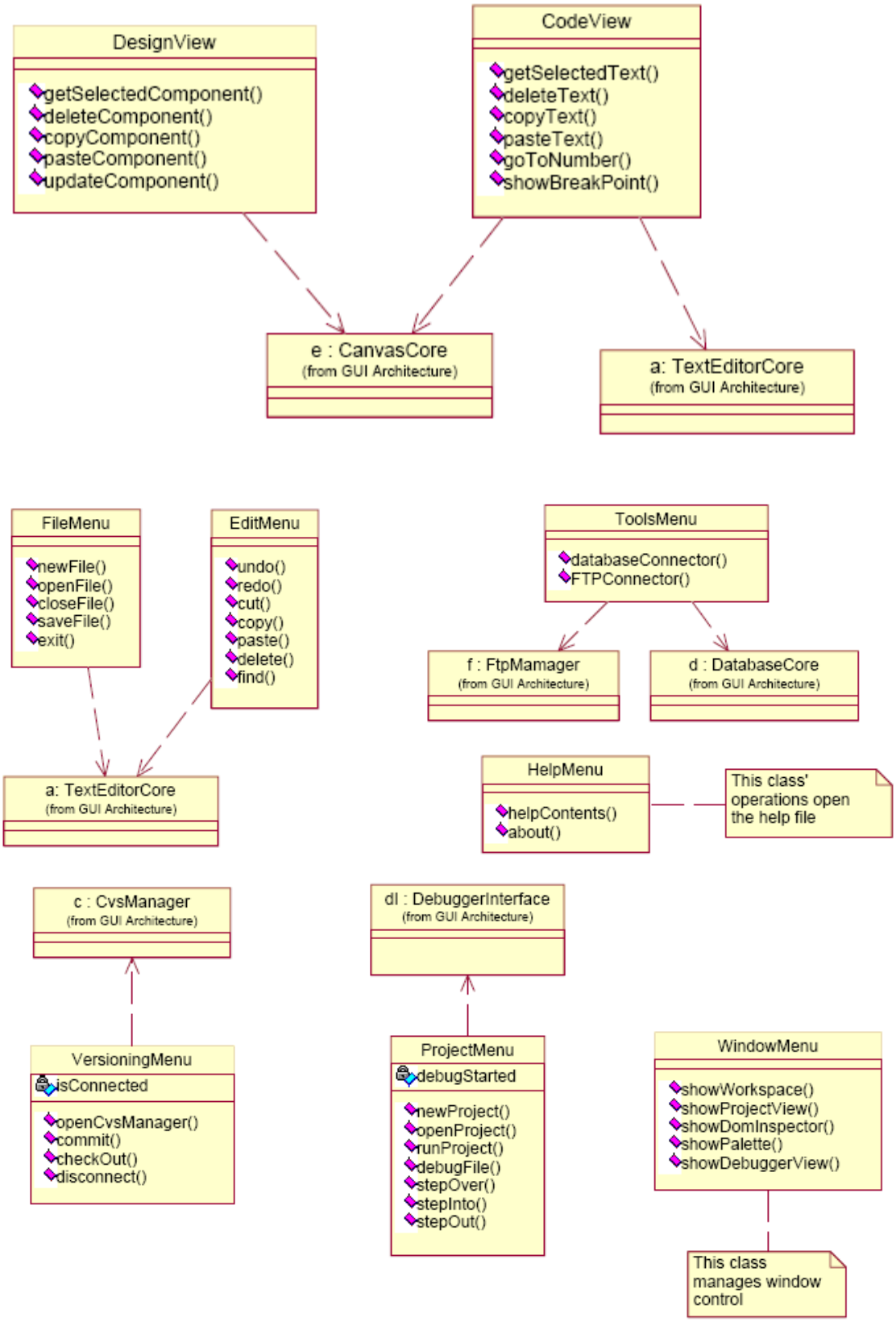


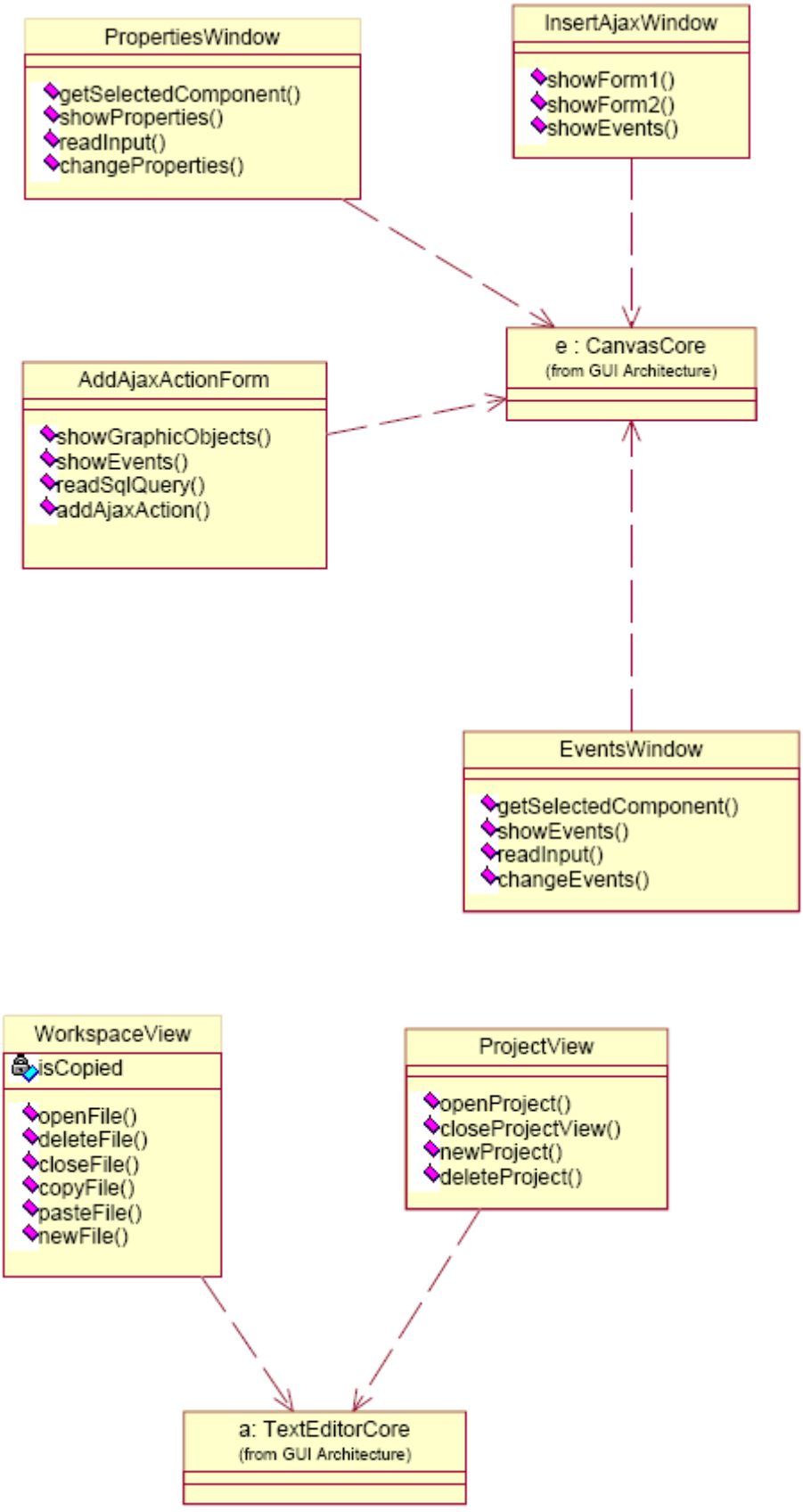
- **breakPointList** = It stores breakpoints locations in a list.
- **variableList** = It stores variables in a list

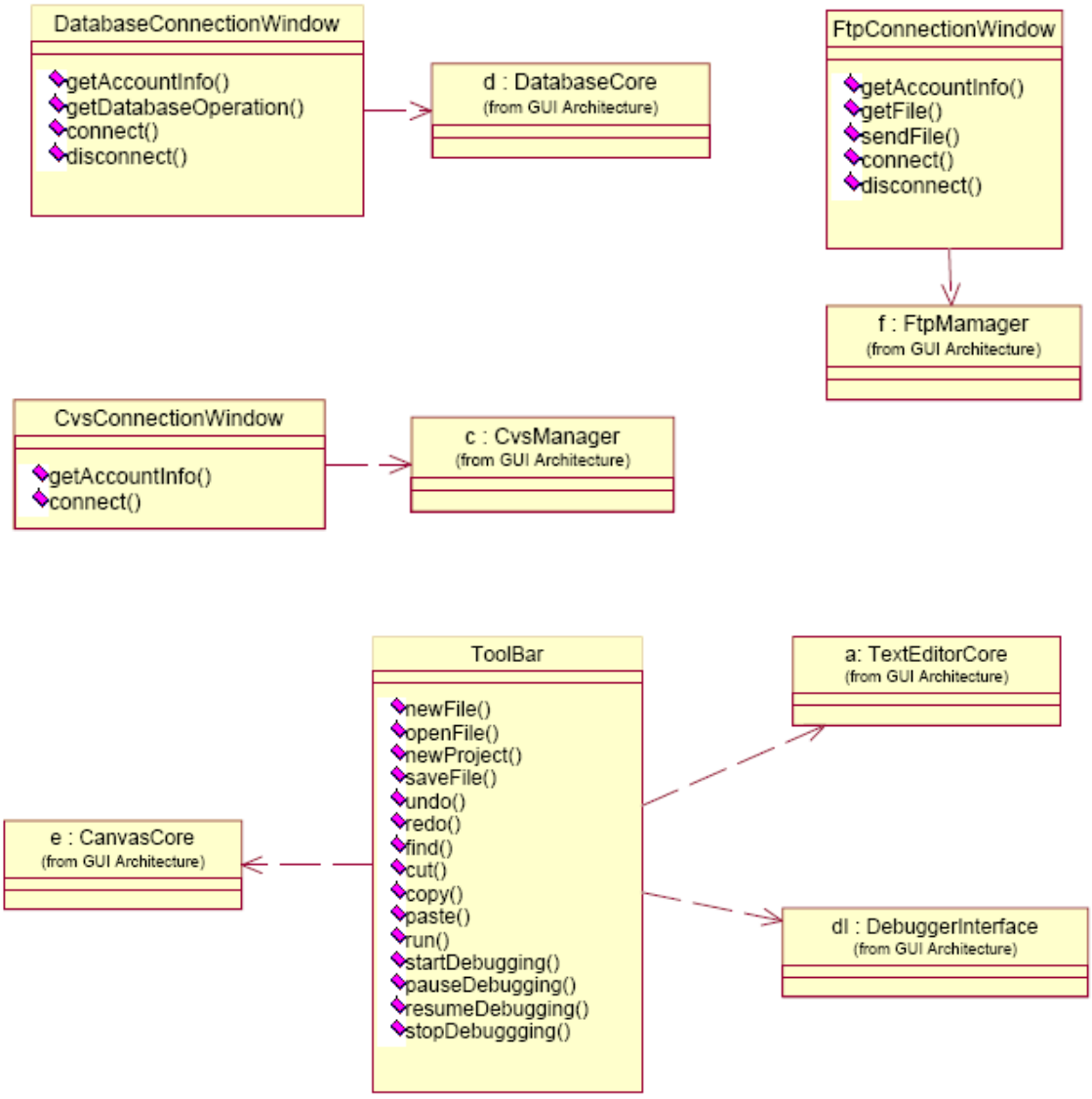
- **setBreakpoint()** = sets the breakpoint location.
- **getBreakpoint()** = gets the breakpoint location.
- **startDebug()** = it starts debugging.
- **stopDebug()** = it stops debugging.
- **setVariable()** = sets the variable that user wants to trace.
- **getVariableValue()** = gets current value of the variable.
- **stepOver()** = steps over the breakpoint while debugging.
- **stepInto()** = step into the breakpoint while debugging.
- **stepOut()** = step out the breakpoint while debugging.
- **getCallStack()** = gets current call stack of the debugging program.
- **pauseDebugging()** = it pauses debugging.
- **resumeDebugging()** = it resumes debugging.

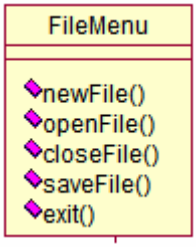
5.3.5 GUI



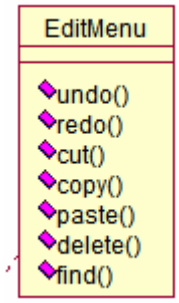

DesignView
getSelectedComponent()
deleteComponent()
copyComponent()
pasteComponent()
updateComponent()
CodeView
getSelectedText()
deleteText()
copyText()
pasteText()
goToNumber()
showBreakPoint()
e : CanvasCore
(from GUI Architecture)
a: TextEditorCore
(from GUI Architecture)
FileMenu
newFile()
openFile()
closeFile()
saveFile()
exit()
EditMenu
undo()
redo()
cut()
copy()
paste()
delete()
find()
ToolsMenu
databaseConnector()
FTPConnector()
f : FtpMamager
(from GUI Architecture)
d : DatabaseCore
(from GUI Architecture)
a: TextEditorCore
(from GUI Architecture)
HelpMenu
helpContents()
about()
This class' operations open the help file
c : CvsManager
(from GUI Architecture)
dl : DebuggerInterface
(from GUI Architecture)
VersioningMenu
isConnected
openCvsManager()
commit()
checkOut()
disconnect()
ProjectMenu
debugStarted
newProject()
openProject()
runProject()
debugFile()
stepOver()
stepInto()
stepOut()
WindowMenu
showWorkspace()
showProjectView()
showDomInspector()
showPalette()
showDebuggerView()
This class manages window control


PropertiesWindow
getSelectedComponent()
showProperties()
readInput()
changeProperties()
InsertAjaxWindow
showForm1()
showForm2()
showEvents()
e : CanvasCore
(from GUI Architecture)
AddAjaxActionForm
showGraphicObjects()
showEvents()
readSqlQuery()
addAjaxAction()
EventsWindow
getSelectedComponent()
showEvents()
readInput()
changeEvents()
WorkspaceView
isCopied
openFile()
deleteFile()
closeFile()
copyFile()
pasteFile()
newFile()
ProjectView
openProject()
closeProjectView()
newProject()
deleteProject()
a: TextEditorCore
(from GUI Architecture)

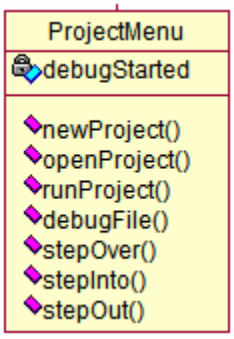




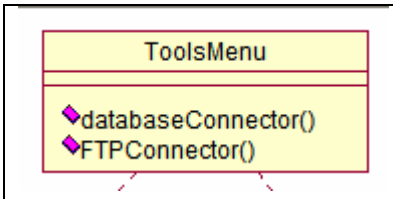
- **newFile()** : Creates a new file and invokes TextEditorCore.
- **openFile()** : Invokes TextEditorCore to open a file.
- **closeFile()** : Invokes TextEditorCore to close a file.
- **saveFile()** : Invokes TextEditorCore to save a file.
- **exit()** : Invoke exit function of system.



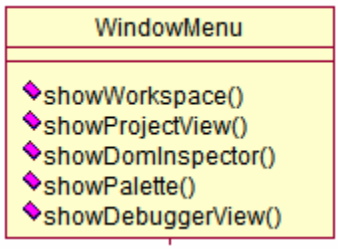
- **undo()** : Invokes Text Editor's related function.
- **redo()** : Invokes Text Editor's related function.
- **cut()** : Invokes Text Editor's related function.
- **copy()** : Invokes Text Editor's related function.
- **paste()** : Invokes Text Editor's related function.
- **delete()** : Invokes Text Editor's related function.
- **find()** : Invokes Text Editor's related function.



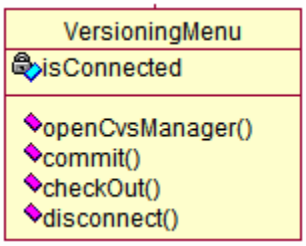
- **newProject()** : Creates a new project, invokes related function.
- **openProject ()** : Invokes related function to open a project.
- **runProject ()** : Invokes related function to run project.
- **debugProject()** : Invokes DebuggerInterface to debug.
- **stepOver() :** Invokes DebuggerInterface to step over.
- **stepInto() :** Invokes DebuggerInterface to step over.
- **stepOut() :** Invokes DebuggerInterface to step over.

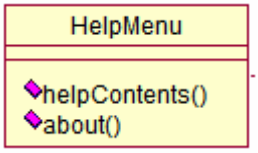


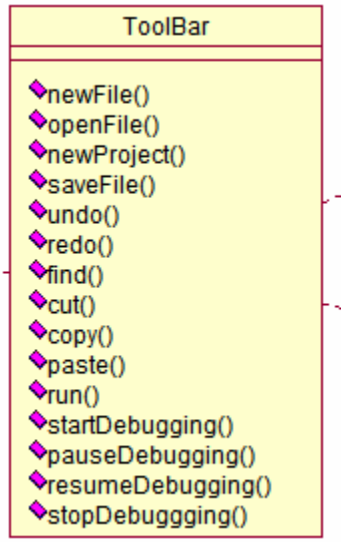
- **databaseConnector()** : Shows Database connection console and invoke DatabaseCore
- **FTPConnector()** : Shows FTP connection console and invoke FTPManager

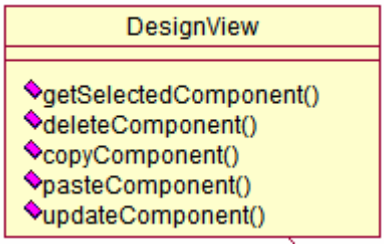


- **showWorkspace()** : Shows Workspace view of GUI.
- **showProject ()** : Shows Project view of GUI.
- **showDomInspector ()** : Shows DomInspector view of GUI.
- **showPalette ()** : Shows Palette view of GUI.
- **showDebugger ()** : Shows Debugger view of GUI.

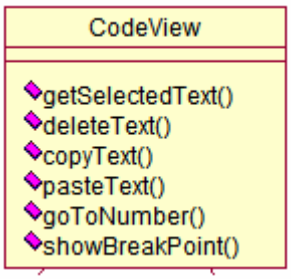
	• **isConnected** : Stores connection information. • **openCvsManager() :** Invokes CVS Manager to open CVS window • **commit() :** Invokes CVS Manager to commit a file. • **checkOut() :** Invokes CVS Manager to check out a file. • **disconnect() :** Invokes CVS Manager to close CVS connection.

	• **helpContents()** : Shows help contents. • **about()** : Shows information about IDE.

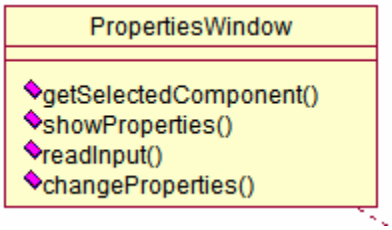
	• **newFile()** : Creates a new file and invokes TextEditorCore. • **openFile()** : Invokes TextEditorCore to open a file. • **newProject()** : Creates a new project and invokes related function. • **saveFile()** : Invokes TextEditorCore to save a file. • **undo()** : Invokes Text Editor's related function. • **redo()** : Invokes Text Editor's related function. • **find()** : Invokes Text Editor's related function. • **cut()** : Invokes Text Editor's related function. • **copy()** : Invokes Text Editor's related function. • **paste()** : Invokes Text Editor's related function. • **delete()** : Invokes Text Editor's related function. • **run()** : Invokes related function to run project. • **startDebugging()** : Invokes DebuggerInterface to start debugging. • **pauseDebugging()** : Invokes DebuggerInterface to pause debugging. • **resumeDebugging()** : Invokes DebuggerInterface to resume debugging. • **stopDebugging()** : Invokes DebuggerInterface to stop debugging.



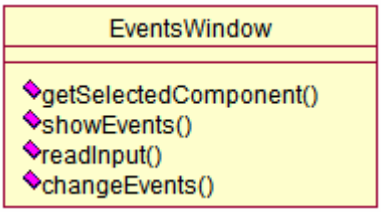
- **getSelectedComponent()** : send selected component to CanvasCore.
- **deleteComponent()** : Invokes CanvasCore to delete a component.
- **copyComponent()** : Invokes CanvasCore to copy a component.
- **pasteComponent()** : Invokes CanvasCore to paste a component.
- **updateComponent()** : Invokes CanvasCore to update a component.



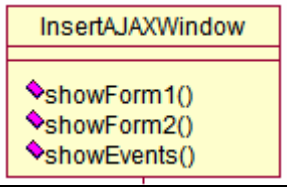
- **getSelectedText()** : send selected text to TextEditorCore.
- **deleteText ()** : Invokes TextEditorCore to delete text.
- **copyText ()** : Invokes TextEditorCore to copy text.
- **pasteText ()** : Invokes TextEditorCore to paste text.
- **goToNumber()** : Invokes TextEditorCore to go selected line.
- **showBreakPoint()** : shows breakpoints of Debugger.



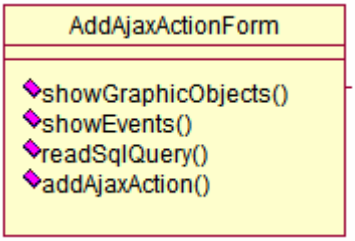
- **getSelectedComponent()** : send selected component to CanvasCore.
- **showProperties()** : show properties of component.
- **readInput()** : gets input from user.
- **changeProperties()** : update properties of selected component.



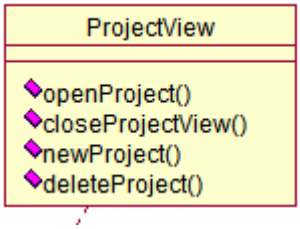
- **getSelectedComponent()** : send selected component to CanvasCore.
- **showEvents()** : show events of selected component.
- **readInput()** : gets input from user.
- **changeEvents()** : update events of selected component.



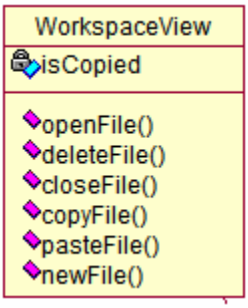
- **showForm1()** : shows the form to insert object1.
- **showForm2()** : shows the form to insert object2.
- **showEvents()** : shows events of inserted objects.



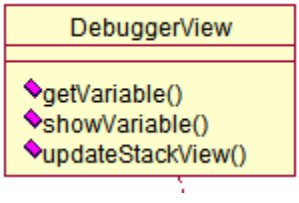
- **showGraphicObjects()** : shows the list of objects.
- **showEvents()** : shows events of inserted objects.
- **readSqlQuery()** : gets the SQL query input of user.
- **addAjaxAction()** : invokes system to add new AJAX object.



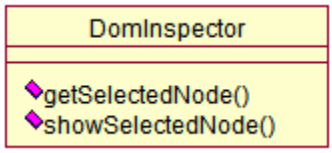
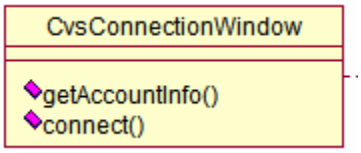
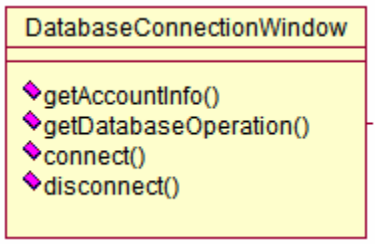
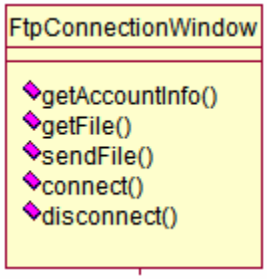
- **openProject ()** : Invokes related function to open a project.
- **closeProject ()** : Invokes related function to close a project.
- **newProject()** : Creates a new project and invokes related function.
- **deleteProject ()** : Invokes related function to delete project.



- **openFile()** : Invokes TextEditorCore to open a file.
- **deleteFile()** : Invokes TextEditorCore to delete a file.
- **closeFile()** : Invokes TextEditorCore to close a file.
- **copyFile()** : Invokes TextEditorCore to copy a file.
- **pasteFile()** : Invokes TextEditorCore to paste a file.
- **newFile()** : Creates a new file and invokes TextEditorCore.



- **getVariable()** : gets the entered varible information
- **showVariable()** : shows information of variable.
- **updateStackView()** : Invoke DebuggerInterface to update program stack.

	• **getSelectedNode()** : gets the node information • **showVariable()** : shows information of selected node.
	• **getAccountInfo()** : gets input of connection information from user. • **connect()** : Invokes CVSManager to connect.
	• **getAccountInfo()** : gets input of connection information from user. • **getDatabaseOperation()** : gets input of operation from user. • **connect()** : Invokes DatabaseCore to connect. • **disconnect()** : Invokes DatabaseCore to disconnect.
	• **getAccountInfo()** : gets input of connection information from user. • **getFile()** : gets file information from user and invokes FtpManager to get file. • **sendFile()** : gets file information from user and invokes FtpManager to send file.. • **connect()** : Invokes FtpManager to connect. • **disconnect()** : Invokes FtpManager to disconnect.

6. GUI DESIGN

6.1. Overview of GUI

AJAXDEV project has to provide developers a user-friendly environment which they can create interactive and rich web applications especially using AJAX actions. “GUI Design” module of our project is one of the most important parts because it provides the permanent interaction of user with Development Environment. We will design 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 GUI functional requirements mainly in “Requirement Analysis Report”. We revised and made some changes about our GUI to provide users more usability.

Consequently, we have started to design and implement GUI of our development environment. As we decided to implement our project by using JAVA, we have used “JAVA Swing” package while implementing GUI.

6.2 GUI Requirements

- User will be able to 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 “Cut”, “Copy”, “Paste”, “Delete”, “Select All” 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 “Cut”, “Copy”, “Paste”, “Delete” and “Select All” actions.

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

- User will able to 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.

- 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.
  - If user will click any cell of “Properties” table, that cell will be ready to edit or update, entered text can be the name of component or column number of a table.
  - User will define his/her component’s events (handlers, actions) by using “Events” table.
  - If user will click any cell of “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 tabs which are “Stack” and “Watch” views.
    - In “Stack” view, user will be able to see variables and functions currently placed in program stack.
    - In “Watch” 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 “Connect Database” button and connects his/her database without any problem.
    - When user clicks “Connect Database” button, an input dialog will open and will get information of user’s account name, password, location of database and type of database (MySQL or Oracle).
    - If request is accepted by DBMS, “Database” view will be shown to user to interact with his/her database.
    - If request is denied system will show an error message and request account information again.
    - After user connects to a database, schemas in that database will be shown to user. User can select a schema among the list.
    - 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.
    - User will be able 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.
    - User will 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 “Update” icon, or delete it by clicking “Delete” icon.
- User will be able to switch between different views.
- If user wants to execute his/her query by using the query window on the top of “Database” view, s/he will write queries 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 will be able to perform “New File”, “Open File”, “Close File”, “Save File”, “Save File As” and “Exit”.
  - If user selects “Edit” submenu of “Menu Bar”, s/he will be able to perform “Undo”, “Redo”, “Cut”, “Copy”, “Paste”, “Delete” and “Find” actions.
  - If user selects “Project” submenu of “Menu Bar”, s/he will be able to perform “New Project”, “Open Project”, “Close Project”, “Run Project”, “Debug Project” and “Step Over”, “Step Into”, “Step Out” actions.
  - If user selects “Tools” submenu of “Menu Bar”, s/he will be able use “Database Connection Manager” to connect database or send his/her files by using “Publish via FTP” option.
  - If user selects “Window” submenu of “Menu Bar”, s/he will be able to show or hide “Workspace View”, “Project View”, “Outline View”, “Palette View” and “Debugger View”.
  - If user selects “Versioning” submenu of “Menu Bar”, s/he will be able use “CVS Manager”. User can easily “Import” or “Check-out” his/her files.
  - If user selects “Help” submenu of “Menu Bar”, s/he will be able to choose “Help Contents” or “About”.

- User will be able 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 will be able to perform the action of that icon.
    - Possible icons that will be shown on the toolbar are, “New File”, “Open File”, “New Project”, “Save File”, “Undo”, “Redo”, “Search”, “Cut”, “Copy”, “Paste”, “Run”, “Debug” and “Stop”.
    - User will be able to customize the toolbar, by clicking arrow at the right of toolbar and selecting an icon to be shown on the seen part.

- 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 will be provided for user.

    - New (CTRL + N)
    - Save (CTRL + S)
    - Load (CTRL + L)
    - Find (CTRL + F)
    - Find & Replace (CTRL + H)
    - Cut (CTRL + X)
    - Copy (CTRL + C)
    - Paste (CTRL + V)
    - Select all (CTRL + A)
    - Undo (CTRL + U)
    - Redo (CTRL + R)
    - Switch to design view (CTRL + D)
    - Switch between tabs (CTRL + T)

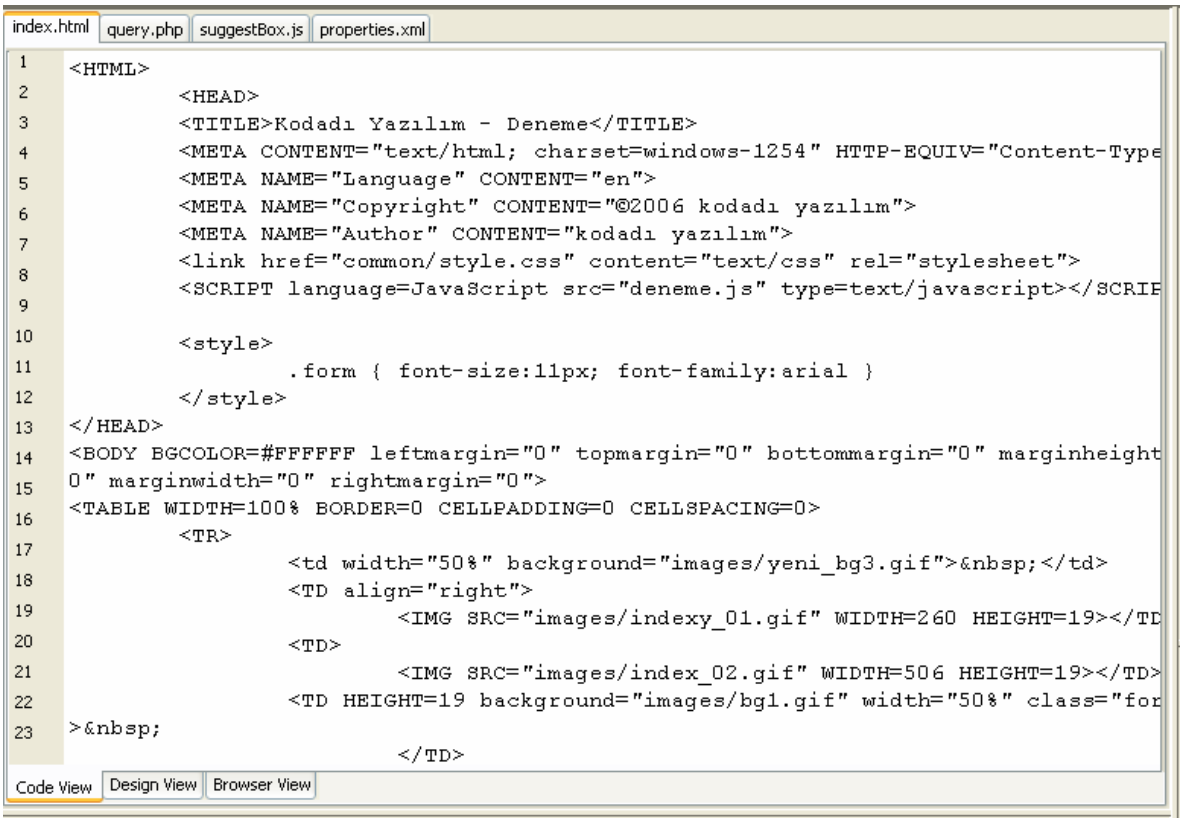
- Keyboard shortcuts for pause, resume, step in/over/out, break will be provided.
    - Break (Pause)
    - Go (F5)
    - Step into (F11)
    - Step over (F7)
    - Step out (F8)

- 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.

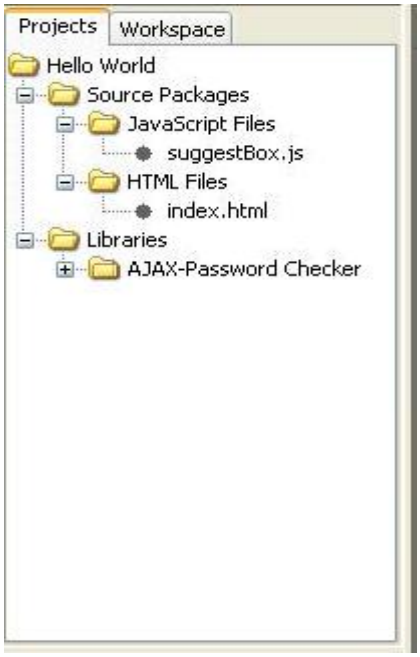
6.3 Screenshots of GUI

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

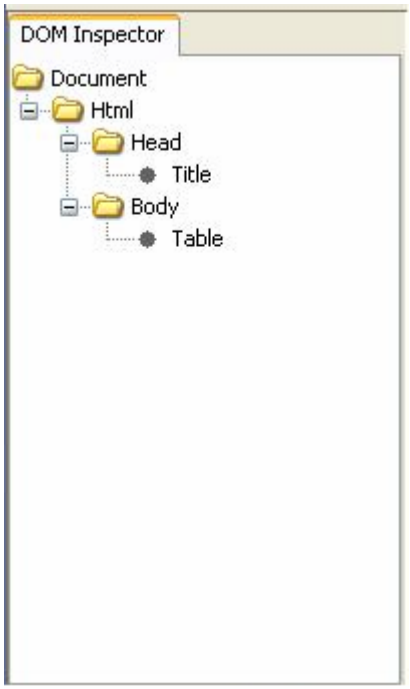
6.3.1 “Code”, “Design” and “Browser” views



6.3.2 “Project” and “Workspace” views

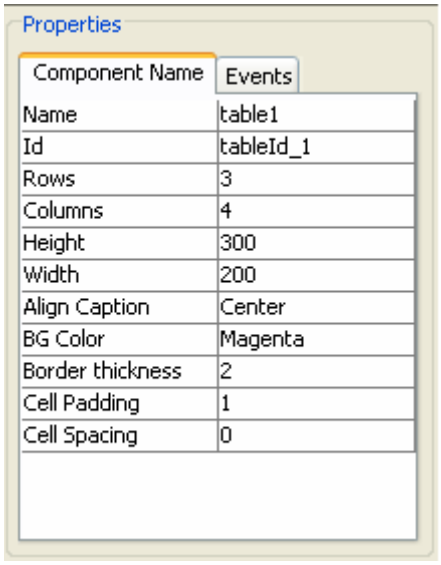


6.3.3 “DOM Inspector” view

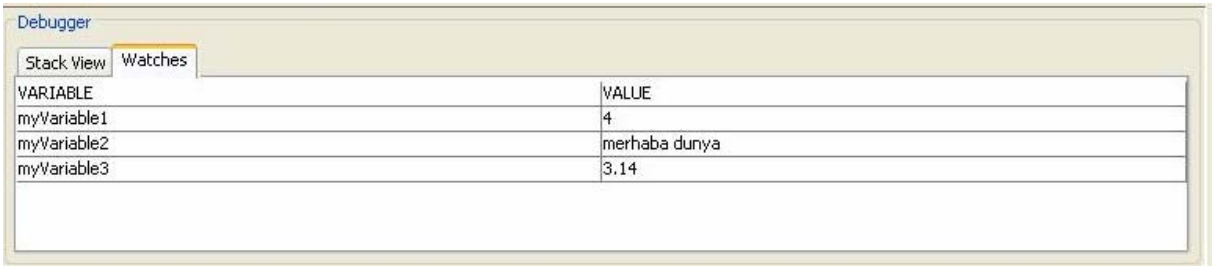


6.3.4 “Palette” view

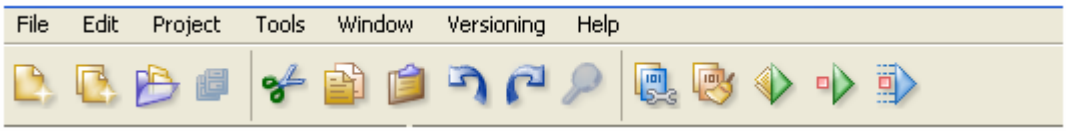
6.3.5 “Properties” and “Events” views

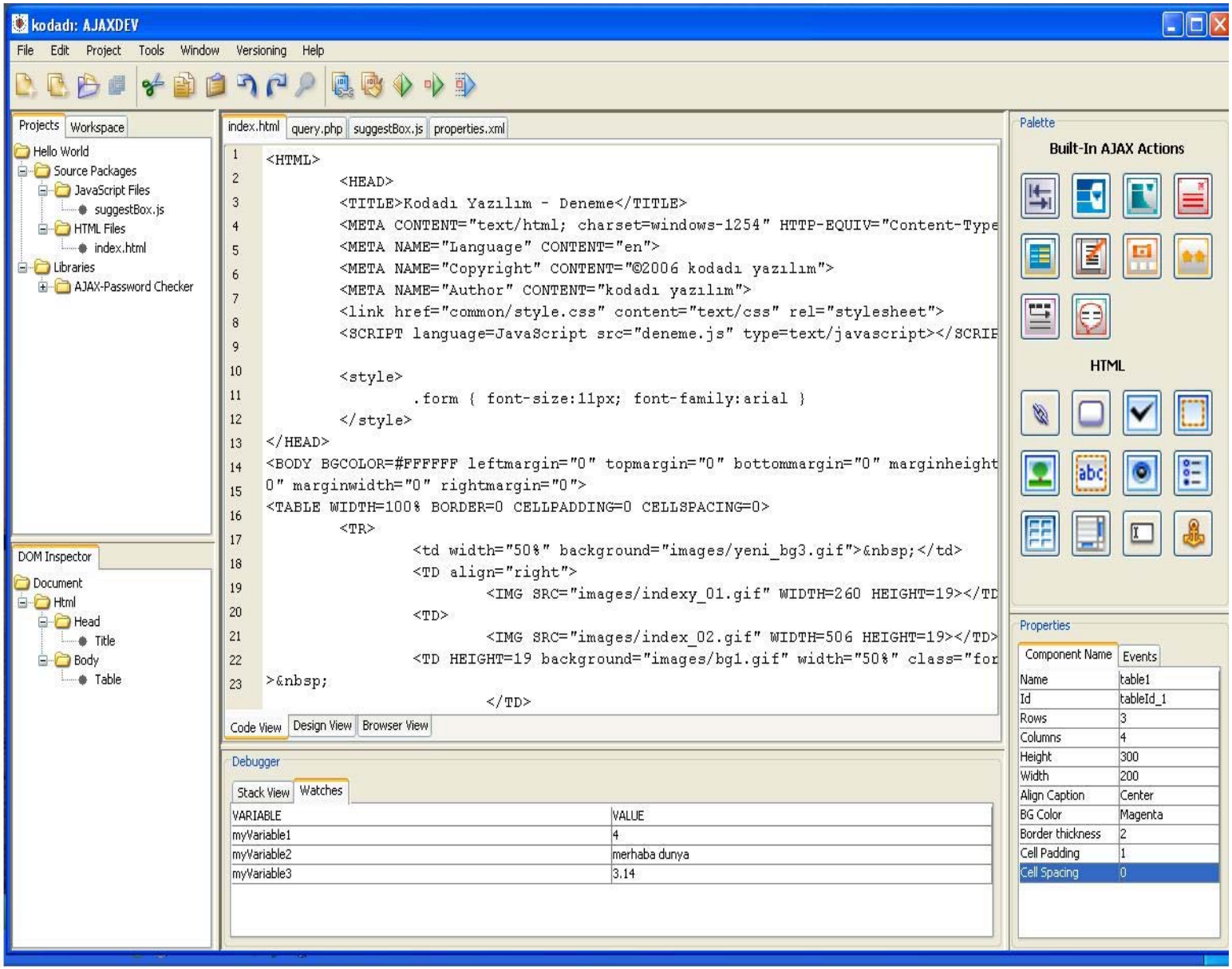


6.3.6 “Debugger” view



6.3.7 “Menu Bar” & “Tool Bar”





6.3.8 Final view of GUI

7. SYNTAX SPECIFICATION

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. TESTING ISSUES

8.1 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.

8.1.1 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**

8.1.2 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.

8.1.3 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.

9. CONCLUSION

This is the Initial Design Report of kodadı: AJAXDEV project. During preparing this report, we have tried to decide on the way we will implement our product. We have made several discussions when deciding on class attributes and operations however this still needs more review which will be done until Final Design Report. Up to Final Design Report we plan to improve our design procedure and provide some functionality on GUI.

10. APPENDIX