MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

CEYLAN©

AJAX SOFTWARE DEVELOPMENT STUDIO

FINAL DESIGN REPORT

Group Members:

Saliha ALTUNSOY

Candan CEYLAN

Canan ESKİ

Yavuz GÖKIRMAK

Duygu GÖRGÜN

Table of Contents

1.0 INTRODUCTION

This document aims to provide detailed information about the design process of the problem. During the analysis phase of our project, we investigated the possible problems and then in the initial design report we had come up with solutions. In this final design report we refined our architectural design and add some new modules to our project.

Firstly, problem definition and the scope of the project can be found in this document. Detailed version of the revised Data Flow Diagrams and data dictionary are also included. Moreover, to describe the processes, we used use case diagrams, sequence diagrams for each use case, and activity diagrams when necessary. Class diagrams are also added to describe the relationships between classes and modules. In addition to these, detailed description of our GUI design and actions are told. Our test strategy and preliminary test case specifications are described. Finally, a revised version of our projects Gantt Chart is included.

1.1 Project Title

Our development studio is called CEYLAN Ajax Development Studio.

1.2 Problem Definition

These days, web applications are being used in a wide range of areas; from

banking applications, search engines, e-government applications, online-library applications, etc. to non-institutional web pages. In most of these areas, response time is a very important issue. To illustrate, for a search engine the application must retrieve the result and display it in a short time period. Or in a banking application, the purpose is to allow the customer to transact instead of going to the bank branch and waiting for other people to complete their operations. If the user waits too long for response in front of the screen, using the web application will be meaningless. AJAX offers a different approach to make these web applications respond faster.

In the traditional approach application model works like this: most user actions in the interface trigger an HTTP request sent to the web server, the server does the processing – retrieving data, crunching numbers, etc. – and then returns an HTML page to the client. Even if there is a very little change in the interface, the whole page is reloaded again. And the user waits until the server responds. Instead, AJAX offers an approach which eliminates the start-stop-start-stop nature of interaction on the web. That is; instead of loading the entire page again and again, only the changed parts are loaded and the user can interact with the application asynchronously, independent of communication with the server. Actually, **AJAX** is shorthand for **A**synchronous **Ja**vaScript **a**nd **X**ML.

Due to the advantages offered by Ajax, most of the developers use this approach. During development stage, the development environment is a very important issue and some utilities are expected from IDEs to make the job of developers easier. The developers of Ajax need a user-friendly and platform independent IDE and in the market, there is not many products working as desktop applications to satisfy the developers' needs.

1.3 Statement Of Scope

Our project, CEYLAN, is an integrated development environment for developing web pages with Ajax. Our aim is to offer a user friendly and functional IDE which will additionally include database connection and configuration wizard for the developers from different backgrounds; professionals or amateurs. We will develop a desktop application which doesn't need an internet connection to run. Our development environment will be developed regarding the following four basic design principles;

- User friendliness
- Modularity
- Platform Independency
- Consistency

Our IDE's interface will be graphical and user friendly. It will work regardless of operating system and the code developed using it will run in different browsers. It will be consistent; will not surprise the user presenting unexpected behaviors. Concerning these design parameters, our product will include the following features;

- Widgets such as tree view, tool bars and tabs
- Server connectivity
- Extensive tools for development and debugging
- Syntax highlighting
- Syntax checking
- Auto completion
- Error handling
- Code indentation
- Help menu

- Integrated browser
- Code and design view
- Predefined code generation
- Customizable user interface
- Database connection wizard
- Views of database tables
- Displays of SQL queries
- File Transfer (FTP)
- CVS support

1.4 Application Areas of Our Software

Our software will be used in most of the areas where web application development is the issue; by companies, individual developers, students in educational concepts, etc. Our target user profile consists of the whole software developers using Ajax technologies. Our product will be able to serve more than enough functionality for users having different amount of knowledge and experience.

2.0 PROJECT REQUIREMENTS

2.1 Functional Requirements

2.1.1 Widgets such as tree view, toolbars and tabs

User will be provided with widgets such as tree view, database view, toolbars and tabs. With these tools it'll be easier for the developer to follow and

manage her/his work.

2.1.2 Server connectivity

Connection Wizard: Connection operation is done via a wizard. User chooses appropriate options step by step. Options are database address, connection engine, connection name, etc

2.1.3 Query Execution

- SQL Command Screen: Executes SQL queries and returns results in dos like command screen.
- SQL Executer Interface: Similar SQL operations “SQL Command Screen” supplied in more user-friendly interface. And some sub operations are provided which are as follows:
    - Save SQL
    - Load SQL
    - Execute SQL
    - Prepared SQL

2.1.4 Editing, and Debugging

There are some features provided with for editing and debugging. These are;

- Syntax Highlighting: The code will be highlighted according to specified types, keywords and functions.
- Code indentation: User can indent the file or some part of the file.
- Auto-completion: While the developer is writing the code, upon her/his request s/he will be provided with auto-completion. This completion can be done using libraries or user code.
- Error-handling: When the user interprets the code, errors are detected and the user will be provided with the appropriate error message.

- Run: Invokes interpretation via GUI.
- Debug: Starts debug mode. User can set breakpoints or do step in step out etc.

2.1.5 Work Space

The user is provided a workspace with the features below:

- Hotkeys: Hotkeys of editor functions is provided. Simply as: ctrl+c copy ctrl+v paste.
- Auto Completion: Auto completion of code. Appropriate code is generated according to the table that the parser generated..
- Split View: Partitioned design screen is available. Design screen is divided as Code&Design.
- File Tree: Organization of files represented as tree view.
- Database Tree: Gives the tree view of the databases connected, and the tables, functions, triggers etc. corresponding to these databases.
- Drag&Drop: By simply dragging and dropping an item into design screen, users can generate code.

2.1.6 Help menu

Help menu will contain:

- Search: User can make search by entering the word to be searched.
- Tutorial: Usage of program is explained step by step.
- FAQ: Some frequently asked questions and answers.
- Online Help: Link to internet site.

2.1.7 Menu Components

Menu components will be explained in details in the User Interface part.

2.1.8 Integrated browser

There will be an integrated browser which pops up after the code is interpreted and ready for display. So the user can see what s/he does on a web page view.

2.1.9 Code and design view

While the user is editing her/his codes/he will be able to see her/his code as code view, design view or split view.

2.1.10 Predefined code generation

When the user is in design or split view and buts a figure on design part, the corresponding predefined code will be automatically put exact place in code file.

2.1.11 FTP Operations

User connects to the server through connection wizards. After the connection is established, user will be able to transfer files from local machine to server or from server to local machine.

2.1.12 CVS Operations

User connects to the server through connection wizards. After the connection is established, user will be able to lock or release files. Moreover, the user will be able to check in files from local machine to server or check out files from server to local machine.

2.2 Non-Functional Requirements

2.2.1 User-friendliness

Our aim is to provide a user-friendly program to user. In our project we can accomplish this goal with two components; an User Friendly Interface and Easy Coding Features. GUI provides easy-to-understand and easy-to-use interface. User won't lose within menus. Functions which are similar are grouped. Our program provides Drag&Drop operations. With the help of this user can create applications very easy.

2.2.2 Modularity

Modularity is important in our design because with the help of it we can add and remove components easily. We designed system as modules, this modules work together but one module don't need to know internal design and internal processes of another module.

2.2.3 Platform Independency

We chose Java as programming language. Java choice is actually related Java Virtual Machine (JVM). We need machine independent software and JVM provides us a virtual layer between software and machine. Thus we can write machine independent code. Our software will be used on every system which has JVM.

2.2.4 Consistency

Our system will behave in a predetermined manner. We will lessen the unexpected states or behaviors of system as much as possible.

2.3 System Requirements

2.3.1 Software Requirements:

Development Phase

Project CEYLAN will be developed in Java, since **Java** allows us to develop a platform independent code. Moreover, Java offers a wide range of functions to develop a high-quality graphical user interface. Thus **Java Runtime Environment** and **Java SDK** are the essential tools to be used in this project. Moreover, for reporting and preparing help menus we will use design tools such as **MS Visio, Borland Together.**

To develop our application, we will work in Windows XP and Linux platforms.

For the End User

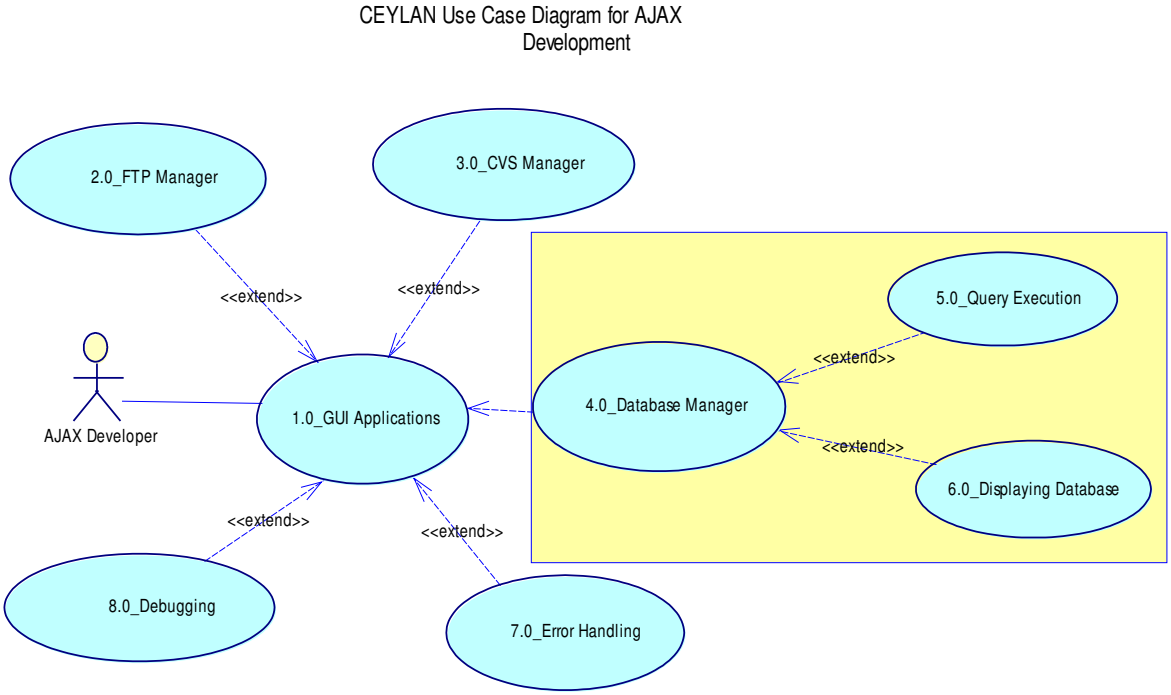
Since our application will be developed in Java and will be platform independent, the end user will only need a compatible operating system and a Java Runtime Environment. And since we are using web service in the development phase, the user can choose whichever database he/she wants.

2.3.2 Hardware Requirements:

- 512 MB disk space
- 512 MB RAM (1024 preferable)
- Intel 500 MHz Processor (or above)

3.0 ARCHITECTURAL DESIGN

3.1 Ceylan Use Case Diagram for AJAX Development



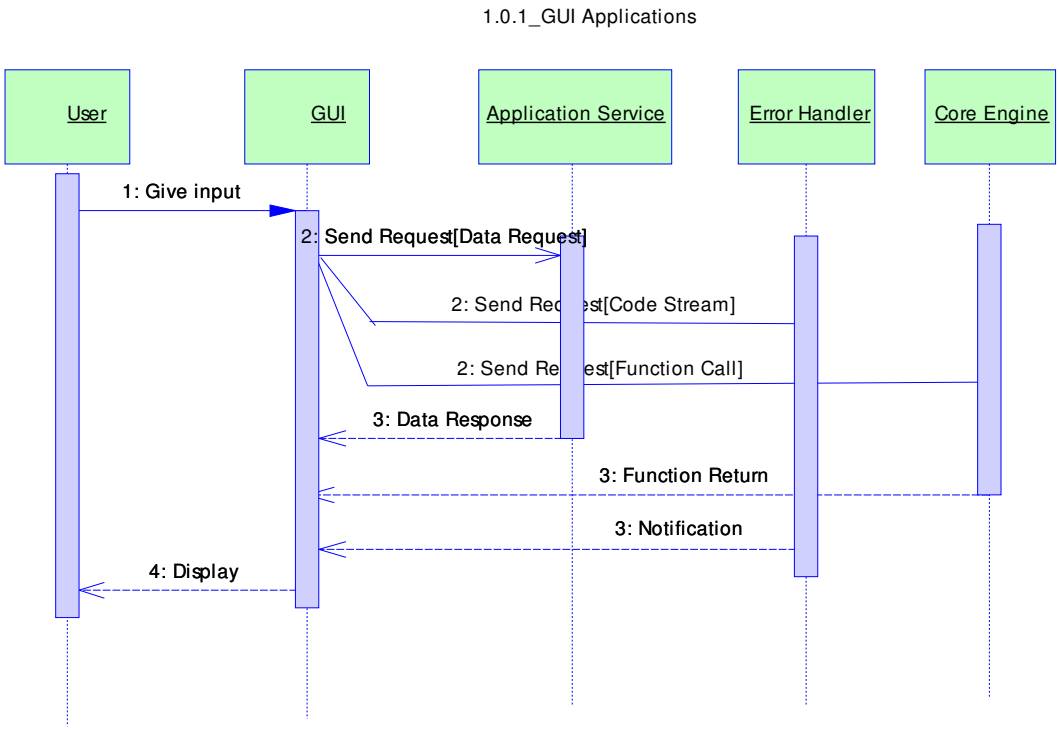
Ajax developer does necessary operations through GUI. Actually, besides GUI applications, there are basically five operations that can be done by using GUI:

- FTP operations: These are done by our FTP Manager which gives user the opportunity that is file transfers. More detailed information about FTP Manager is at our class diagrams.
- CVS operations: These are done by our CVS Manager which gives user the opportunity that is also file transfers but in controlled way. More detailed information about CVS Manager is at our class diagrams.
- Database operations: These are done by our Database Manager which gives user the two basic opportunities those are executing queries and displaying tables. More detailed information about Database Manager is at our class diagrams.

- Error Handling: Tokenizing and parsing code, sending the information about errors and parts to be highlighted and constructing the language data table are done by Error Handler. More detailed information about Error Handling is at our class diagrams.
- Debugging: The debugging operations done by debugger.

3.2 Interaction Models of Our System

3.2.1 Sequence Diagram of GUI Applications



This diagram shows operations' sequence which are done only through GUI.

Step 1: User (AJAX Developer) gives his/her input to GUI in order to start operation.

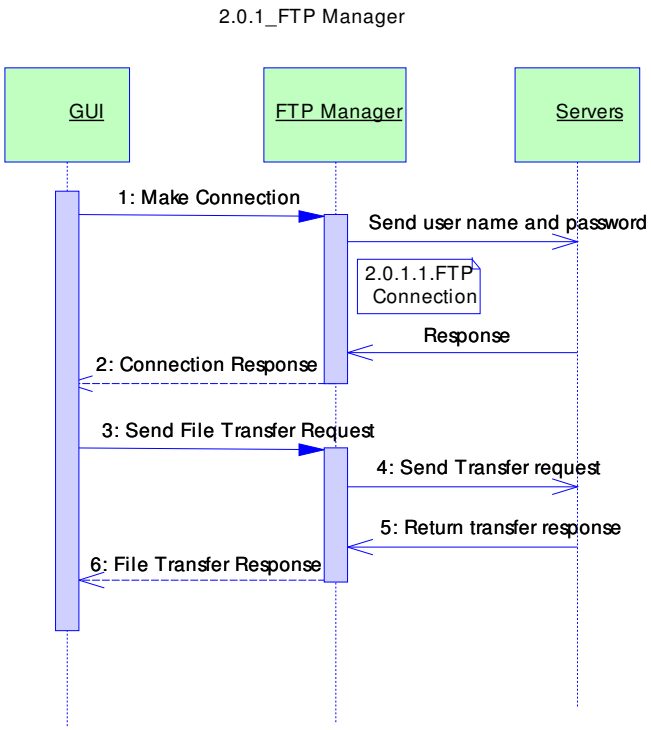
Step 2: GUI does the operation according to the input which was got in first

step. The operations may be data request sending or code steam sending or function call request.

Step 3: According to the operation done in step 2, related answer returns in this step. If the operation was data request, then the answer would be data response, if the operation was code stream, then the answer would be notification and lastly if the operation was function call, then the answer would be function return.

Step 4: GUI displays relevant result to user.

3.2.2 Sequence Diagram of FTP Manager



This diagram shows the operations' sequence which are done by FTP Manager

Step 1**:** In order to use FTP manager, firstly it must be activated through GUI. The operation is making connection. Then FTP manager send user name and password to server in order to check the correctness and send a result

whether connection is established or not. This operation is shown detailed in activity diagram named 2.0.1.1_FTP Connection at part 3.2.2.1.

Step 2: According to the response of server, FTP manager returns response.

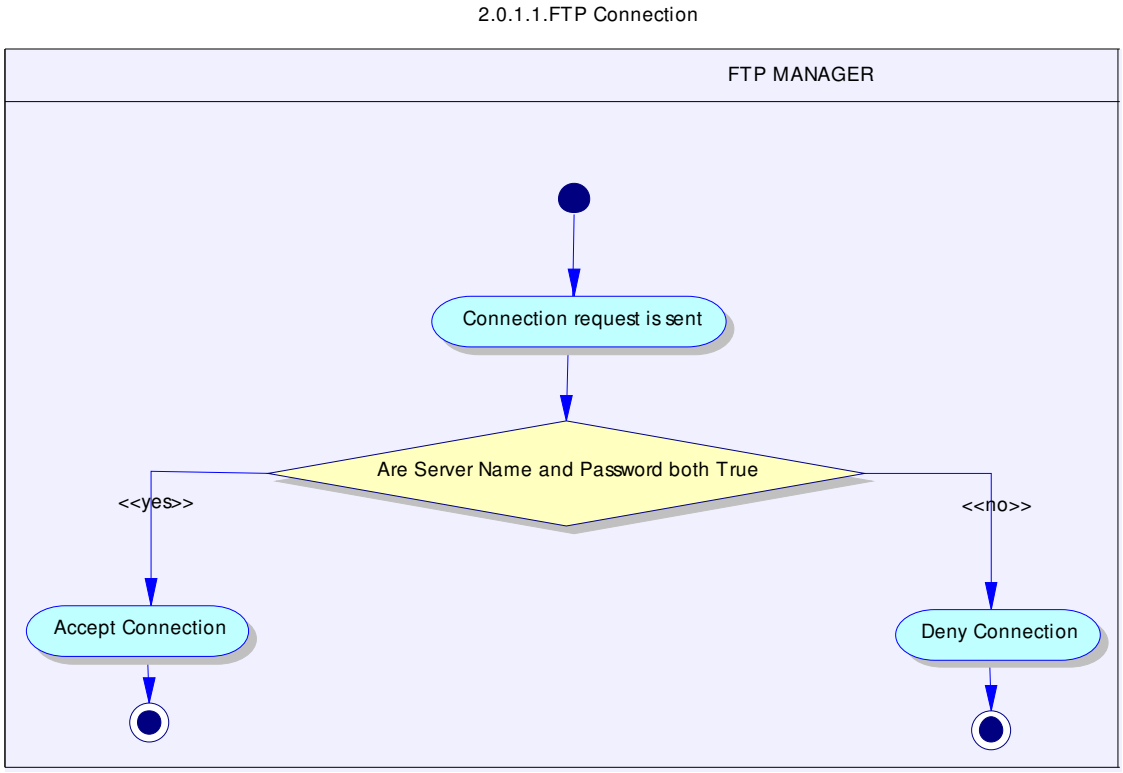
Step 3: From GUI, the request is sent to FTP manager. Request is file transfer.

Step 4: File transfer request is sent to from FTP manager to server.

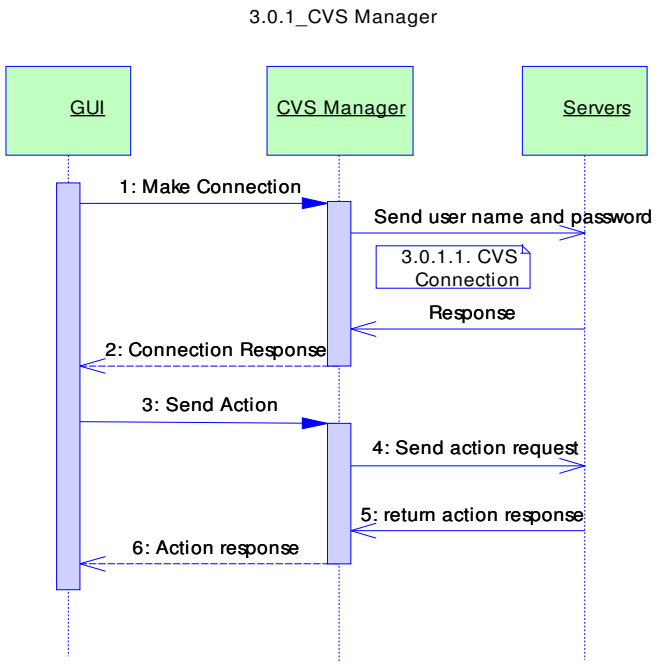
Step 5: File transfer response is sent from server to FTP manager.

Step 6: According to the result come from server, FTP manager returns the result of file transfer request.

3.2.2.1. Activity Diagram of FTP Connection



3.2.3 Sequence Diagram of CVS Manager



This diagram shows the operations' sequence of CVS manager.

Step 1: In order to use CVS manager, firstly it must be activated through GUI. The operation is making connection. Then CVS manager send user name and password to server in order to check the correctness and send a result whether connection is established or not. This operation is shown detailed in activity diagram named 3.0.1.1_ CVS Connection at part 3.2.3.1

Step 2: According to the response of server, CVS manager returns response.

Step 3: From GUI, the request is sent to CVS manager. Request is CVS related operations.

Step 4: CVS related operation's request is sent to from CVS manager to server.

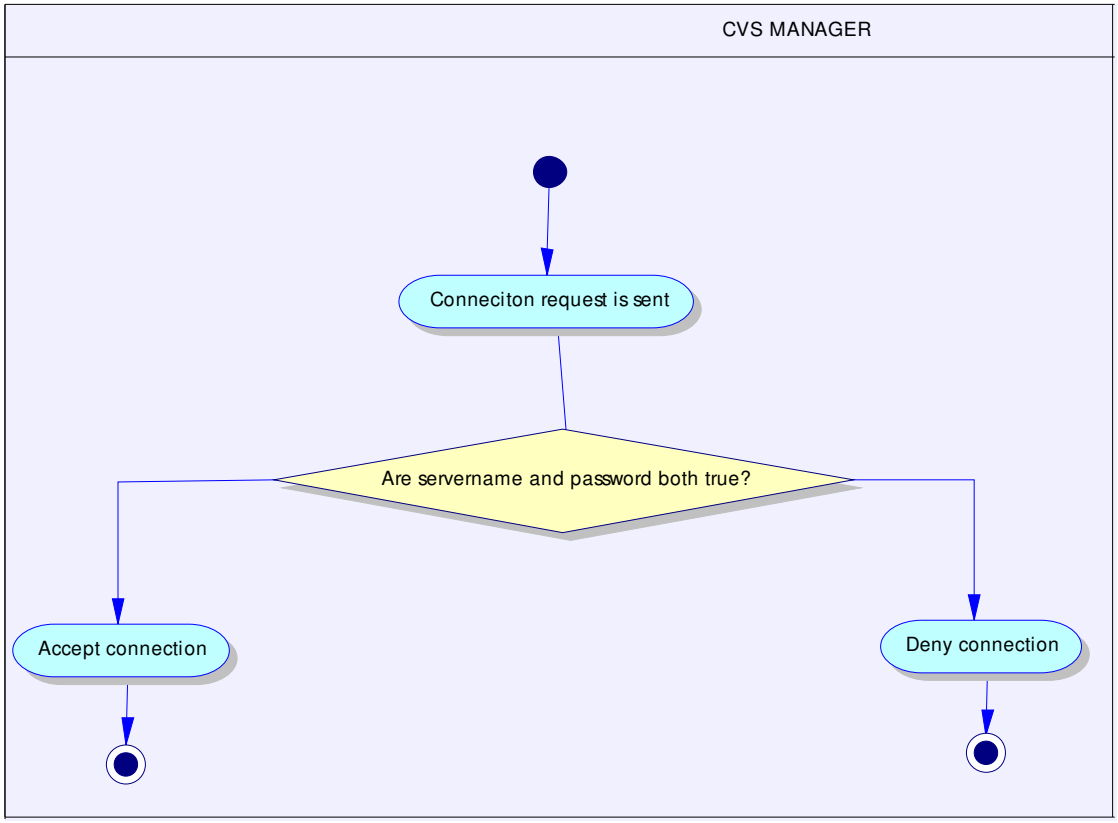
Step 5: CVS related operation's response is sent from server to CVS manager.

Step 6: According to the result come from server, CVS manager returns the

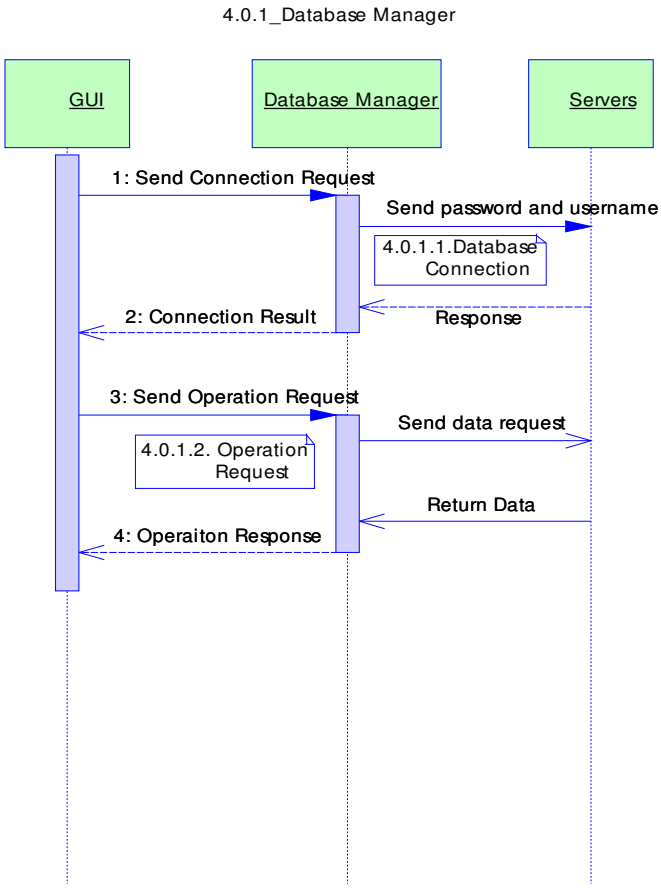
result of CVS related operation’s request.

3.2.3.1. Activity Diagram of CVS Connection

3.0.1.1. CVS Connection



3.2.4 Sequence Diagram of Database Manager



This diagram shows the operations' sequence of Database manager.

Step 1: In order to use Database manager, firstly it must be activated through GUI. The operation is making connection. Then Database manager send user name and password to server in order to check the correctness and send a result whether connection is established or not. This operation is shown detailed in activity diagram named 4.0.1.1_Database Connection at part 3.2.4.1

Step 2: According to the response of server, Database manager returns response.

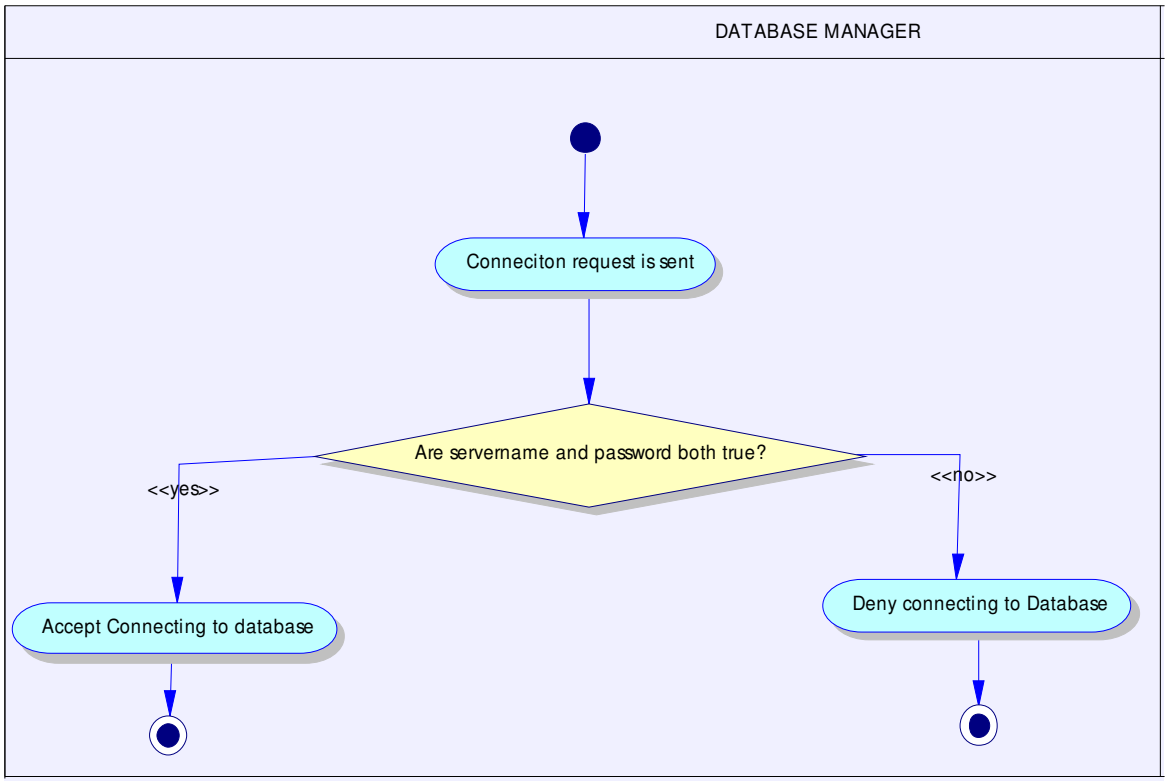
Step 3: From GUI, the request is sent to Database manager. Requests are query executing or displaying tables. More detailed information about these

requests is shown at activity diagram named 4.0.1.2_Operation Request at part 3.2.4.2. Then the request is sent to server in order to get relevant data.

Step 4: According to the e result come from server, Database manager returns the result of the request.

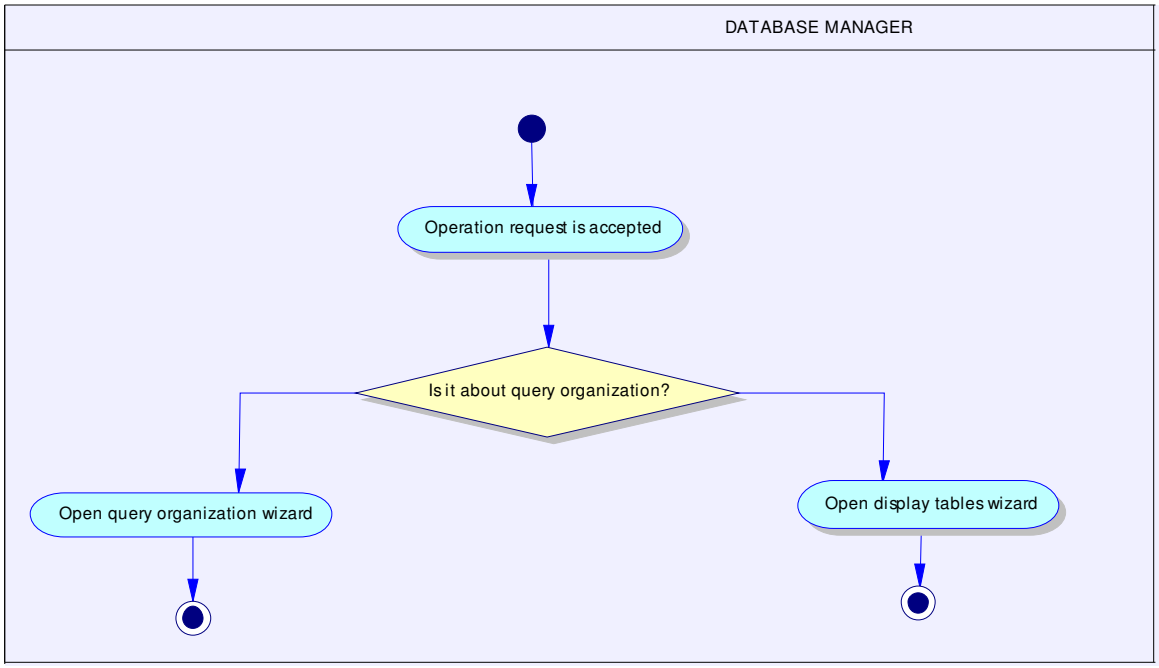
3.2.4.1 Activity Diagram of Database Connection

4.0.1.1.Database Connection



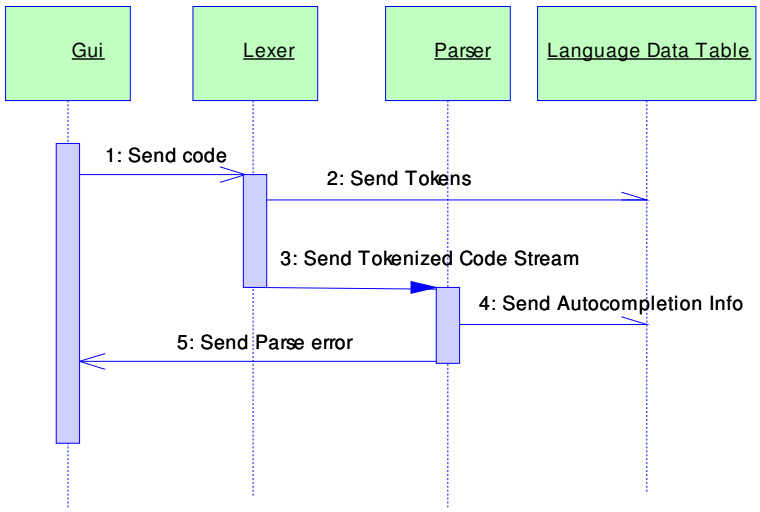
3.2.4.2. Activity Diagram of Operation Request

4.0.1.2. Operation Request



3.2.5 Sequence Diagram of Error Handler

7.0.1_Error Handler



This diagram describes how to handle the errors.

Step 1: The user writes code and it is sent to Lexer through GUI.

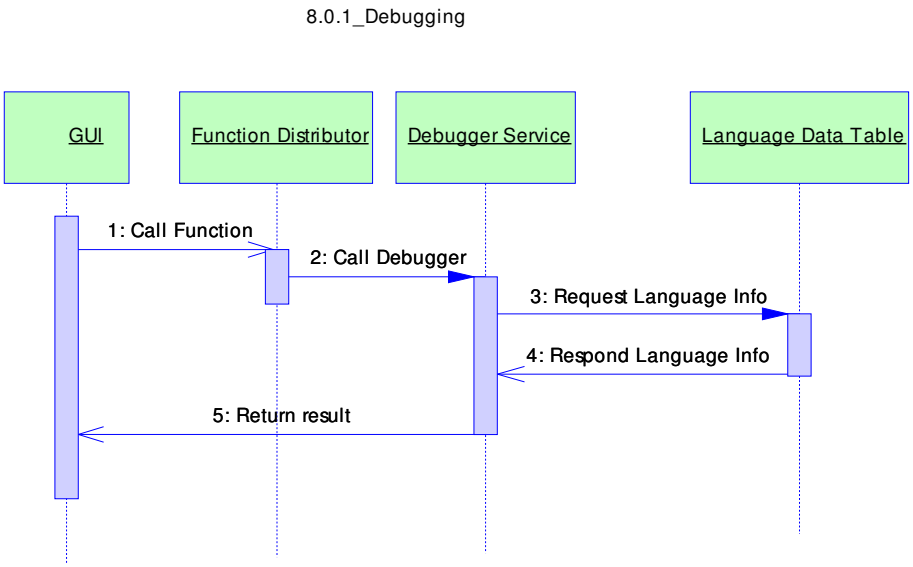
Step 2: The Lexer builds up the tokens and send them to language data table in order to make them usable later errors.

Step 3: Lexer sends tokenized data to parser in order to check the errors.

Step 4: The parser sends auto completion information to Language data table in order to make it usable later on.

Step 5: The errors are sent to GUI.

3.2.6 Sequence Diagram of Debugger



This diagram shows the basic operations done for debugging.

Step 1: A function call is made and the debugger is activated, if the function call is for debugger.

Step 2: After function distributor decides the function is debugger call, it calls debugger.

Step 3: The debugger sends request to language data table in order to check the code.

Step 4: Language data table returns relevant data information.

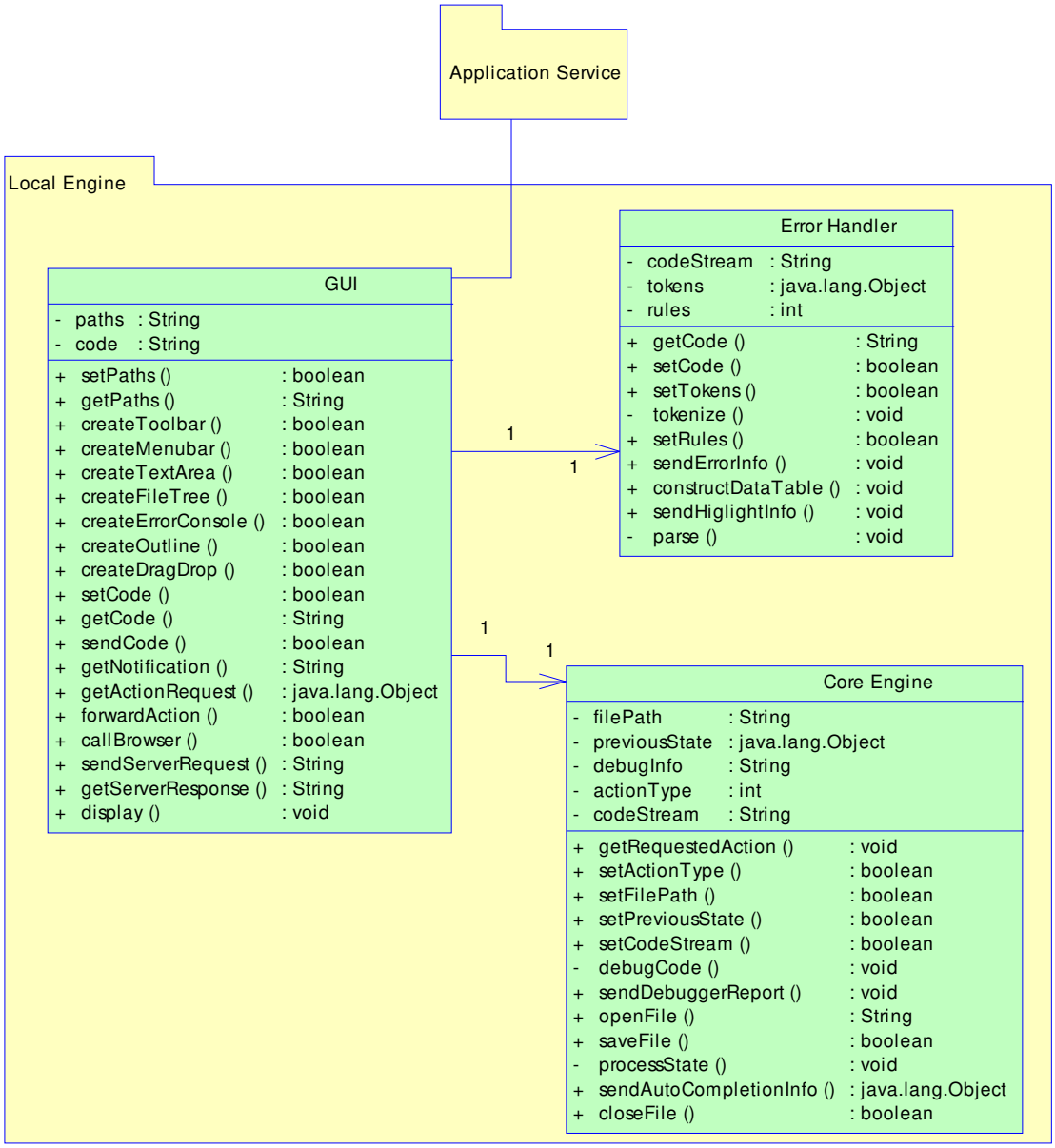
Step 5: The result is sent to GUI.

3.3 Classes of CEYLAN and Their Relationships

In design of CEYLAN Ajax Developers' Studio, we decided to have two main modules. These are Local Engine and Application Service. Local Engine and Application Service modules are shown as packages in the following class diagrams, which introduce our classes, relationships between them and other packages. Both Local Engine and Application Service packages are consisting of classes which are responsible from constructing CEYLAN. These classes are in communication with each other and this communication is achieved by function calls. The data flows between these classes are handled according to the interfaces provided by these classes as functions. As a result, data flows in Ceylan are well formed.

3.3.1 Local Engine

Local Engine is the main module of Ceylan. Most of the job is done here. GUI sub module is included in this module which is responsible from the interactivity between user and our internal system. GUI takes the requests from user and forwards them to relevant classes if necessary. Second sub module included in this module is Error Handler. Error Handler is responsible from lexical analysis and construction of the parse tree from the code. The third sub module is Core Engine. Core Engine is responsible from file operations, auto completion and debugging applications. The class diagram for Local Engine and explanations can be found below.



3.3.1.1 GUI:

This class is responsible from the interaction between user and our internal system. It gets the requests from user, and according to the request type takes the necessary action. It either does the job itself or forwards to other classes. However, whatever the request is, displaying the result is performed by this class. GUI is in communication with two other classes and Application Service package. It sends requests about syntax highlighting or syntax checking to Error Handler class. And the requests about debugging, auto completion and file operations are sent to Core Engine class. In addition to these, the requests about database, CVS and

FTP operations are sent to the Application Service package. The functions in this class are listed below:

- **setPaths():** Paths are used to reach to some necessary files that will be used in the startup and this function is used to set this paths property.
- **getPaths():** Returns the stored value of paths property.
- **createToolbar():** Creates the customizable toolbar.
- **createMenubar():** Creates the menubar with sub menus.
- **createTextArea():** Creates the text area where the user will enter his/her code.
- **createFileTree():** Creates the tree view of workspace files.
- **createErrorConsole():** Creates the error console where the errors and warnings will be shown.
- **createOutline():** Creates the outline frame where the HTML tags will be shown to the user in tree view.
- **createDragDrop():** Creates the drag and drop frame where the user can find predefined codes for visual elements like combo boxes, radio buttons etc.
- **setCode():** Sets the code property of the GUI class.
- **getCode():** Returns the code property.
- **sendCode():** Sends the string stored in the code property to Error Handler.
- **getNotification():** Gets the errors, warnings and associated syntax highlight information from Error Handler.
- **getActionRequest():** Gets the action request from user by listening actions.
- **forwardAction():** Forwards the requested actions to the relevant classes with information needed to take action.
- **callBrowser():** Calls the browser to display the design view of the code supplied by user.
- **sendServerRequest():** Sends the server related requests to Application Service package.
- **getServerResponse():** Gets the response of the server for requested

server related actions.

- **display():** Combines all the graphical view and displays to the user.

3.3.1.2 Error Handler:

This class is responsible from -as the name implies- error handling. By its internal functions, tokenizes and parses the code, sends the information about errors and parts to be highlighted and constructs the language data table. The functions of this class are listed below:

- **getCode():** Returns the codeStream property of the class.
- **setCode():** Sets the codeStream property with the specified value.
- **setTokens():** Sets the tokens property of the class with the tokens after tokenization process.
- **tokenize():** Does the lexical analysis of the code and tokenizes it according to the rules specified in the rules property.
- **setRules():** Sets the rules property of the class with the rules necessary for generating parse trees.
- **sendErrorInfo():** Sends information about erroneous parts found in the code.
- **constructDataTable():** Constructs information to be stored in language data table using the information got from parsing process.
- **sendHighlightInfo():** Sends information about which parts will be highlighted in the code.
- **parse():** Parses the code after tokenization process.

3.3.1.3 Core Engine:

This class is responsible from file operations such as open, save, close etc. Moreover, debugging process is performed here. Auto completion is also handled in this class. The functions included in this class are listed below:

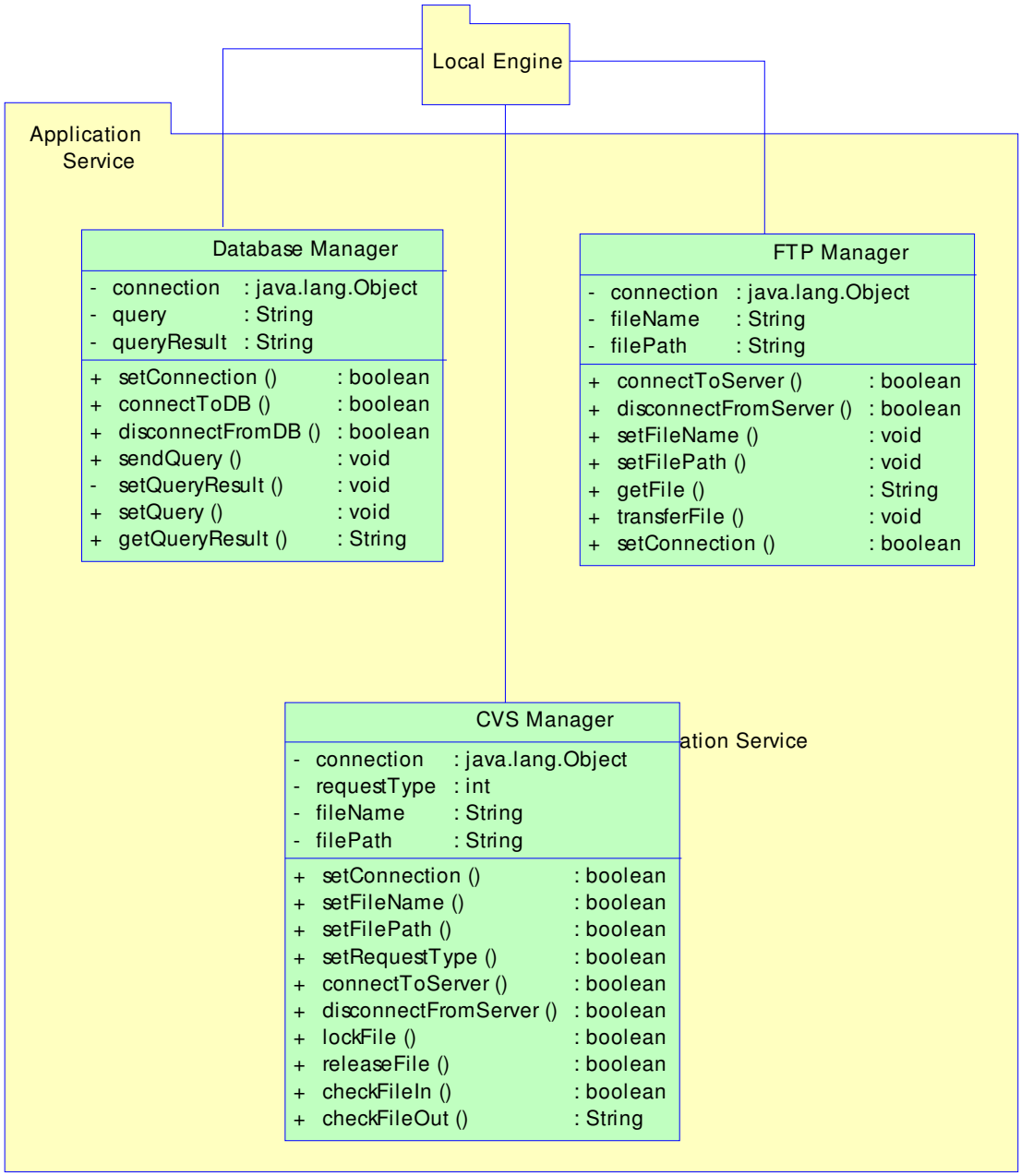
- **getRequestedAction():** Gets the requested action from GUI class.
- **setActionType():** Sets the actionType property of this class according to

the action type requested by the user.

- **setFilePath():** Sets the filePath property which will be used in file operations.
- **setPreviousState():** Sets the previousState property which will be used if the requested action is auto completion.
- **setCodeStream():** Sets the codeStream property to be used in debugging process.
- **debugCode():** The function used for debugging the code.
- **sendDebuggerReport():** Sends result to be shown to user constructed during debugging process.
- **openFile():** Opens the file specified by filePath property of the class.
- **saveFile():** Saves the file specified by filePath property of the class.
- **closeFile():** Closes the file specified by filePath property of the class.
- **processState():** Processes the previousState property with the help of language data table in order to generate a list of possible suggestions for auto completion.
- **sendAutoCompletionInfo():** Sends the information generated by processState function to be displayed to user.

3.3.2 Application Service

Application Service is the second module of Ceylan which actually is the bridge between Local Engine and Servers. GUI class gets the server related requests from user. It consists of three sub modules. These are Database Manager, FTP Manager and CVS Manager modules which are shown as classes of Application Service package. The first class Database Manager handles requested database operations such as connection to the database and query execution. The second class, FTP Manager's job is to handle file transfer related requests. And lastly, CVS Manager is responsible from performing necessary actions for requested CVS related operations. The class diagram for Application Service and explanations can be found below.



3.3.2.1 Database Manager:

This class is responsible from database related operations such as connection, query execution etc. It is the job of this class to establish the connection to the database, disconnect from the database, and send queries entered by user or queries used to get information about database items for the graphical interface to the database server. The functions that are defined in this class are listed below:

- **setConnection():** Sets the connection object of this class which will be used to establish connection.
- **connectToDB():** Establishes the connection to the database using the username, password and target database related information stored in the connection object.
- **disconnectFromDB():** Disconnects from database.
- **sendQuery():** Sends the query to be executed to the database.
- **sendQueryResult():** Sends the query result returned from database to Local Engine package to be displayed in GUI.
- **setQuery():** Sets the query property of the class with the requested query.
- **getQueryResult():** Gets the query result from database.

3.3.2.2 FTP Manager:

This class is responsible from file transfer related operations such as connection, transferring files to or from server etc. It is the job of this class to establish the connection to the server, disconnect from the server and transfer operations. The functions that are defined in this class are listed below:

- **connectToServer():** Connects to server where the file transfer operations will be according to the information stored in the connection object.
- **disconnectFromServer():** Disconnects from server after the file transfer operations finish.
- **setFileName():** Sets the file name property of the class which specifies the file name to be transferred from server or to server.
- **setFilePath():** Sets the file path property of the class which specifies the file path to be transferred from server or to server.
- **getFile():** Gets the file specified by file path and file name from the server.
- **transferFile():** Transfers the file specified by file path and file name from local machine to server.
- **setConnection():** Sets the connection object belonging to this class.

3.3.2.3 CVS Manager:

This class is responsible from CVS related operations such as connection, locking or releasing files and checking in or out files etc. The functions defined in this class are listed below:

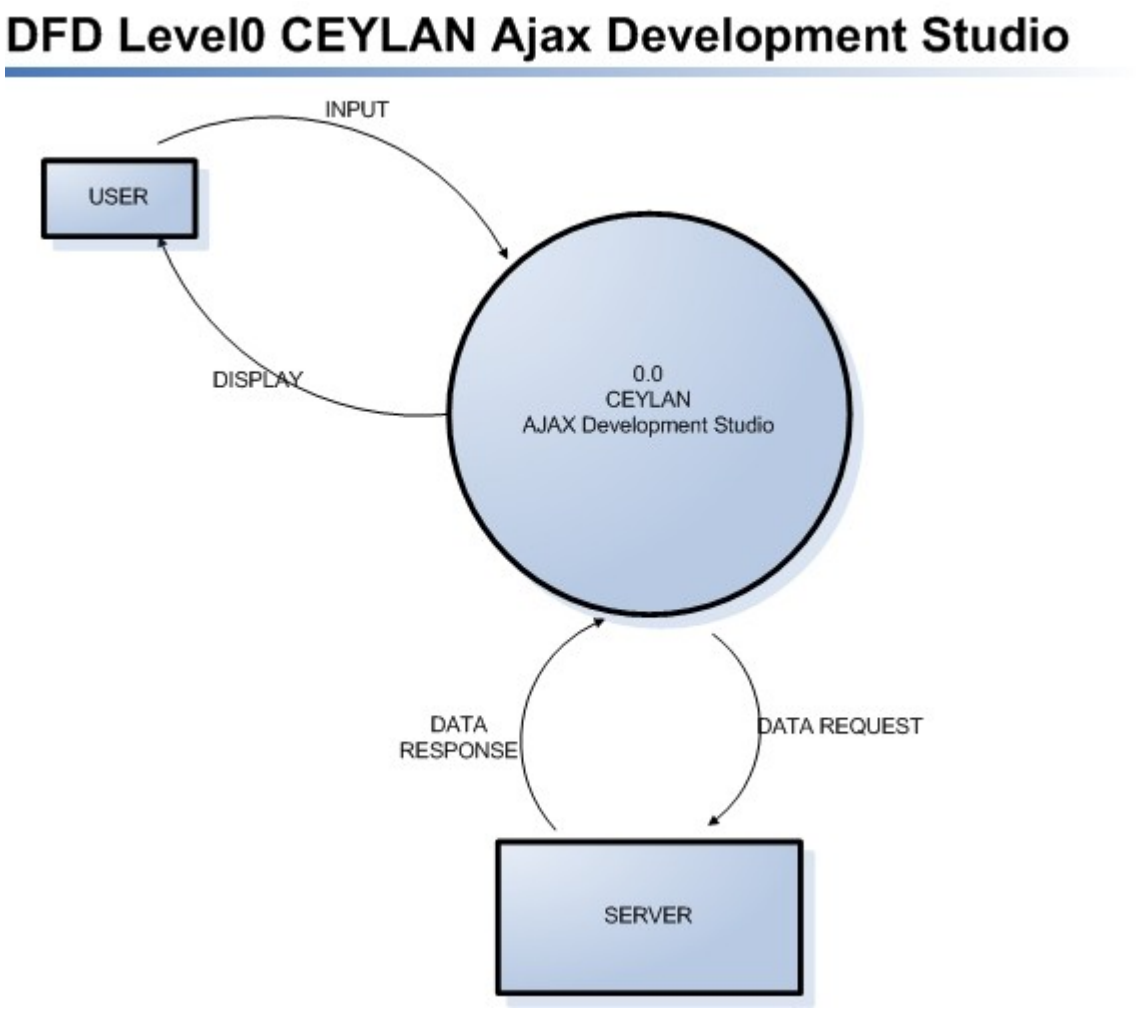
- **setFileName():** Sets the file name property of the class which specifies the file name to be checked in or out.
- **setFilePath():** Sets the file path property of the class which specifies the file path to be checked in or out.
- **setConnection():** Sets the connection object belonging to this class.
- **setRequestType():** Sets the requestType property according to the user request. This request type can specify following; locking, releasing, check-in operation or check-out operation.
- **connectToServer():** Establishes the connection to server.
- **disconnectFromServer():** Closes the server connection.
- **lockFile():** Sends the request to lock the file so that nobody can make changes on that file.
- **releaseFile():** Sends the request to release the file so that other people can lock it.
- **checkFileIn():** Sends the request to check the file in, in other words to send the file contents in the local machine to server to be stored.
- **checkFileOut():** Sends the request to check the file out, in other words to get the file contents from the server to be stored in the local machine.

4. MODELLING

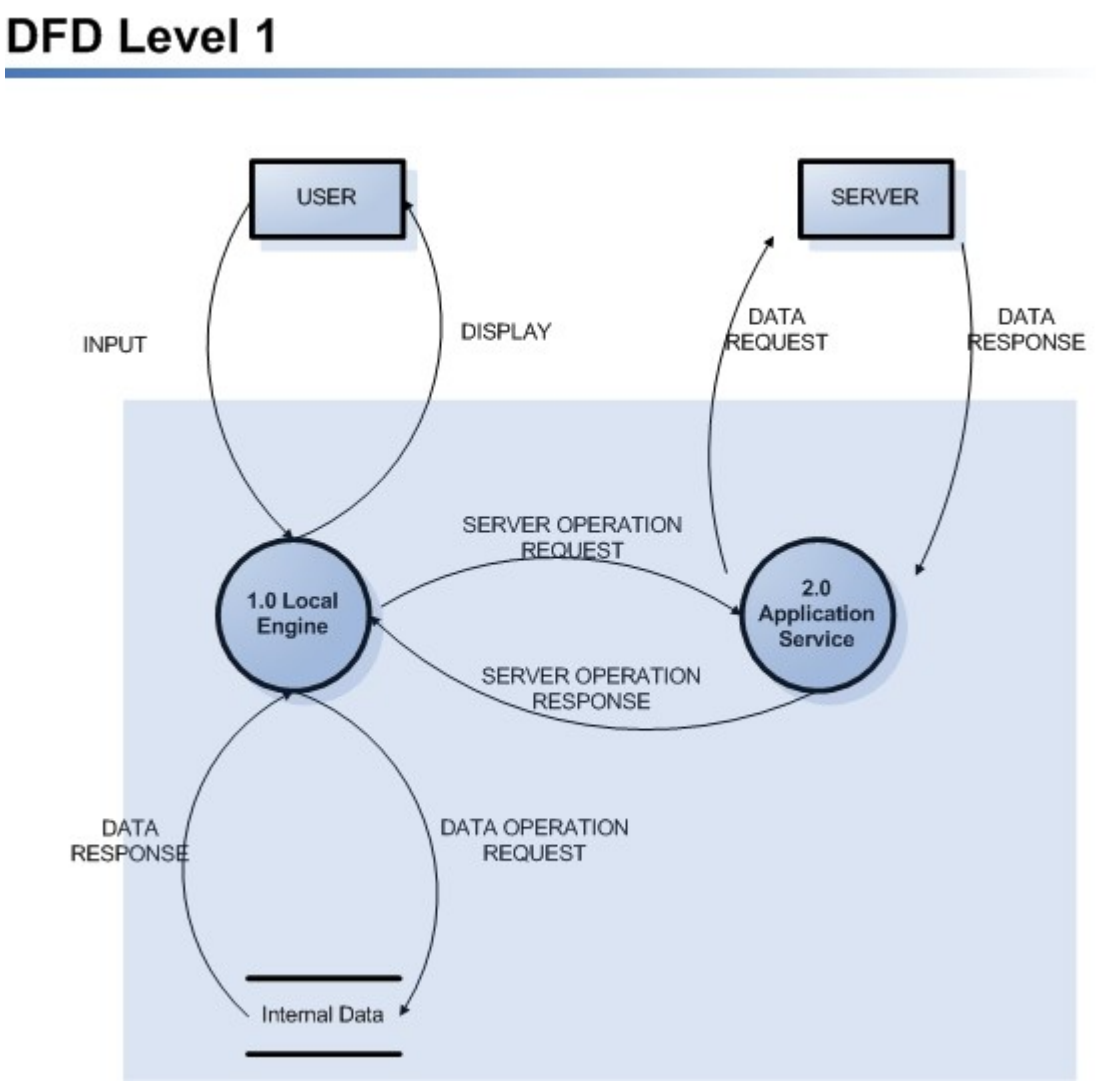
4.1 Functional Modeling

4.1.1 Data Flow Diagrams (DFD)

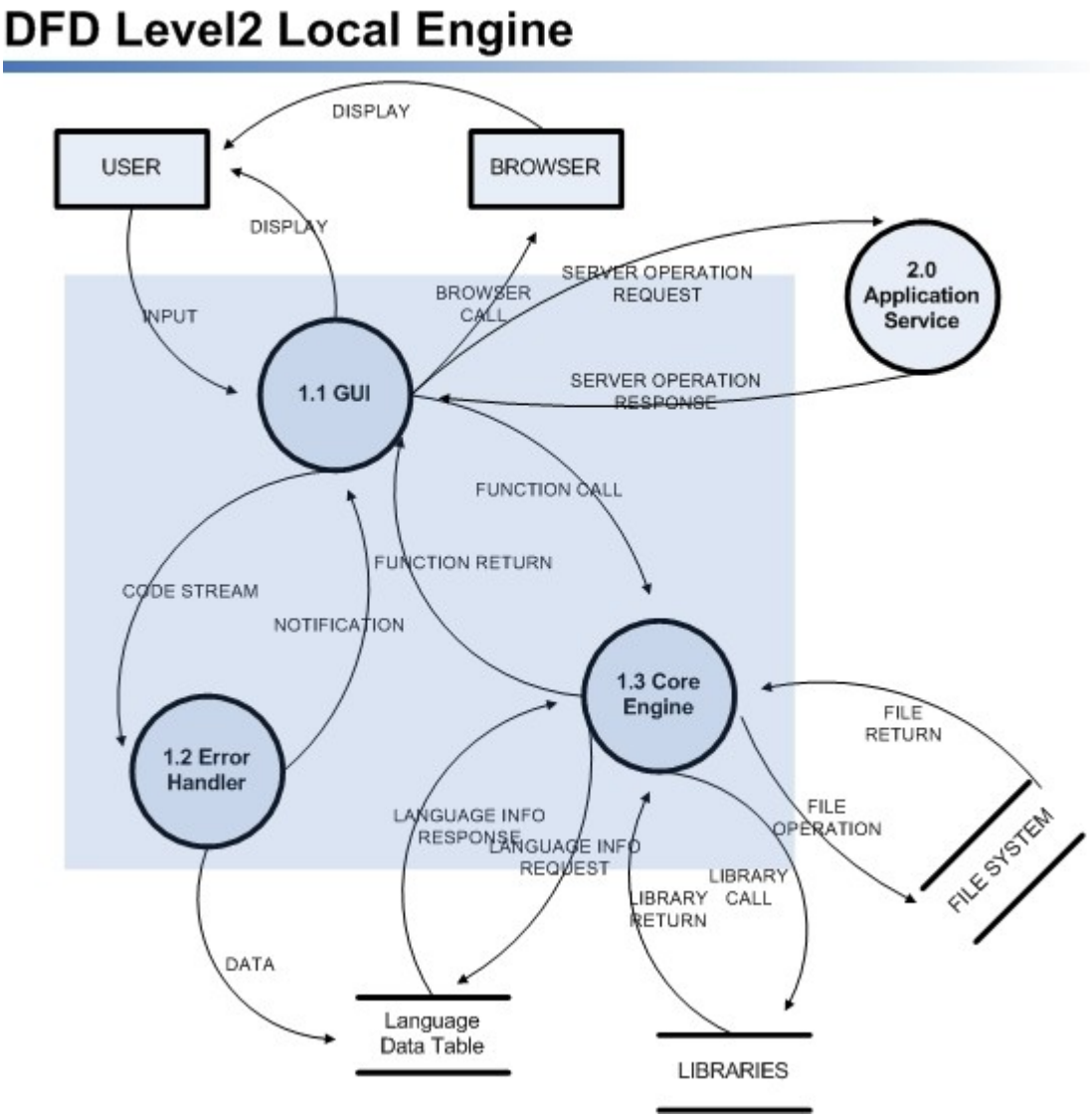
4.1.1.1 DFD Level 0



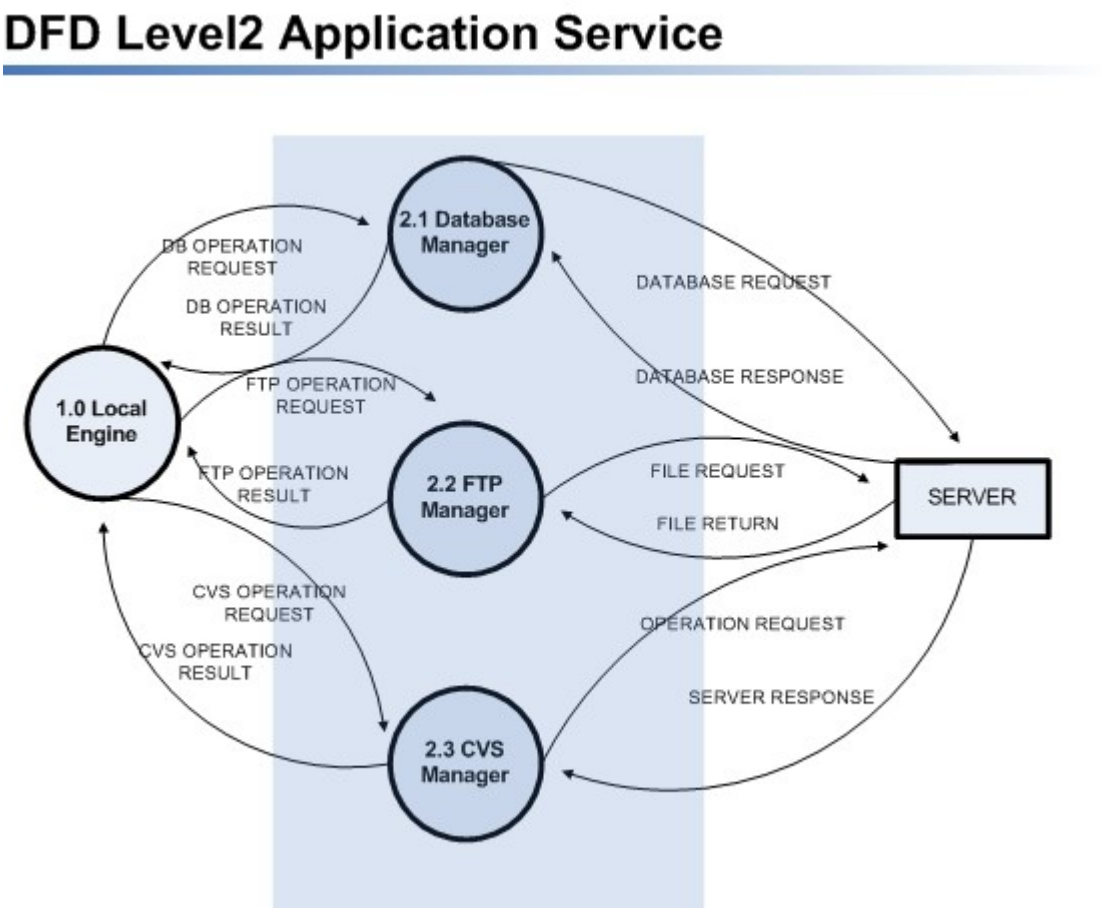
4.1.1.2 DFD Level 1



4.1.1.3 DFD Level 2: Local Engine



4.1.1.4 DFD Level 2: Application Service



4.1.2 Process Specifications (PSPEC)

0.0 CEYLAN Ajax Development Studio

CEYLAN Ajax Development Studio is our whole project. It is in communication with user and server. It has two main components; 1.0 Local Engine and 2.0 Application Server.

1.0 Local Engine

Local Engine is the part where most of the basic properties that our

development studio have are handled. Local Engine is in communication with user and Application Service. Moreover, it is responsible from constructing the internal data which is kept for auto completion. It is consisting of three sub modules; 1.1 GUI, 1.2 Error Handler, 1.3 Core Engine.

1.1 GUI (**G**raphical **U**ser **I**nterface)

Our graphics-based user interface incorporates movable frames, icons and mouse actions. The ability to resize or hide application frames and customizable style and font properties are significant advantages of our Graphical User Interface compared to a character-based interface. This process includes all the graphical views provided by our development studio such as menu bar, tool bar, file explorer, tree view for the tags of the documents, area for error reporting, drag & drop components and text editor which includes code, design and split views. Moreover, the interfaces for database, FTP and CVS applications are parts of this sub module. GUI provides interaction between user and system. That is, all requests coming from user are captured by GUI and these requests are either forwarded to the responsible modules or handled in this module. The responses to the requests are displayed via GUI.

GUI collaborates with other modules in order to return response to the user. It sends the code stream to the Error Handler, and gets the information about errors to be reported and also about auto completion. It forwards the file operations requests to the Core Engine and gets the response for display. Moreover, it forwards the server oriented requests to the Application Service and displays the response.

1.2 Error Handler

Error handler takes the written code and makes necessary operation on it. First, it does the lexical analysis of the code and tokenizes the code stream, then parses the code. If there exist errors, it sends notification to GUI in two ways: it may return an error message or it may show the error on

screen by underlining the erroneous parts. The information for highlighting the code also comes from this module. While examining the code, error handler stores some important information to LANGUAGE DATA TABLE which will be used by auto completion process and other processes that need language specified information.

1.3 Core Engine

Core Engine controls operations on local machines such as saving file, opening file, creating workspace etc. In addition to these trivial tasks it is responsible for “auto completion” which is invoked via pressing ctrl+space. Debugging operations are also included in this module. This sub module is in communication with GUI, Language Data Table, Libraries and the File System. It gets the request coming from GUI, takes the requested action and returns the response to GUI module.

2.0 Application Service

CEYLAN allows users to make several database connections and monitor them. Moreover, it incorporates CVS and FTP support. This module is in communication with Local Engine and Server. Mainly, it gets the request from Local Engine, processes the request and forwards to the Server. The response coming from the server is again processed by Local Engine and sent to Local Engine for display. Application Service is consisting of three sub modules; 2.1 Database Manager, 2.2 FTP Manager and 2.3 CVS Manager.

2.1 Database Manager

Database Manager is responsible from database operations. User will enter the information needed to establish the connection via a connection wizard. When the connection is established, the user will be able to see database items and make changes on them via console or user interface. This sub module is the bridge between Local Engine and Database Server.

2.2 FTP Manager

FTP Manager is responsible from file transfer operations. Like Database Manager, it gets the information needed to establish the connection from user via a form. And with the advantage of having graphical interface support for file transfer, FTP manager saves the user from dealing with commands in a non-graphical console.

2.3 CVS Manager

CVS Manager is responsible from CVS operations. It will ease the job of the programmer working in a group. User enters the information necessary for server connection via a form. After the connection is established, user will be able to see the file and version information of them. S/he requests the action to be performed through a graphical interface.

4.1.3 Data Dictionary

Name	INPUT
Description	This user provided data establishes one part of the interaction. It consists of code or action request or information entered through forms.
Output from	User
Input to	GUI 1.1
Format	[keyboard action | mouse action]

Name	DISPLAY
Description	This is the graphical user interface provided to user by GUI and browser. Actually, it is the response of the action requests returned to user.
Output from	GUI 1.1 Browser
Input to	User
Format	[java swing objects]

Name	CODE STREAM
Description	User code written in text area.
Output from	GUI 1.1
Input to	ERROR HANDLER 1.2
Format	[HTML | JAVASCRIPT | CSS | PHP]

Name	NOTIFICATION
Description	[LEXER ERROR | PARSER ERROR | HIGHLIGHT INFORMATION]
Output from	ERROR HANDLER 1.2
Input to	GUI 1.1
Format	Formatted [String]

Name	DATA
Description	Data produced in lexical analysis and parser stages
Output from	ERROR HANDLER 1.2
Input to	LANGUAGE DATA TABLE
Format	Formatted [String]

Name	BROWSER CALL
Description	The action request (coming from user through GUI) which is forwarded to Browser
Output from	GUI 1.1
Input to	Browser
Format	Action

Name	DATA REQUEST
Description	Database, FTP or CVS related operation invoke
Output from	GUI 1.1
Input to	APPLICATION SERVICE 2.0
Format	[Formatted [String] ׀ Action]

Name	DATA RESPONSE
Description	Database, FTP or CVS operation result
Output from	APPLICATION SERVICE 2.0
Input to	GUI 1.1
Format	Formatted [String]

Name	FUNCTION CALL
Description	Function call from user related to file request or auto completion request
Output from	GUI 1.1
Input to	CORE ENGINE 1.3
Format	Formatted [String]

Name	FUNCTION RETURN
Description	Function return from system
Output from	CORE ENGINE 1.3
Input to	GUI 1.1
Format	Formatted [String]

Name	FILE OPERATION
Description	File operation (open /save /load) attempt on local machine
Output from	CORE ENGINE 1.3
Input to	FILE SYSTEM
Format	Formatted [String]

Name	FILE RETURN
Description	File operation return
Output from	FILE SYSTEM
Input to	CORE ENGINE 1.3
Format	Formatted [String]

Name	LIBRARY CALL
Description	Library function and file request
Output from	CORE ENGINE 1.3
Input to	LIBRARIES
Format	Formatted [String]

Name	LIBRARY RETURN
Description	Library function and file call return
Output from	LIBRARIES
Input to	CORE ENGINE 1.3
Format	Formatted [String]

Name	LANGUAGE INFO REQUEST
Description	Language related data request
Output from	CORE ENGINE 1.3
Input to	LANGUAGE DATA TABLE
Format	Formatted [String]

Name	LANGUAGE INFO RESPONSE
Description	Language related data response
Output from	LANGUAGE DATA TABLE
Input to	CORE ENGINE 1.3
Format	Formatted [String]

Name	SERVER OPERATION REQUEST
Description	Server related (Database, FTP or CVS) operation request
Output from	GUI 1.1
Input to	APPLICATION SERVICE 2.0
Format	Formatted [String]

Name	SERVER OPERATION RESPONSE
Description	Server related (Database, FTP or CVS) operation response
Output from	APPLICATION SERVICE 2.0
Input to	GUI 1.1
Format	Formatted [String]

Name	DB OPERATION REQUEST
Description	Queries that will executed on database (either for database connection or operations on database items)
Output from	LOCAL ENGINE 1.0
Input to	DATABASE MANAGER 2.1
Format	Formatted [SQL Query]

Name	DB OPERATION RESULT
Description	Query results returned from database
Output from	DATABASE MANAGER 2.1
Input to	LOCAL ENGINE 1.0
Format	Formatted [String]

Name	DATABASE REQUEST
Description	Queries that are sent to database (either for database connection or operations on database items)
Output from	DATABASE MANAGER 2.1
Input to	SERVER
Format	Formatted [SQL Query]

Name	DATABASE RESPONSE
Description	Query results returned from database
Output from	SERVER
Input to	DATABASE MANAGER 2.1
Format	Formatted [String]

Name	FTP OPERATION REQUEST
Description	File transfer operation request & information entered by user for connection
Output from	LOCAL ENGINE 1.0
Input to	FTP MANAGER 2.2
Format	Formatted [String]

Name	FTP OPERATION RESULT
Description	File transfer operation results processed by FTP Manager
Output from	FTP MANAGER 2.2
Input to	LOCAL ENGINE 1.0
Format	Formatted [String]

Name	FILE REQUEST
Description	File request and requested file information sent to Server
Output from	FTP MANAGER 2.2
Input to	SERVER
Format	Formatted [SQL Query]

Name	FILE RETURN
Description	Results returned from server, either requested file or failure report
Output from	SERVER
Input to	FTP MANAGER 2.2
Format	Formatted [String]

Name	CVS OPERATION REQUEST
Description	CVS operation request & information entered by user for connection
Output from	LOCAL ENGINE 1.0
Input to	CVS MANAGER 2.3
Format	Formatted [String]

Name	CVS OPERATION RESULT
Description	CVS operation results processed by CVS Manager
Output from	CVS MANAGER 2.2
Input to	LOCAL ENGINE 1.0
Format	Formatted [String]

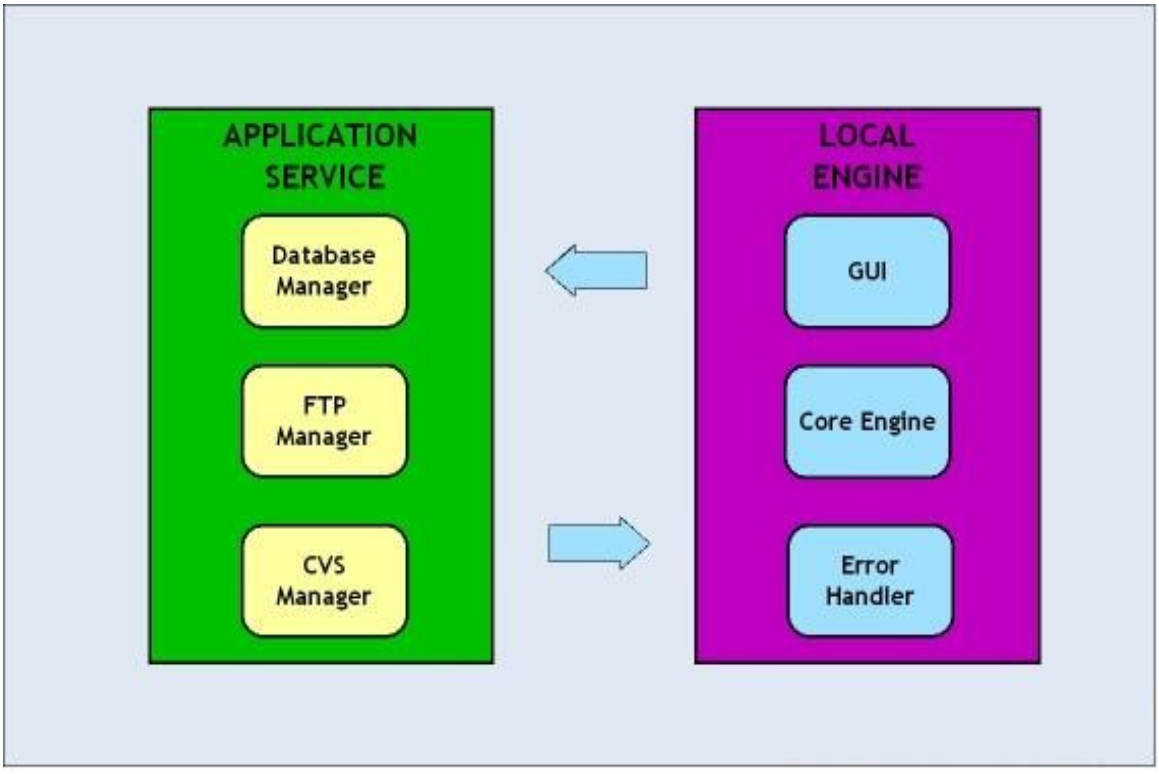
Name	OPERATION REQUEST
Description	CVS operation request and information related to requested operation information sent to Server
Output from	CVS MANAGER 2.2
Input to	SERVER
Format	Formatted [String]

Name	SERVER RESPONSE
Description	Results returned from server (file and file owner information)
Output from	SERVER
Input to	CVS MANAGER 2.2
Format	Formatted [String]

5.0 Big Picture

We have designed our software Ceylan AJAX Development Studio as two main modules: Application Service and Local Engine. This choice is made in order seperate client side related and server side related activities to be able to implement them modulary. This modularity will also provide us with the opportunity of plugging updates, new features to the software. Data passing between these will be established via API's provided by each one.

Details of the modules will be explained below.



5.1 LOCAL ENGINE

Local engine is resposible from all the client side related actions. Desktop

application properties of our program are provided by this module. Some of these are file and I/O operations, debugging, display, etc. It has three modules to carry out these actions.

5.1.1 GUI

This module is designed in order to take all the graphical display related actions out from the functionality of our software and can be tought of as a layer between the system functionalities and the user. This module is the graphical bridge between user and the system. User interacts with system via this sub-module. It includes editor and all the other graphical display such as menu bar, tree view, error messages, etc.

5.1.2 Core Engine

Core engine is the central of our program that manages the overall system activities. Our program has many seperate components. There is an editor waiting for input from user, error handler working upon request at any time, standart file oprations performed by both users and system functions, database manager, FTP manager, CVS manager invoked by user at any time. All those actions have to work synchronously and without crashing with each other. Sometimes they are affected or changed by other ones. Core engine is responsible from this synchronization. It establishes the link and information pass between modules.

Apart from that, core engine performs main operations in the system such as file system actions, auto completion functions, debugging, etc.

5.1.3 Error Handler

This submodule is mainly responsible from lexing and parsing. It provides editor with highlight info. Error recovery policies are also specified by it.

5.2 APPLICATION SERVICE

Application service is designed in order to perform all connection related actions. User can invoke three main connection related functionalities using this submodule. During her/his code development s/he may want to connect her/his database. S/he may view or update her/his tables, execute a query and see the results. Or s/he may want to make file transfer from a remote machine. Additionally s/he may want to use CVS within the system. Our application service module mainly provides these features.

Application Service module does not have to be invoked unless the user makes operations concerning file transfer, database or CVS.

5.2.1 Database Manager

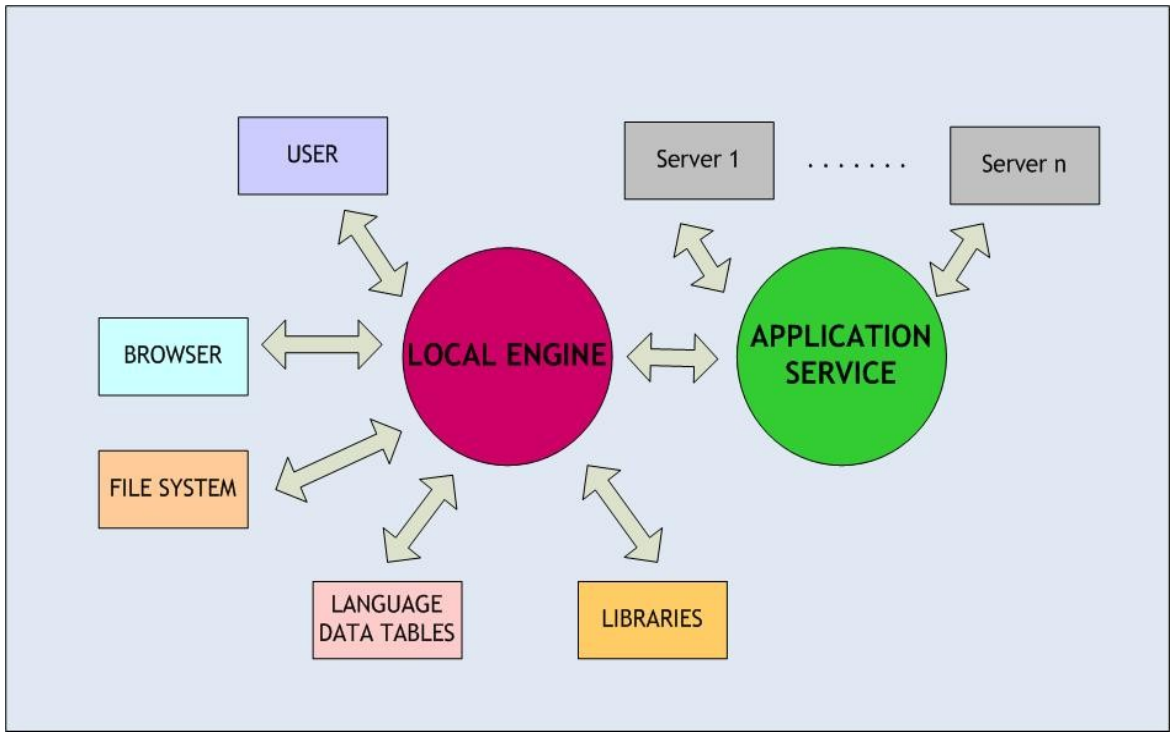
This sub-module takes data request either from user code or query execution tool. It executes query on database and returns the results accordingly either to query execution tool or user code. If the request is coming from code the results are back embedded into code. Or if the request is coming from query execution tool the result is dispalyed back in query execution tool in a prespecified format.

5.2.2 FTP Manager

This submodule is responsible from all file transfer operations. It makes connection to remote machines for transfer.

5.2.3 CVS Manager

According to the information taken from user, this submodule makes the settings needed for CVS operations.



This figure shows data flow between our modules and external entities.Local Engine module establishes the information link between application service, browser, file system, language data tables, libraries and users.

Application side takes data from client side makes the necessary operations on them and passes it to server/s. The information returned from server/s are sent

to client side after application server operates them.

6.0 User Interface Design

6.1 GUI Design Principles

We have designed our GUI, according to the following 7 basic principles;

- Don't be different
- Keep it simple
- Look professional
- Be direct
- Be consistent
- Give cues and feedback
- Forgive mistakes

1) Don’t be different

Humans can remember 7 different items independent from each other during short intervals. So a GUI should reduce memorization as much as possible. If one user is accustomed to one of the operating systems such as Windows or Linux or to one of the applications such as Eclipse or MS Visual Studio, the system should be suitable for the user to adapt to easily. The system should not perform unexpected behaviour. For instance, a user expects a file menu at the bottom-left corner and sub menus such as New, Open, Save, Save As. Otherwise it would be too complicated for the user since s/he has to memorize a lot of items.

For instance we tried to make our application look similar to a common IDE like Eclipse so that user does not have to take training or spend additional effort to be able to use our IDE.

2) Keep it simple

A user should be able perform his/her actions by simple acts such as one click or two selects. S/he should also be able to understand what is going behind and feel that s/he is in the control.

Windows for different actions should be separated from each other. For example preferences and working elements should not be in the same window. It would be too cluttered to use.

According to the Principle of Progressive Disclosure until it is needed, complexity should be hidden. When a user opens “Preferences…” menu which is almost applicable in most of the IDEs, s/he does not see everything every thing once. They are grouped according to what the user is probably trying to do-change the appearance, set security level etc.[1]. We consider this principle while designing menus that are routing to complicated actions.

For instance, when s/he wants to open a file from a project it is enough to click on the file form project tree-just one click.

3) Look professional

Appearance has a considerable importance for the system. Before every thing that can be done via GUI, user first sees the GUI and makes an idea of the system. Suppose s/he is planning to do an action about debugging. If s/he got the idea that the system is unprofessional, even if your system works properly, s/he will think that it won’t be able to perform the work you are intended to do.

Deciding on the elements such as size, alignment, fonts and colour is very important. For instance using capital letters annoys people.

In our project, we have chosen a professional look and feel option to give a professional view to the user.

4) Be direct

Buttons or menu icons should have small and descriptive words. The user should be able to understand by just looking at the keyword assigned to a button or menu.

We also avoid using technical terms like key, transaction, record as much as possible since all of the users are not professionals having knowledge about every technical term.

In our project, we kept our descriptive names for menus as much as short and understandable.

5) Be consistent

Your representations (which are usually icons) should be consistent with real world. For example when s/he sees a question mark icon  s/he should know that upon pressing the icon s/he will be directed to a state where s/he will get information about his/her needs.

GUI should also be consistent internally. If there are two text areas which look almost exactly the same, one of them should not be editable while the other is not. This would be very confusing for the user.

All of our icons are consistent with real world objects according to the action they perform upon clicking.

6) Give cues and feedback

The user should get a feedback for every action s/he performs to be sure whether s/he had succeeded or not. A progress bar at the bottom while the action you invoked is being processed is a good example to such kind of a feedback. Passive items during an interval should be invisible or actionless to imply that user can not activate a function using that item.

GUI should also provide users with cues. For example when s/he moves the

mouse over an icon and wait for a while, there should appear a small descriptive text near the mouse to give an idea to the user what the icon is about.

For instance, when a user establishes a connection to a database, s/he will be provided with a message that the connection succeeded.

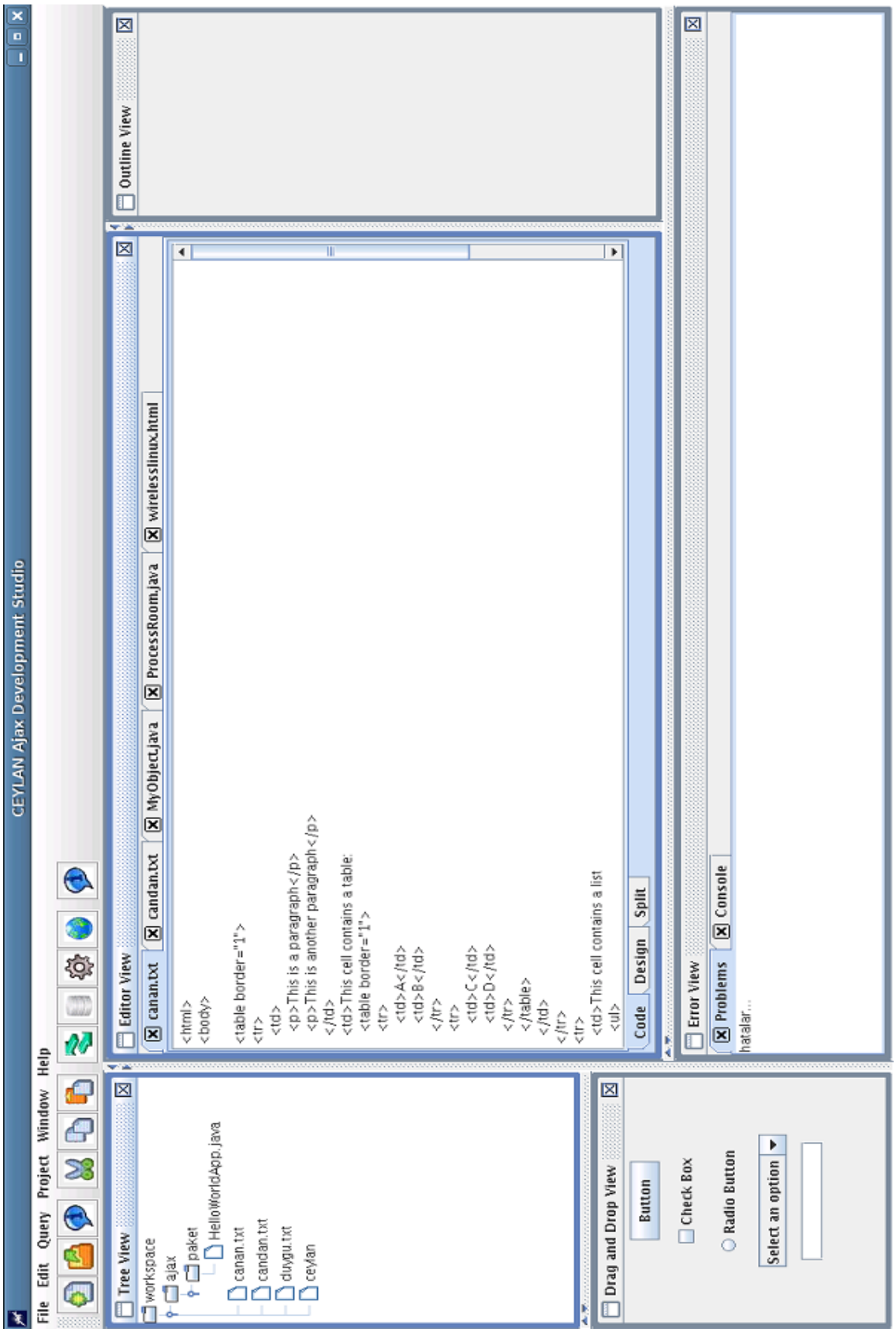
7) Forgive mistakes

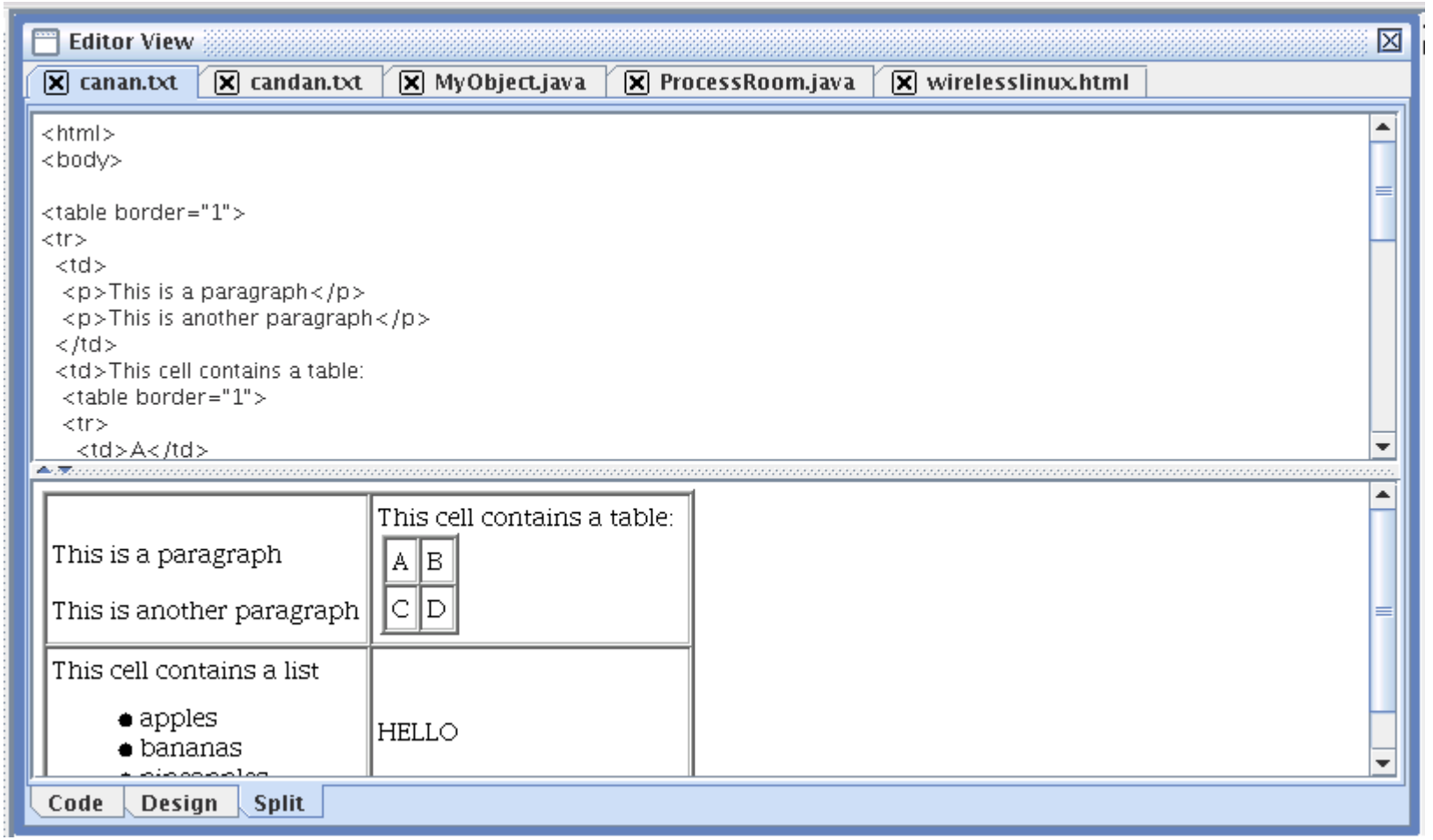
There three ways to handle user errors:

- Prevent users from making mistakes.
- Let the user do what s/he wants while providing him/her with error messages. But also help them to recover from errors.
- Let the user to do what s/he wants and show him/her the results. But give them opportunity to undo what s/he had done.

In our system when a user tries to exit IDE without saving his/her file, there will appear a message asking him to save his/her file.

6.2 Screenshots and Actions




Editor View
canan.txt
candan.txt
MyObject.java
ProcessRoom.java
wirelesslinux.html
<html>
<body>
<table border="1">
<tr>
<td>
This is a paragraph
This is another paragraph
</td>
<td>This cell contains a table:
<table border="1">
<tr>
<td>A</td>
This is a paragraph
This is another paragraph
This cell contains a table:
A
B
C
D
This cell contains a list
apples
bananas
HELLO
Code
Design
Split

Our general view of Ceylan Ajax Development Studio is a modern GUI with a professional look and feel option. Since it has been designed according to the principles stated above in 4.1 it is highly user friendly.

In “Editor” view, user can see the files that he/she opened. Each file comes with its own tab including its three different view options.

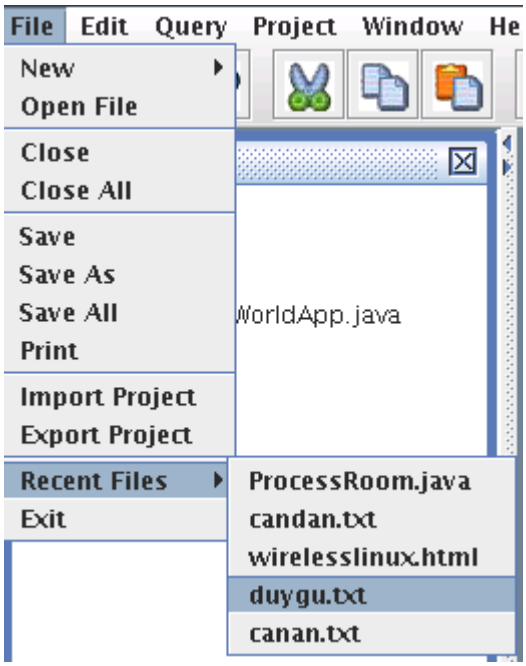
Code View : In code view, user can write his/her code and see his/her code indented and highlighted according to the type of his/her file.

Design View : In design view, user can view his code evaluated as it is in external browser.

Split View : In split view, user can see his/her file both as code and design.

6.2.2 MenuBar

6.2.2.1 File Menu



New : By clicking on the “New” menu item, user can create a new project, or a new file in CSS/PHP/HTML format. The file is opened in a new tab with its code, design and split views.

Open File : By clicking on the “Open File” menu item, there comes a file chooser pop-up menu. User can choose one of his/her existing files. If the file is already open, it will be focused. Otherwise the file is opened in a new tab with its code, design and split views.

Close : By clicking on the “Close” menu item, user can close the file which he/she is working on. If there has been a change in the file from its last save, a popup comes out for asking to save the changes on that file.

Close All : By clicking on the “Close All” menu item, user can close all the files that are open. If there has been changes in the files from their last saves, a popup comes out for asking to save the changes on that files.

Save : By clicking on the “Save” menu item, user can save the file that he/she created or changed. Default save directory is the workspace of the related project. If the file doesn't belong to a project, it will be saved to workspace directory.

Save As : By clicking on the “Save As” menu item, user can save the file that he/she created or changed with a different name, extension. Default save directory is the workspace of the related project. If the file doesn't belong to a project, it will be saved to workspace directory.

Save All : By clicking on the “Save All” menu item, user can save all the files that he/she created or changed. Default save directories are the workspaces of the related projects. Files which don't belong to a project will be saved to workspace directory.

Print : By clicking on the “Print” menu item, user can print the file he/she is working via printer currently attached to the computer.

Import Project : By clicking on the “Import Project” menu item, there comes a project chooser popup menu. User can choose one of his/her existing projects or other external projects and can import them to his/her workspace.

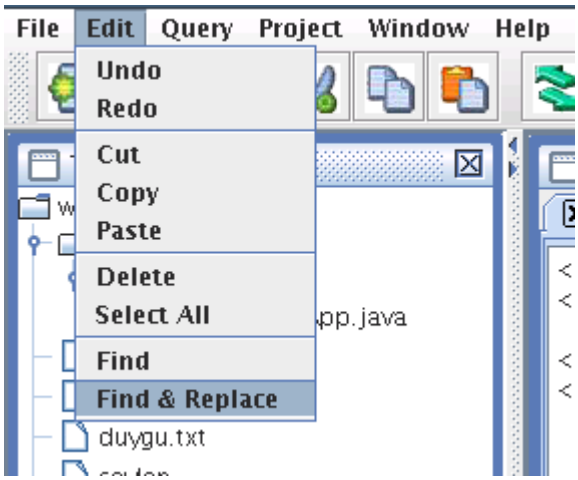
Export Project : By clicking on the “Export Project” menu item, there

comes a project chooser popup menu. User can choose and export one of his/her existing projects to another workspace on the computer.

Recent Files : By clicking on the “Recent Files” menu item, user can see the last 5 opened files. User can open the file which he/she wants to. If the file is already open, it will be focused. Otherwise the file is opened in a new tab with its code, design and split views.

Exit : By clicking on the “Exit” menu item, user can exit from Ceylan Ajax Development Studio. If there are files which aren't saved, user will be warned to save those files.

6.2.2.2 Edit Menu



Undo : By clicking on the “Undo” menu item, user can take reverse action on what s/he had done.

Redo : By clicking on the “Redo” menu item, user can take forward action on what s/he had done.

Cut : By clicking on the “Cut” menu item, user can cut the selected item.

Copy : By clicking on the “Copy” menu item, user can copy the selected item.

Paste : By clicking on the “Paste” menu item, user can paste the item that is in the buffer to the cursor position.

Delete : By clicking on the “Delete” menu item, user can delete the selected

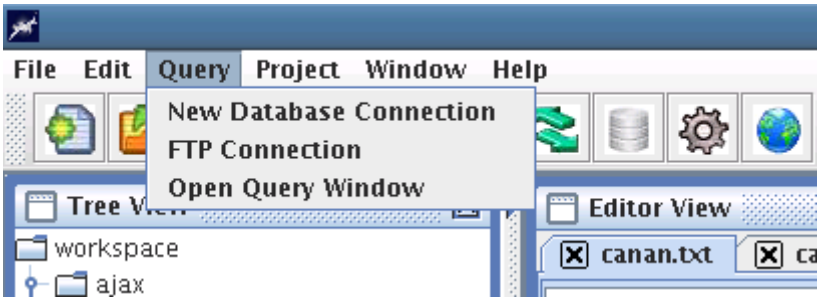
item.

Select All : By clicking on the “Select All” menu item, user can select all the items on the current container.

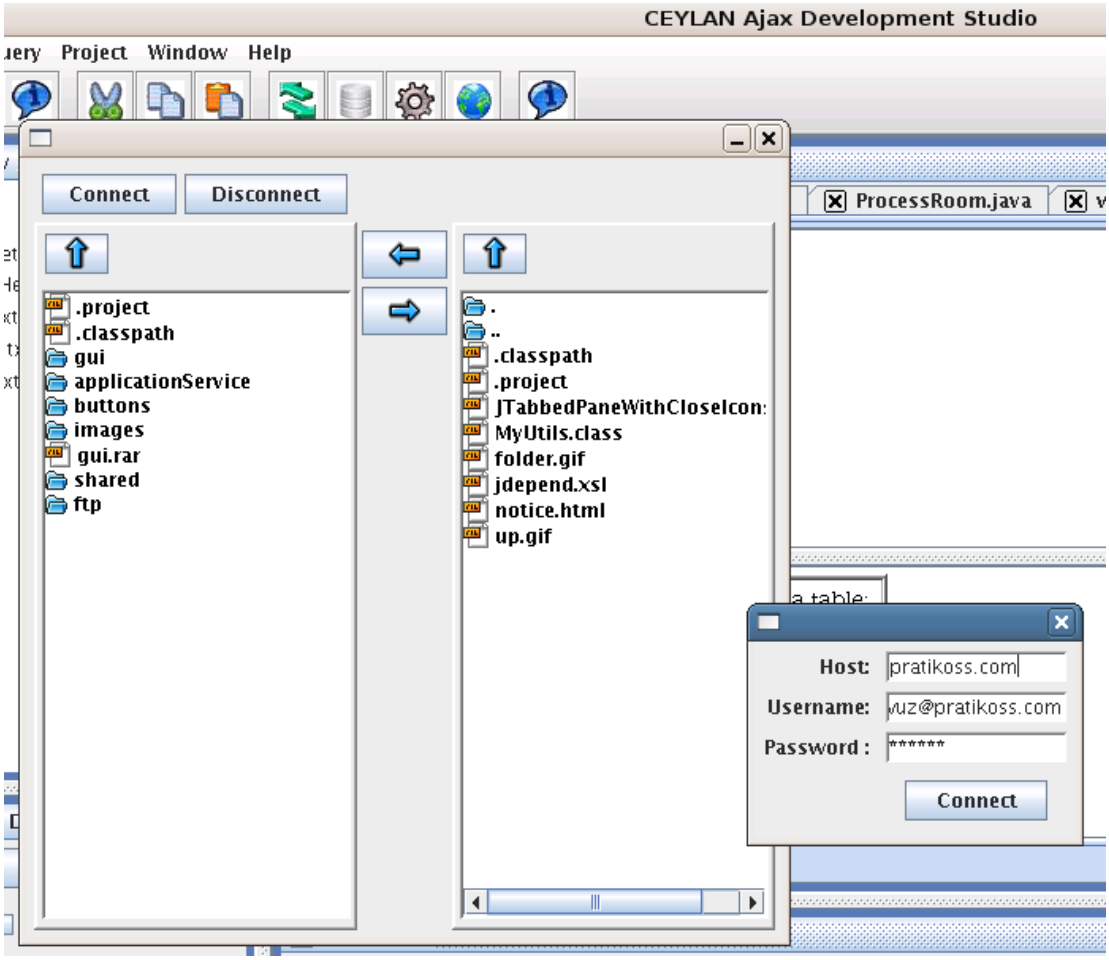
Find : By clicking on the “Find” menu item, there comes out a popup window. In this window, user can enter the keyword s/he is searching for in the current document. The current document is parsed and the keyword is shown to user as highlighted.

Find & Replace : By clicking on the “Find & Replace” menu item, there comes out popup window. In this window user can enter the keyword s/he is searching for and the keyword s/he wants to put instead in the current document. The current document is parsed and the keyword is replaced with the one which user wants to.

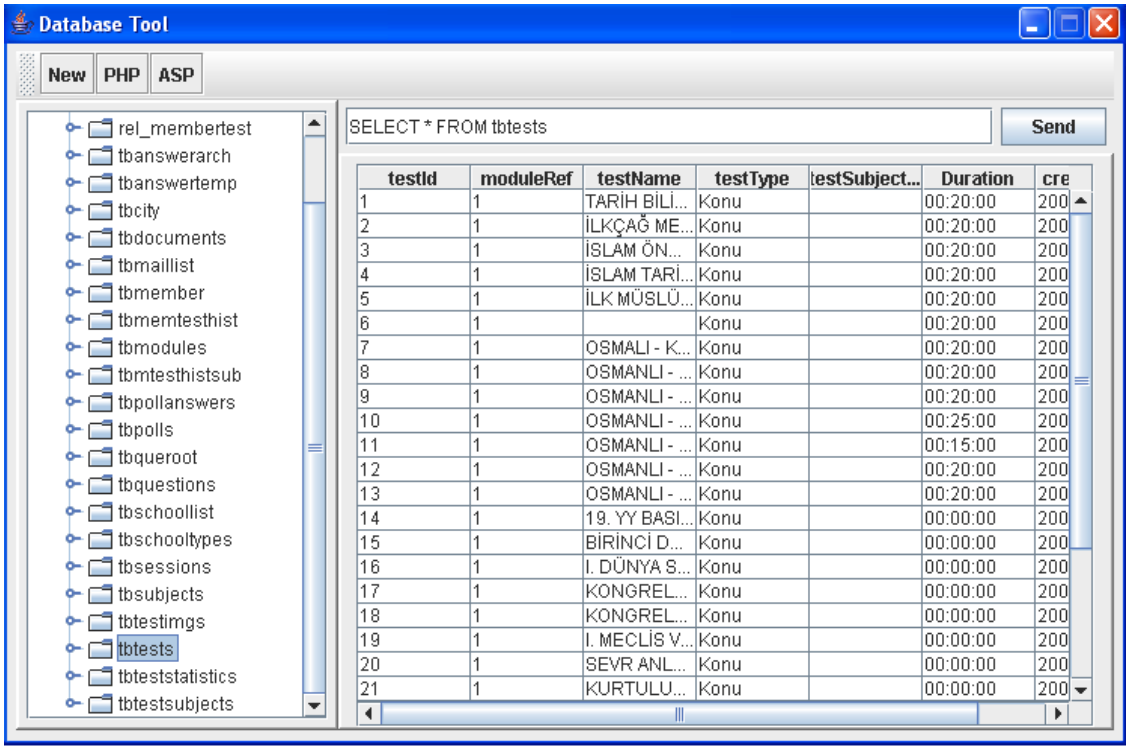
6.2.2.3 Query Menu

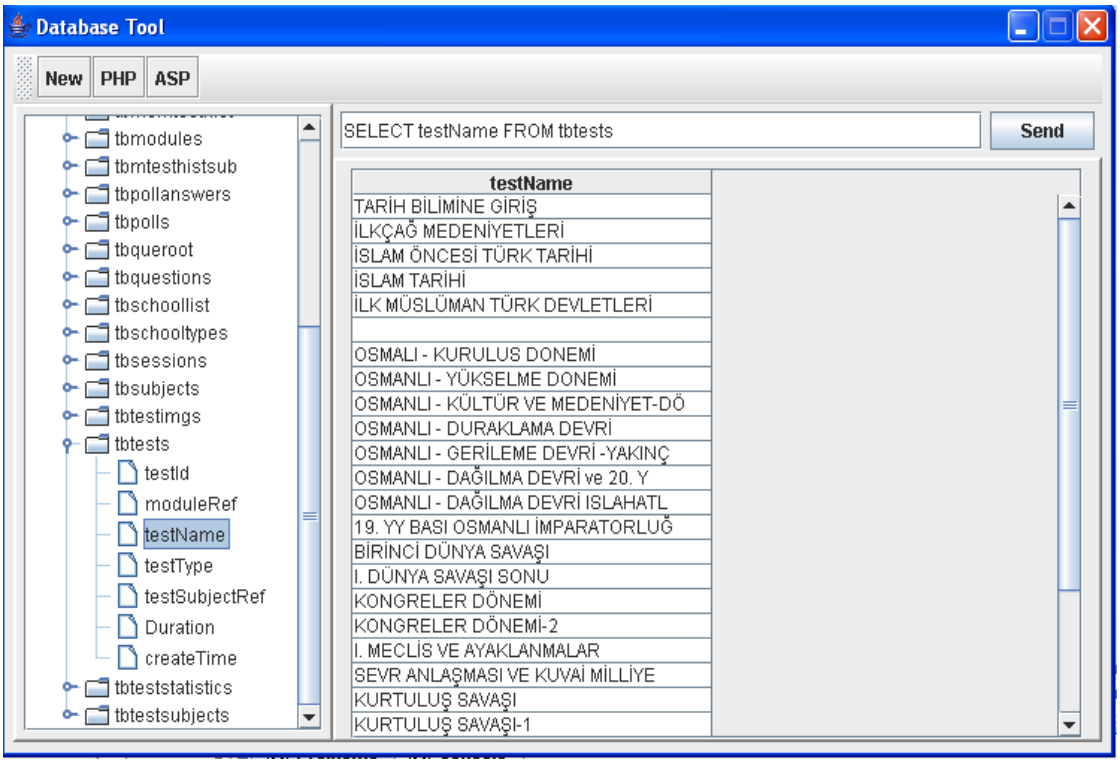


New Database Connection : By clicking on the "New Database Connection" menu item, there comes out popup window. In this window, in order to connect a database, user should enter the related fields which are "Connection Name", "Hostname/IP Address", "Port", "Database Name", "Username", "Password", and "Driver".



FTP Connection : By clicking on the “FTP Connection” menu item, there comes out a popup window. In this window, user can create FTP connection and upload and download files.

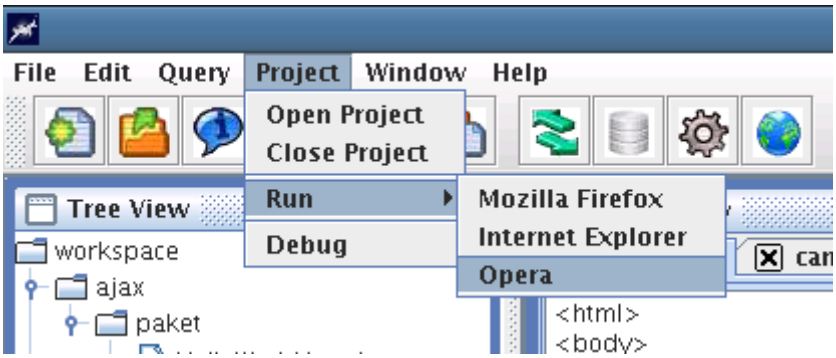




Open Query Window : By clicking on the “Open Query Window”, menu

item, there comes out popup window. In this window, user can see the databases which he/she connected to and he/she write queries and see the results of these queries.

6.2.2.4 Project Menu



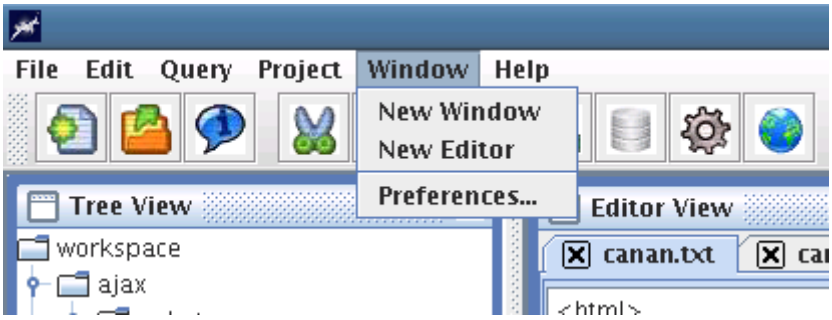
Open Project : By clicking on the “Open Project” menu item, there comes out a popup window which shows the existing projects of the user. User can select one of his saved projects.

Close Project : By clicking on the “Close Project” menu item, user can close the project which he/she is currently working on.

Run : By clicking on the “Run” menu item, the code will be evaluated(parsed, interpreted), and will be shown on the browser. User can invoke “Run” operation with different browsers which are “Mozilla Firefox”, “Internet Explorer”, and “Opera”.

Debug : By clicking on the “Debug” menu item, Mozilla add-on debugger Firebug will be opened, until we implement our own debugger.

6.2.2.5 Window Menu

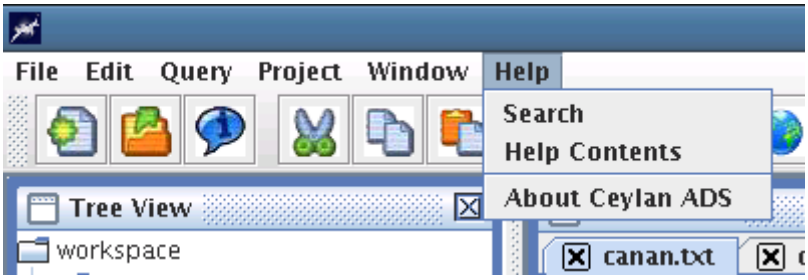


New Window : By clicking on the “New Window” menu item, a new window just like the whole frame will be opened.

New Editor : By clicking on the “New Editor” menu item, a new window just like the editor frame will be opened in the same split area.

Preferences : By clicking on the “Preferences” menu item, there comes out a popup window where user can setup the window properties.

6.2.2.6 Help Menu



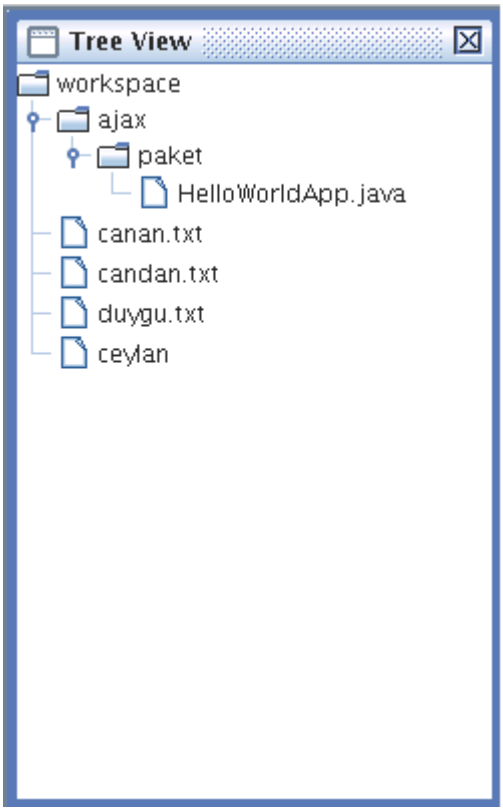
Search : By clicking on the “Search” menu item, there comes out a popup window which contains the help topics will be opened.

Help Contents : By clicking on the “Help Contents” menu item , there comes out a popup window (the same one stated above) which includes the search indexes near help topics will be opened.

About Ceylan ADS : By clicking on the “About Ceylan ADS” menu item,

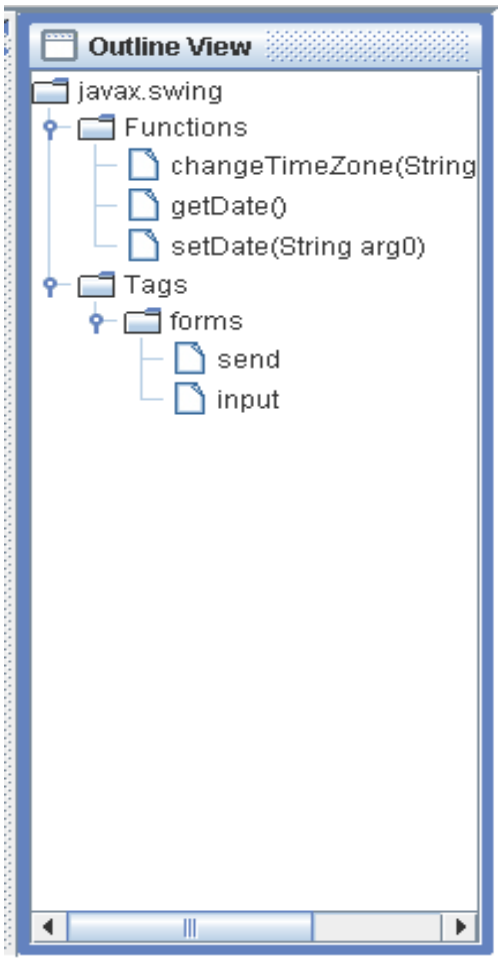
general information about Ceylan Ajax Development Studio will be displayed on a small display window.

6.2.3 Project Workspace



In “Project Workspace” view, user can view his/her all files in his/her workspace as tree view. When s/he clicks on one of his/her files on the tree, it is opened in editor frame with its own tab including code, design and split views unless it is already open. Otherwise, desired file will be focused on the editor view.

6.2.4 Outline



In “Outline” view, the tags of the file that the user is currently working on is displayed on outline frame. For each file that is in workspace, its tags can be displayed on different tabs.

6.2.5 Error

The errors or warnings in user's project will be displayed in problems tab. In console tab, the fields printed by the user will be viewed.

6.2.6 Drag & Drop

In “Drag & Drop” view, there will be auto-generated components such as, list-box, button, text field. User can drag these components and drop them to “Design” view of the related file. Code of these components will be automatically generated in the “Code” view.

7.0 Testing

7.1 Unit Testing

GUI:

User mostly interacts with GUI, so we have to be careful about the issues about the GUI. Even a small error at GUI might cause fatal errors. Besides these, GUI is very important for the user to choose our program, find it powerful, friendly user and stable. So GUI has very big importance. To reduce the probability of failure, we should apply as much as we can. We will do tests for each of the operations which are done through GUI. In addition to these, we will test every menu with its sub menus. We also have some keyboard events so we try them as

well. To test these, we try different cases for them.

Error Handler:

The important problems of error handler would be those: it may not give true errors, it may not found some errors and it may produce wrong data table and then cause different errors for different modules. So in order to test this module, we will try different cases. For example, we wrote syntactically wrong codes and test it to see whether it gives true errors on true lines.

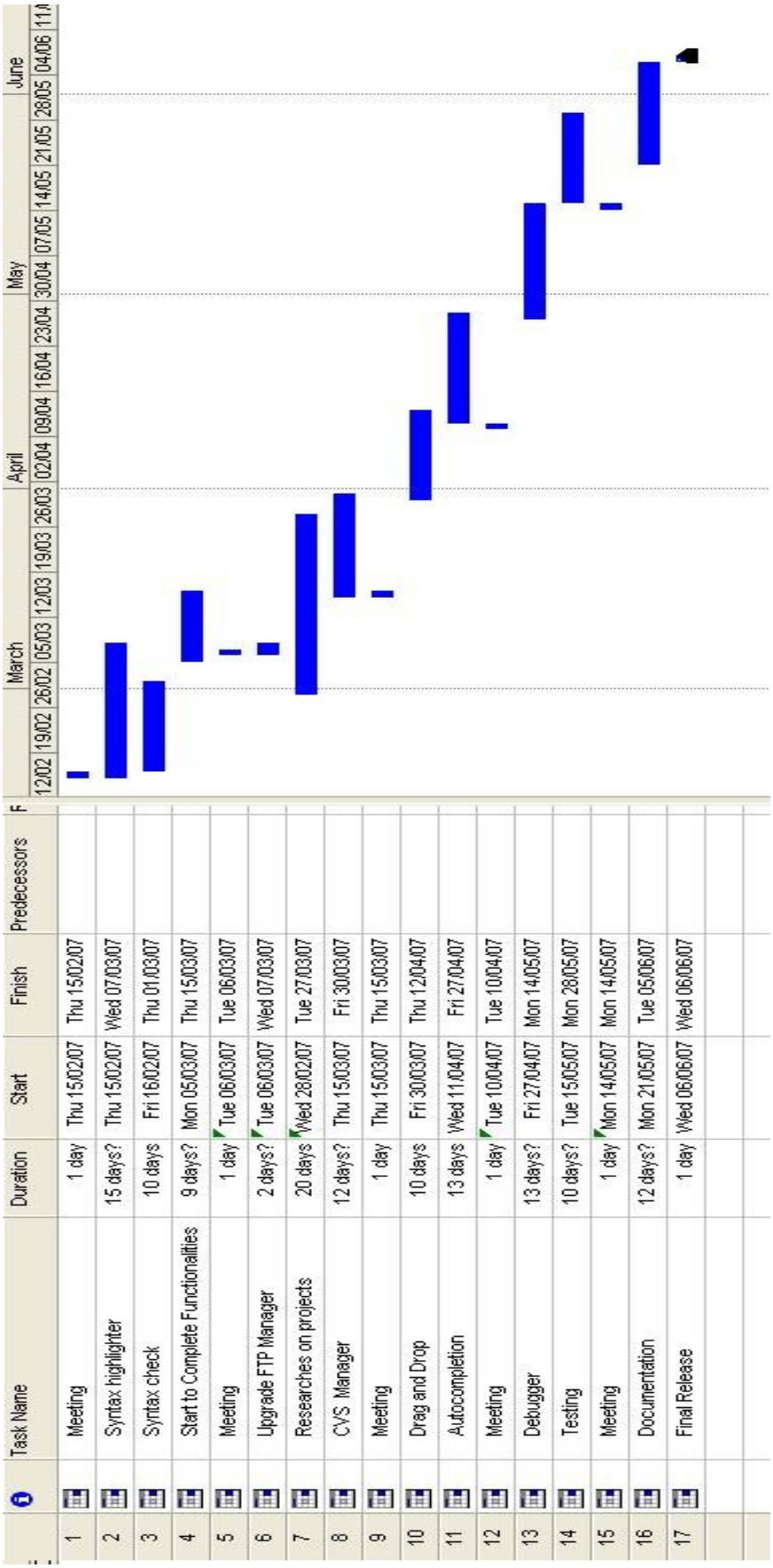
Local Engine:

In order to understand whether our project makes file operations true, we should test this part. We try the basic operations as saving, opening …etc, then we could see the result.

Application Service:

At this part, we would test our operations about database or connecting to other servers. We try some queries in order to see it works or not. Besides these, we try to see the tables after connect to a database.

8.0 Gantt Chart



	Task Name	Duration	Start	Finish	Predecessors
1	Meeting	1 day	Thu 15/02/07	Thu 15/02/07	
2	Syntax highlighter	15 days?	Thu 15/02/07	Wed 07/03/07	
3	Syntax check	10 days	Fri 16/02/07	Thu 01/03/07	
4	Start to Complete Functionalities	9 days?	Mon 05/03/07	Thu 15/03/07	
5	Meeting	1 day	Tue 06/03/07	Tue 06/03/07	
6	Upgrade FTP Manager	2 days?	Tue 06/03/07	Wed 07/03/07	
7	Researches on projects	20 days	Wed 28/02/07	Tue 27/03/07	
8	CVS Manager	12 days?	Thu 15/03/07	Fri 30/03/07	
9	Meeting	1 day	Thu 15/03/07	Thu 15/03/07	
10	Drag and Drop	10 days	Fri 30/03/07	Thu 12/04/07	
11	Autocompletion	13 days	Wed 11/04/07	Fri 27/04/07	
12	Meeting	1 day	Tue 10/04/07	Tue 10/04/07	
13	Debugger	13 days?	Fri 27/04/07	Mon 14/05/07	
14	Testing	10 days?	Tue 15/05/07	Mon 28/05/07	
15	Meeting	1 day	Mon 14/05/07	Mon 14/05/07	
16	Documentation	12 days?	Mon 21/05/07	Tue 05/06/07	
17	Final Release	1 day	Wed 06/06/07	Wed 06/06/07	