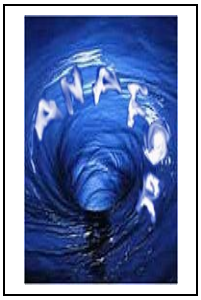
MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

CEYLAN©

AJAX SOFTWARE DEVELOPMENT STUDIO

INITIAL DESIGN REPORT



Group Members:

Saliha ALTUNSOY

Candan CEYLAN

Canan ESKİ

Yavuz GÖKIRMAK

Duygu GÖRGÜN

Table of Contents

1.0 INTRODUCTION 4
1.1Project Title.....4
1.2 Problem Definition.....4
1.3 Statement of scope.....5
1.4 Application Areas of Our Software.....6
2. PROJECT REQUIREMENTS.....7
2.1 Functional Requirements.....7
2.1.1 Widgets such as tree view, toolbars and tabs.....7
2.1.2 Server connectivity.....7
2.1.3 Query Execution.....7
2.1.4 Editing, and Debugging.....8
2.1.5 Work Space.....8
2.1.6 Help menu.....9
2.1.7 Menu Components.....9
2.1.8 Integrated browser.....9
2.1.9 Code and design view.....9
2.1.10 Predefined code generation.....9
2.2 Non-Functional Requirements.....10
2.2.1 User-friendliness.....10
2.2.2 Modularity.....10
2.2.3 Platform Independency.....10
2.2.4 Consistency.....10
2.3 System Requirements.....11
2.3.1 Software Requirements: 11
2.3.2 Hardware Requirements: 11
3. ARCHITECTURAL DESIGN.....12
3.1 Use Case Diagrams.....12
3.2 Sequence Diagrams.....13
3.3 Class Diagrams.....17
3.3.1 GUI 17
3.3.2 Application Service.....20
3.3.3 Error Handler.....22
3.3.4 Local Engine.....24
4. MODELLING.....27
4.1 Functional Modeling.....27
4.1.1 Data Flow Diagrams (DFD).....27
4.1.2 Process Specifications (PSPEC).....33
4.1.3 Data Dictionary.....37
5. USER INTERFACE DESIGN.....42
5.1 GUI Design Principles.....42
5.2 Screenshots and Actions.....46
5.2.1 General Editor View.....46
5.2.2 File Menu View.....47
5.2.3 Edit Menu View.....49
5.2.4 Query Menu View.....50
5.2.5 Project Menu View.....51
5.2.6 Window Menu View.....52

5.2.7 Help Menu View....52
5.2.8 Database View....53
5.2.9 Project Workspace View....54
5.2.10 Outline View....55
5.2.11 ErrorView....55
6. TESTING....56
6.1 Unit Testing....56
7 GANTT CHART....57

1.0 INTRODUCTION

This document aims to provide information about the design process of the problem. During the analysis phase of our project, we investigated the possible problems and in this phase we will try to generate solutions for the design.

Firstly, problem definition and the scope of the project can be found in this document. Detailed version of the 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. 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.

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

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. 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:

- 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.

2.2 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. To supply this, we will put the system behavior on deterministic finite automata in design phase.

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

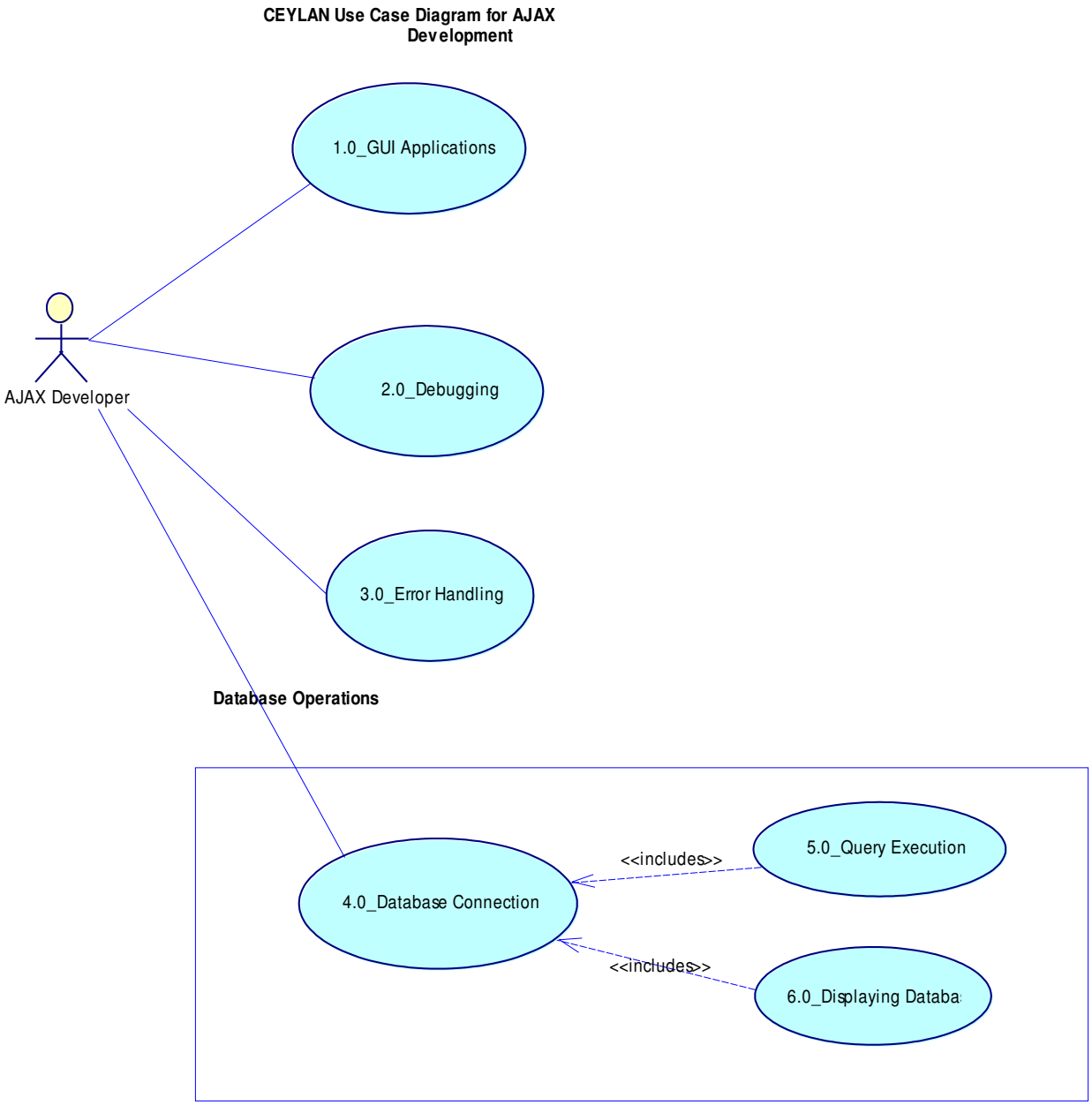
Since our application will be developed in **Java** and will be platform independent, the end user will only need a compatible operating system and a Java **Runtime Environment**. And since we are using web service in the development phase, the user can choose whichever database he/she wants.

2.3.2 Hardware Requirements:

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

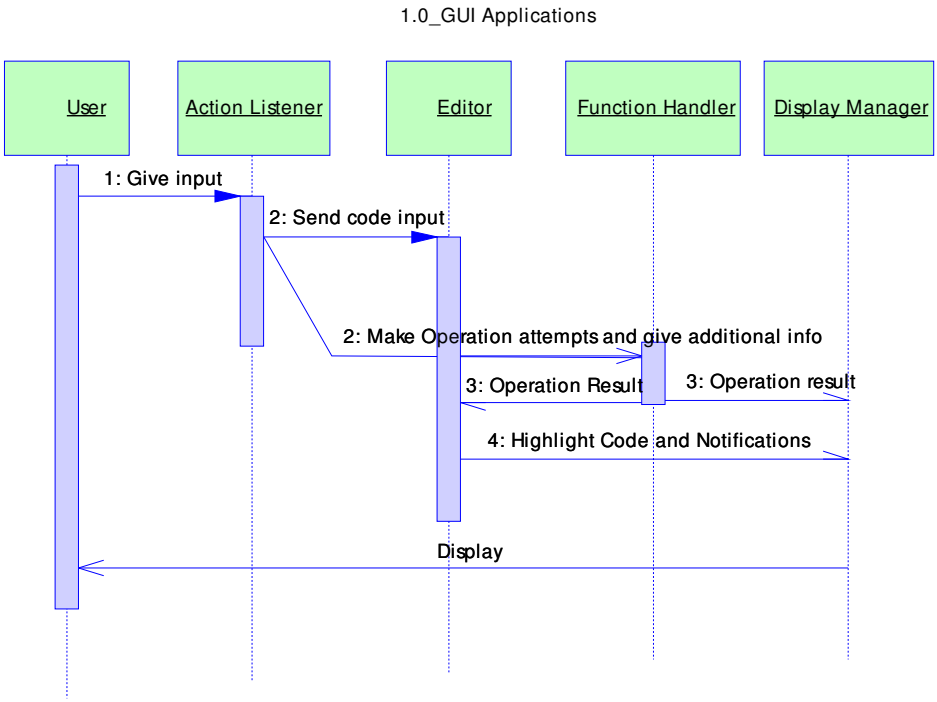
3. ARCHITECTURAL DESIGN

3.1 Use Case Diagrams



3.2 Sequence Diagrams

3.2.1 GUI Applications



This diagram shows the basic operations done by GUI:

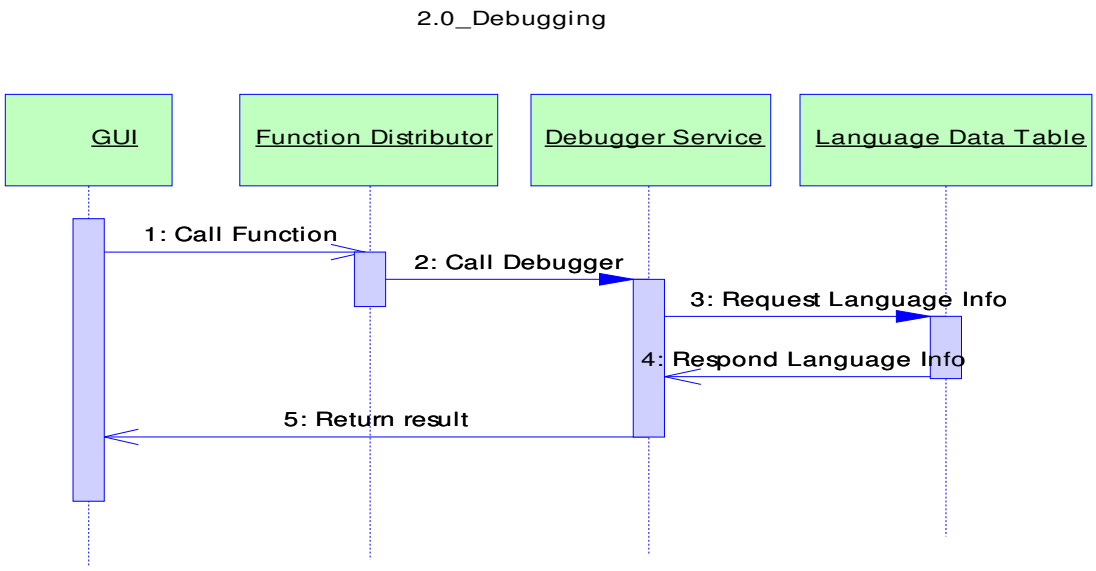
Step 1: AJAX developer opens the GUI and then action listener is activated. The user may write code or use some functions belong to GUI and second step is done relatively.

Step 2: If the user is writing code, it means the code was sent to editor by action listener. If the user uses keyboard in order to make operations or s/he uses button on GUI, this means action listener calls function handler.

Step 3: Operation result is sent to both display manager and editor.

Step 4: Highlight code and some notifications are sent to display manager. Then the result is displayed by display manager to the user.

3.2.2. Debugging



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.

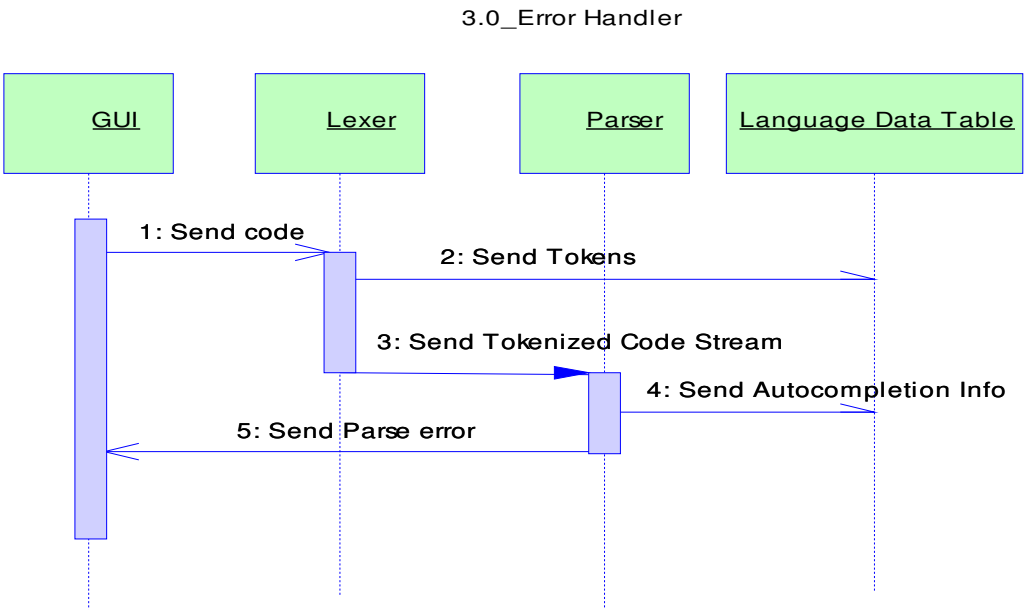
Step 2: After function distributor decides the function is debugger call, it calls debugger.

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

Step 4: Language data table returns relevant data information.

Step 5: The result is sent to GUI.

3.2.3. Error Handler



This diagram describes how to handle errors.

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

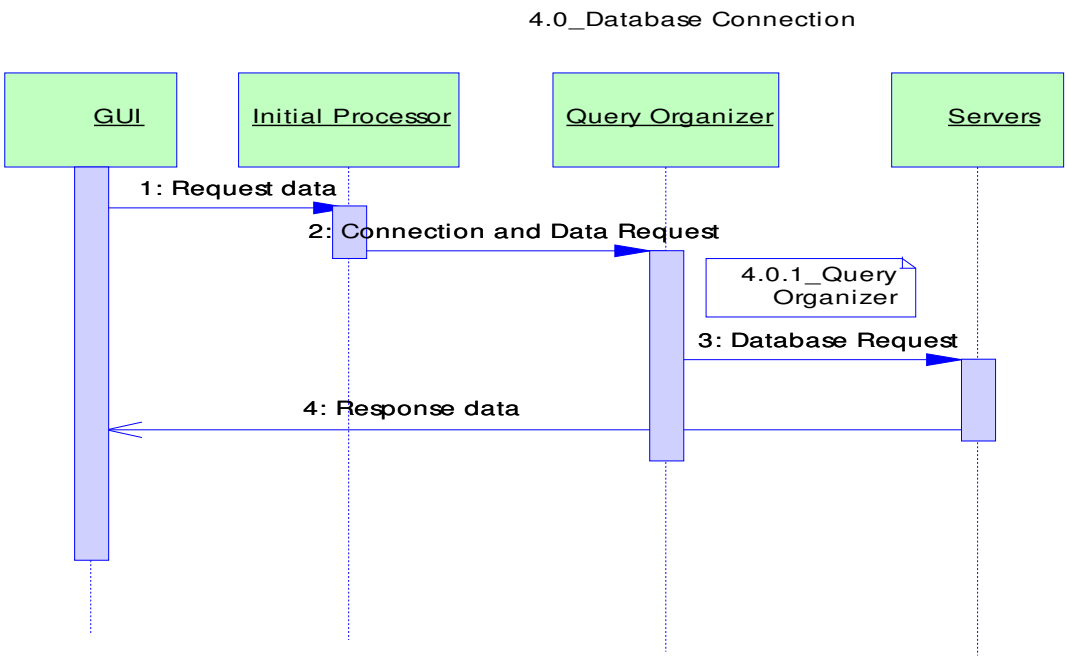
Step 2: The lexer builds up the tokens and send them to language data table in order to make them usable later to handle errors.

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

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

Step 5: The errors are sent to GUI.

3.2.4. Database Connection



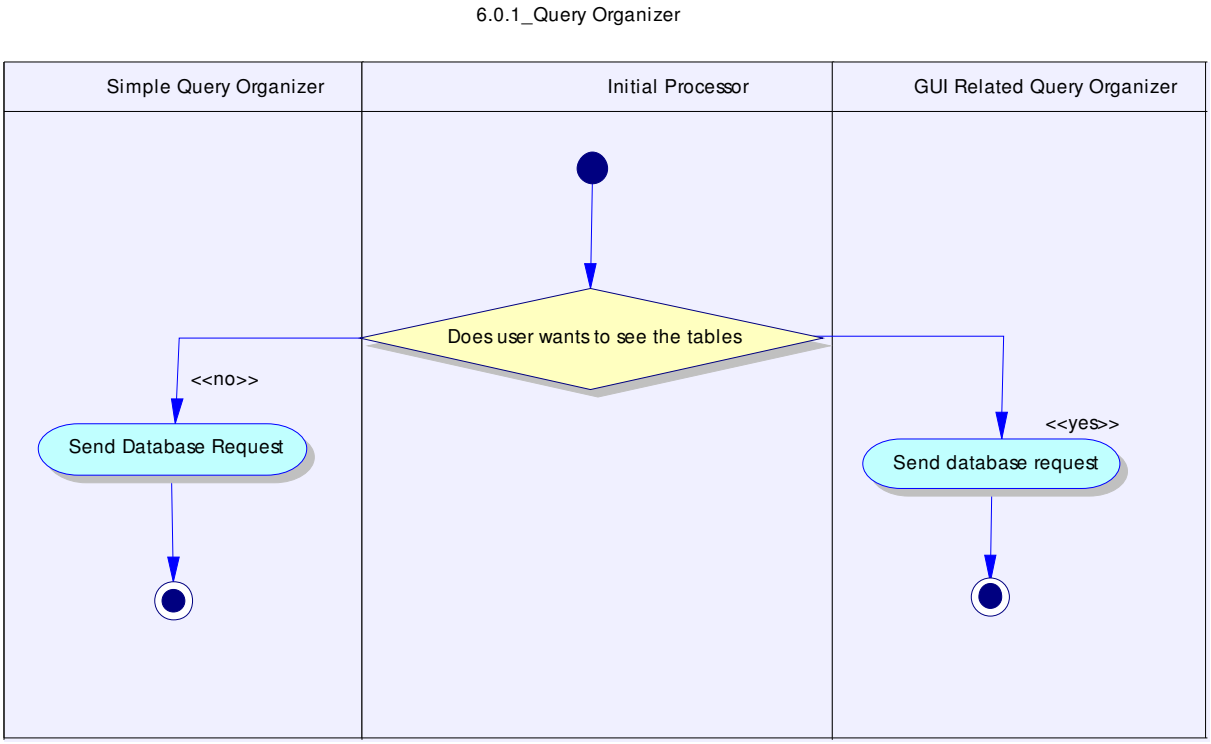
This diagram shows basic steps of database operations

Step 1: Data is requested through GUI.

Step 2 and 3: Initial processor makes the connection and request the data forum relevant server. If user wants to see the tables as a tree in GUI, the request goes to GUI related query organizer or if the user wants to make connection and write queries, the request goes to simple query organizer.

Step 4: Then the relevant result is got from two of them and sent to GUI.

3.2.4.1. Query Organizer (Activity Diagram)



3.3 Class Diagrams

Our design has mainly four modules. These are Application Service, Error Handler, GUI and Local Engine. These modules are figured as packages. And the related classes are in these modules. There are interfaces between these modules so that the data flow between these modules is well formed.

3.3.1 GUI

This package contains four classes. These are; Action Listener, Editor, Display Manager and Function Handler.

3.3.1.1 Action Listener:

This class is responsible from listening actions and directs them to relevant class. If the requested action is about an editor issue, action listener forwards it to the editor class with the relevant information. If the requested action is about functional operations, it forwards the request to the Function handler class.

- getInput(): Gets input from user and stores it in the input_stream property.
- sendCode(): According to the action type, sends the code information to the editor.
- sendOperation(): According to the action type, sends the operation information to the editor.
- parseInput(): Parses the input and according to the parse information stores the action type in the action type property.

3.3.1.2 Editor:

This class is responsible from code operations. It gets the code information from the Action Listener, and performs the necessary action.

- getCode(): Gets code input from the Action Listener, stores it in the code property.
- getNotification(): Gets notification about errors from Error Handler, stores it in the notification property.
- sendNotification(): Sends the notification stored in the notification property to the Display Manager.
- sendCodeStream(): Sends the code stream to Error Handler in order to get the errors.
- sendDisplayInfo(): Sends the information stored in the processedCode property and the information for display to the Display Manager
- getOperationResult(): Gets the operation result from the Function Handler, which can be an open file action.
- processCode(): Processes the returned code.

3.3.1.3 Function Handler:

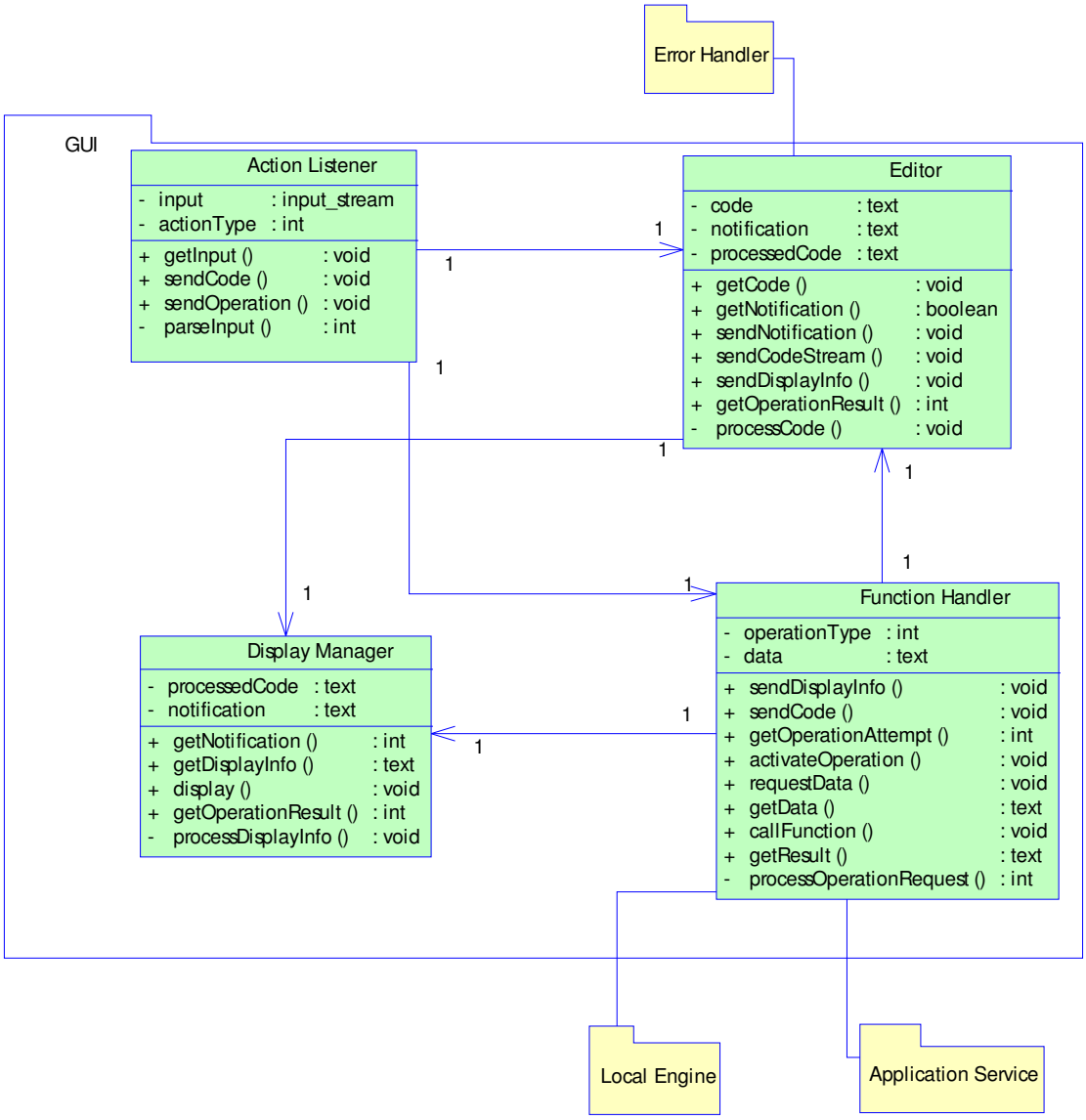
This class is responsible from handling the requested functions.

- sendDisplayInfo(): Sends the display info to the Display Manager.
- sendCode(): Sends the requested code to the Editor.
- getOperationAttempt(): Gets the operation attempt from the Action Listener and stores the operation type in the operationType property.
- activateOperation(): Activates the relevant operation according to the operationType.
- requestData(): Requests data from the Application Service.
- getData(): Gets the resultant data from the Application Service and stores in the data property.
- callFunction(): Requests the relevant function call from Local Engine.
- getResult(): Gets the result of the called function.
- processOperationResult(): Processes the operation result and directs it to the relevant class.

3.3.1.4 Display Manager:

This class is responsible from getting the operation results and displaying them to the user.

- getNotification(): Gets notification about errors and stores in the notification property.
- getDisplayInfo(): Gets display info from the Editor which is about highlighting etc. and stores in the processed code property.
- display(): Displays the information stored in the properties.
- getOperationResult(): Gets the operation result from the Function Handler.
- processDisplayInfo(): Processes the display info got from the Editor.



3.3.2 Application Service

This package contains three classes. These are; Initial Processor, Simple Query Organizer and GUI Related Query Organizer.

3.3.2.1 Initial Processor:

This class is responsible from processing the request and forwarding to the relevant class.

- getDataRequest(): Gets the data request from GUI stores it in the request property. Connects to the database and stores the connection in the connection property.
- processRequest(): Processes the request.
- directRequest(): Directs the request to the relevant class.
- forwardConnectionInfo(): Forwards the connection information to other classes so that no need for re-connection.

3.3.2.2 Simple Query Organizer:

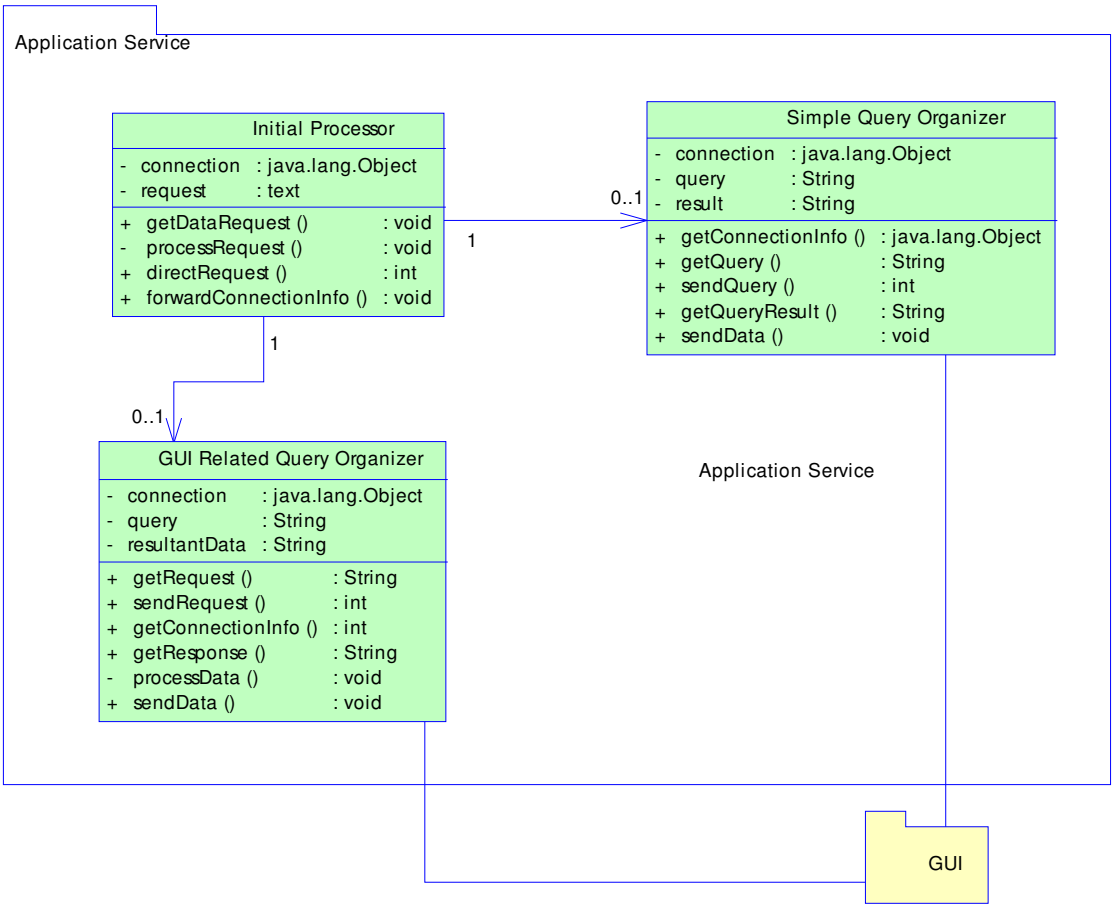
This class is responsible from getting the queries and sending them to database.

- getConnectionInfo(): Gets the connection info from Initial Processor and stores in the connection property.
- getQuery(): Gets the query and stores in the query property.
- sendQuery(): Sends the query stored in the query property to the database.
- getQueryResult(): Gets the result from database and stores in the result property.
- sendData(): Sends the resultant data to GUI.

3.3.2.3 GUI Related Query Organizer:

This class is responsible from displaying the database components.

- getRequest(): Gets the request from Initial Processor and stores it in the query property.
- sendRequest(): Sends the request to database.
- getConnectionInfo(): Gets the connection info from Initial Processor and stores in the connection property.
- getResponse(): Gets the response from the database and stores in the resultantData.
- processData(): Processes data got from database.
- sendData(): Sends the data to GUI.



3.3.3 Error Handler

This package contains two classes. These are; Lexer and Parser.

3.3.3.1 Lexer:

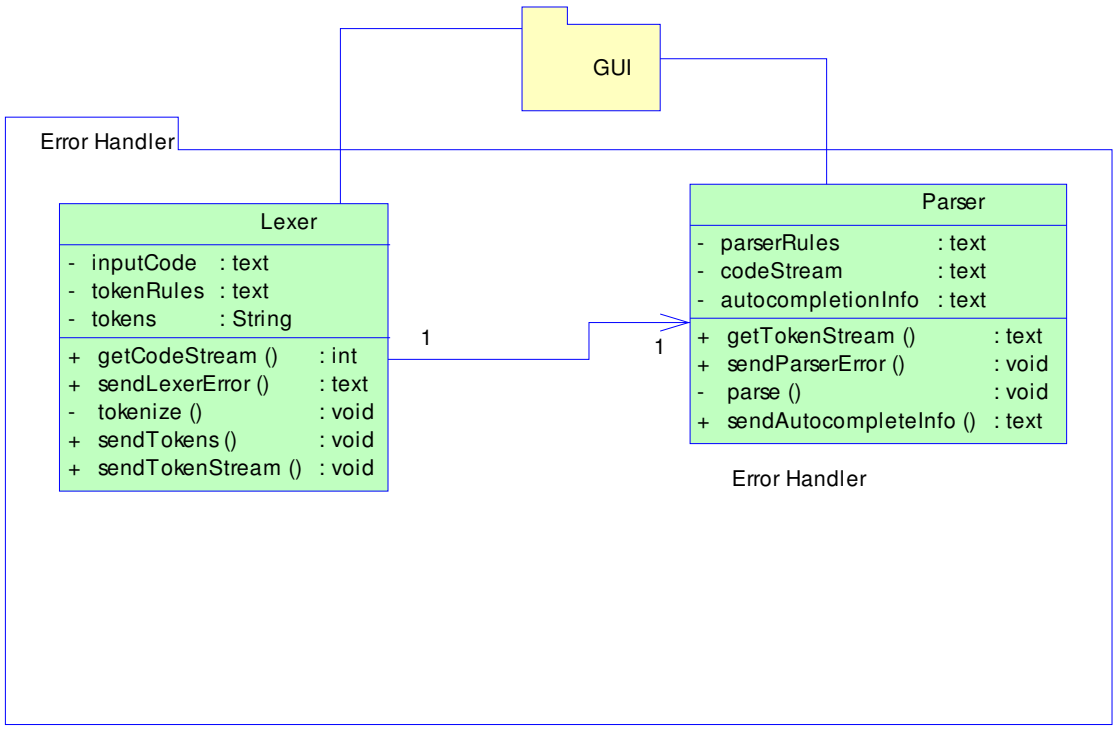
This class is responsible from tokenizing the input.

- getCodeStream(): Gets the code stream from GUI and stores it in the inputCode property.
- sendLexerError(): Send the errors found in Lexer to GUI.
- tokenize(): Tokenizes the input code and stores the tokens in tokens property.
- sendTokens(): Send tokens to Language Data Table and stores them to that table.
- sendTokenStream(): Sends the tokenized stream to parser.

3.3.3.2 Parser:

This class is responsible from parsing the input.

- getTokenStream(): Gets the token stream from Lexer and stores in codeStream property.
- sendParseError(): Sends parse error to GUI.
- parse(): Parses the input according to parserRules and constructs the autocompletion information.
- sendAutoCompletionInfo(): Send the auto completion information to Language Data Table.



3.3.4 Local Engine

This package contains four classes. These are; Function Distributor, Autocompletion Service, Debugger Service and File System Functions.

3.3.4.1 Function Distributor:

This class is responsible from distributing the function calls.

- getCall(): Gets the function call from GUI.
- processFunctionType(): Processes the call input and stores the function type in functionType property.
- activateFunction(): Activates the relevant function.

3.3.4.2 Autocompletion Service:

This class is responsible from autocompletion.

- getCall(): Gets the call from Function Distributor.
- sendRequest(): Sends the request to Language Data Table with the prevState property.
- getResult(): Gets the result from Language Data Table.
- processState(): Processes the previous state which is stored in the prevState property.
- sendResult(): Sends the result to GUI.

3.3.4.3 Debugger Service:

This class is responsible from debugging.

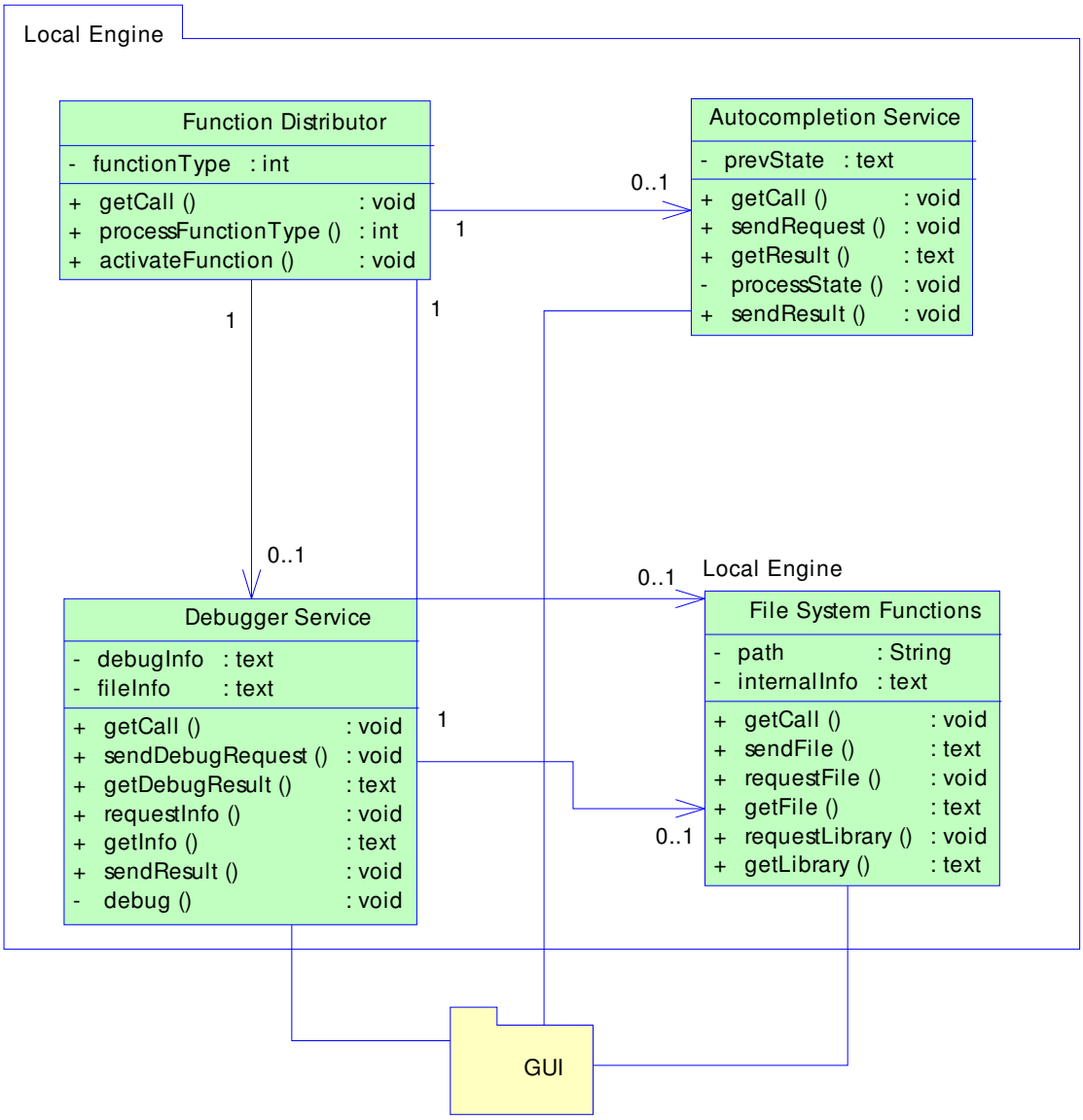
- getCall():Gets the call from Function Distributor.
- sendDebugRequest(): Sends the request for debug information.
- getDebugResult(): Gets the result .
- requestInfo(): Requests information from file system.
- getInfo():Gets information from file system and store in fileInfo.

- sendResult(): Sends the result to GUI.
- debug(): Used for debugging code and stores info in debugInfo.

3.3.4.4 File System Functions:

This class is responsible from file system operations.

- getCall():Gets the call from Function Distributor and store in path.
- sendFile(): Sends the file to Debugger Service.
- requestFile(): Request file from file system.
- getFile(): Get file from File system and stores in the internalInfo.
- requestLibrary():Request library from library.
- getLibrary():Get library from library.

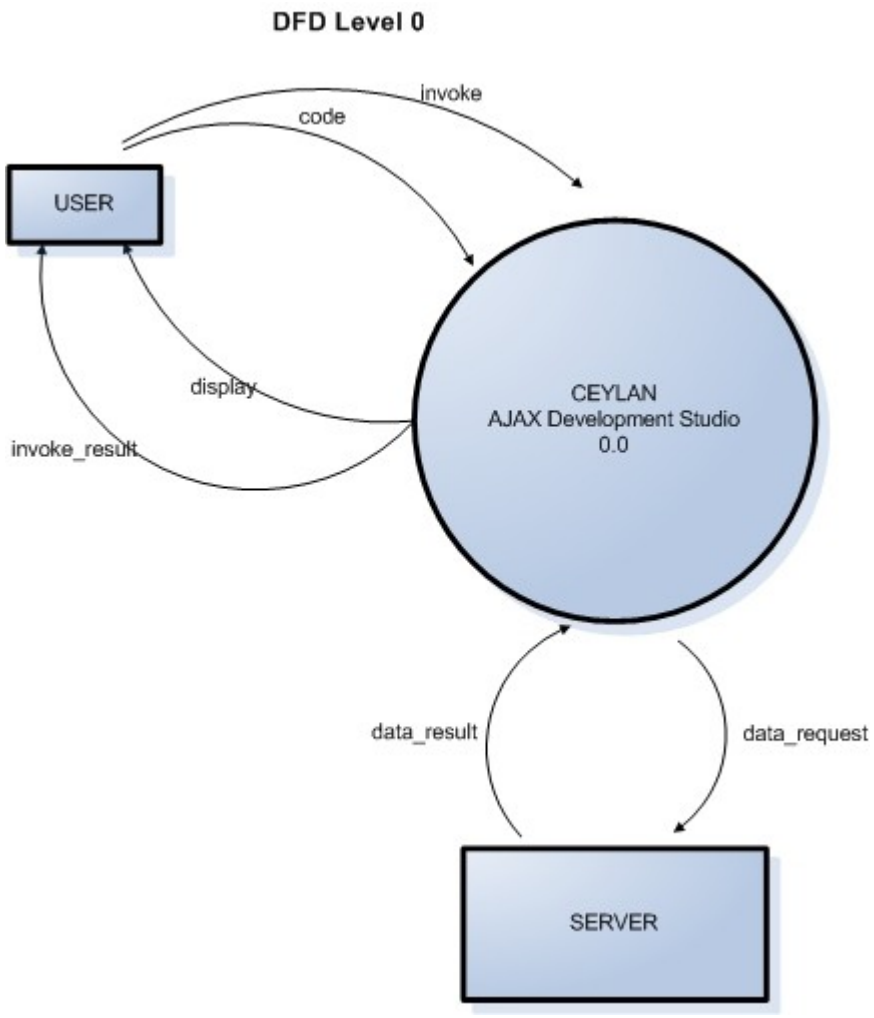

Local Engine
Function Distributor
- functionType : int
+ getCall () : void
+ processFunctionType () : int
+ activateFunction () : void
Autocompletion Service
- prevState : text
+ getCall () : void
+ sendRequest () : void
+ getResult () : text
- processState () : void
+ sendResult () : void
0..1
1
1
1
0..1
0..1
Local Engine
Debugger Service
- debugInfo : text
- fileInfo : text
+ getCall () : void
+ sendDebugRequest () : void
+ getDebugResult () : text
+ requestInfo () : void
+ getInfo () : text
+ sendResult () : void
- debug () : void
1
File System Functions
- path : String
- internalInfo : text
+ getCall () : void
+ sendFile () : text
+ requestFile () : void
+ getFile () : text
+ requestLibrary () : void
+ getLibrary () : text
0..1
GUI

4. MODELLING

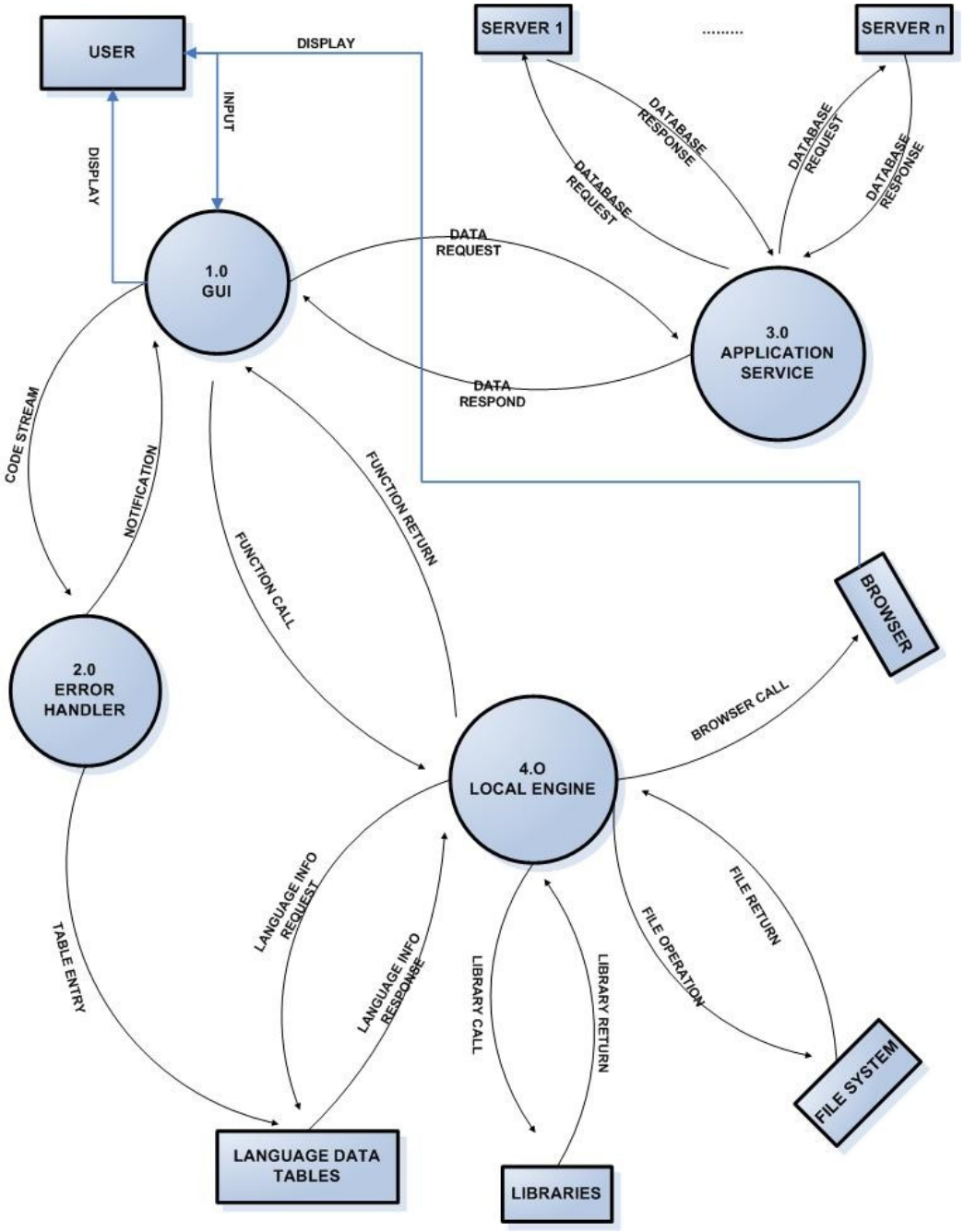
4.1 Functional Modeling

4.1.1 Data Flow Diagrams (DFD)

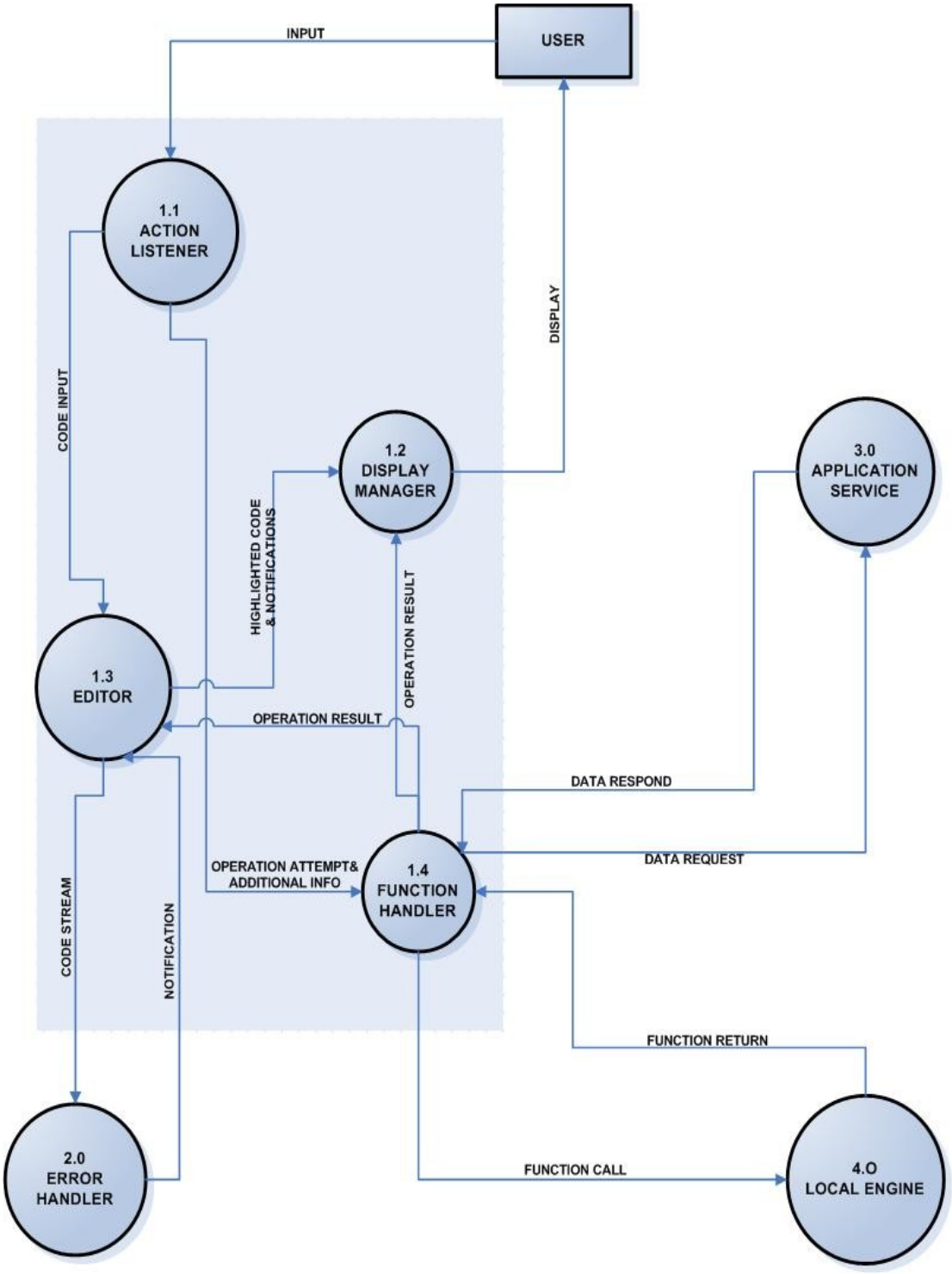
4.1.1.1 DFD Level 0



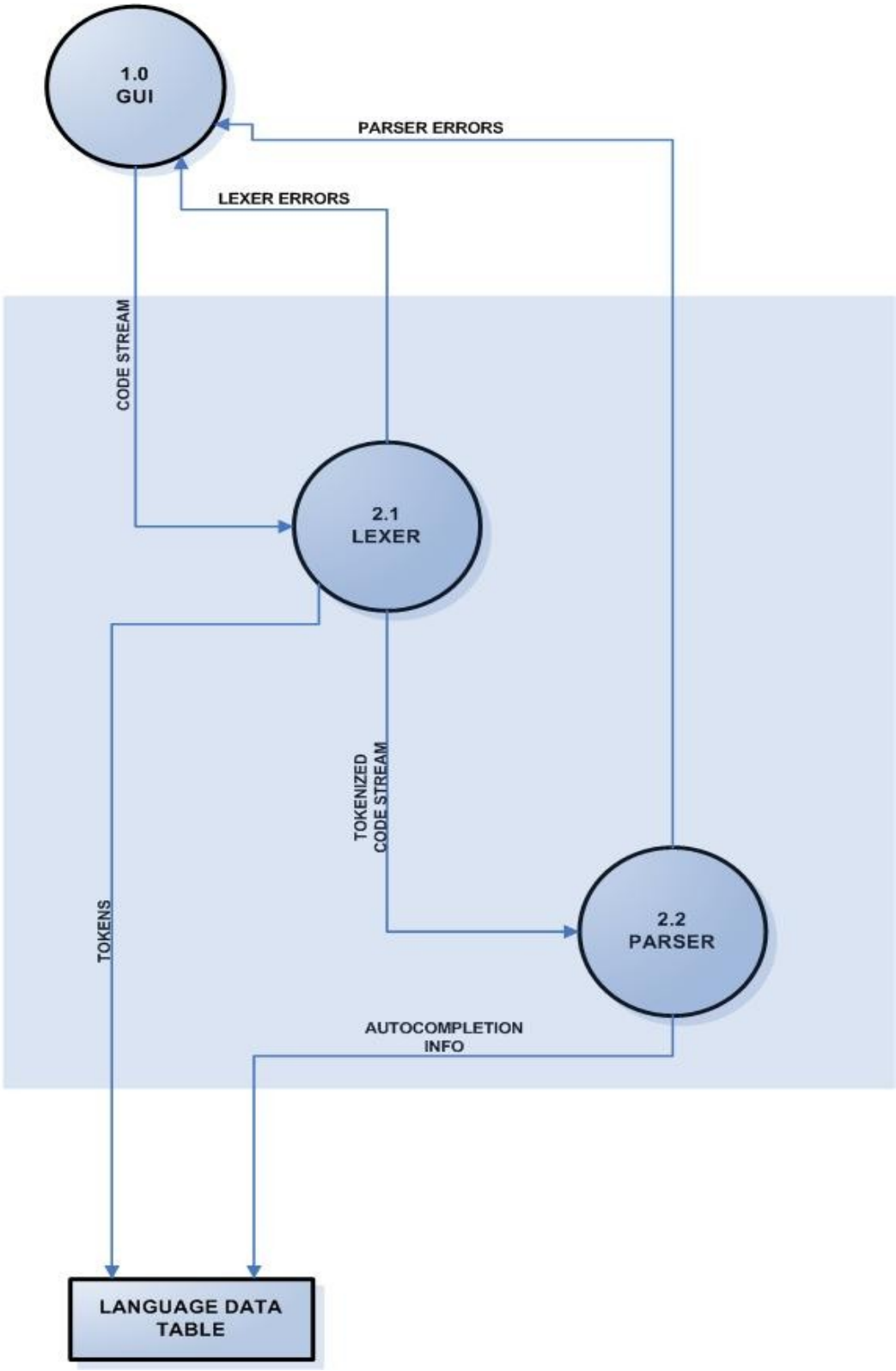
4.1.1.2 DFD Level 1



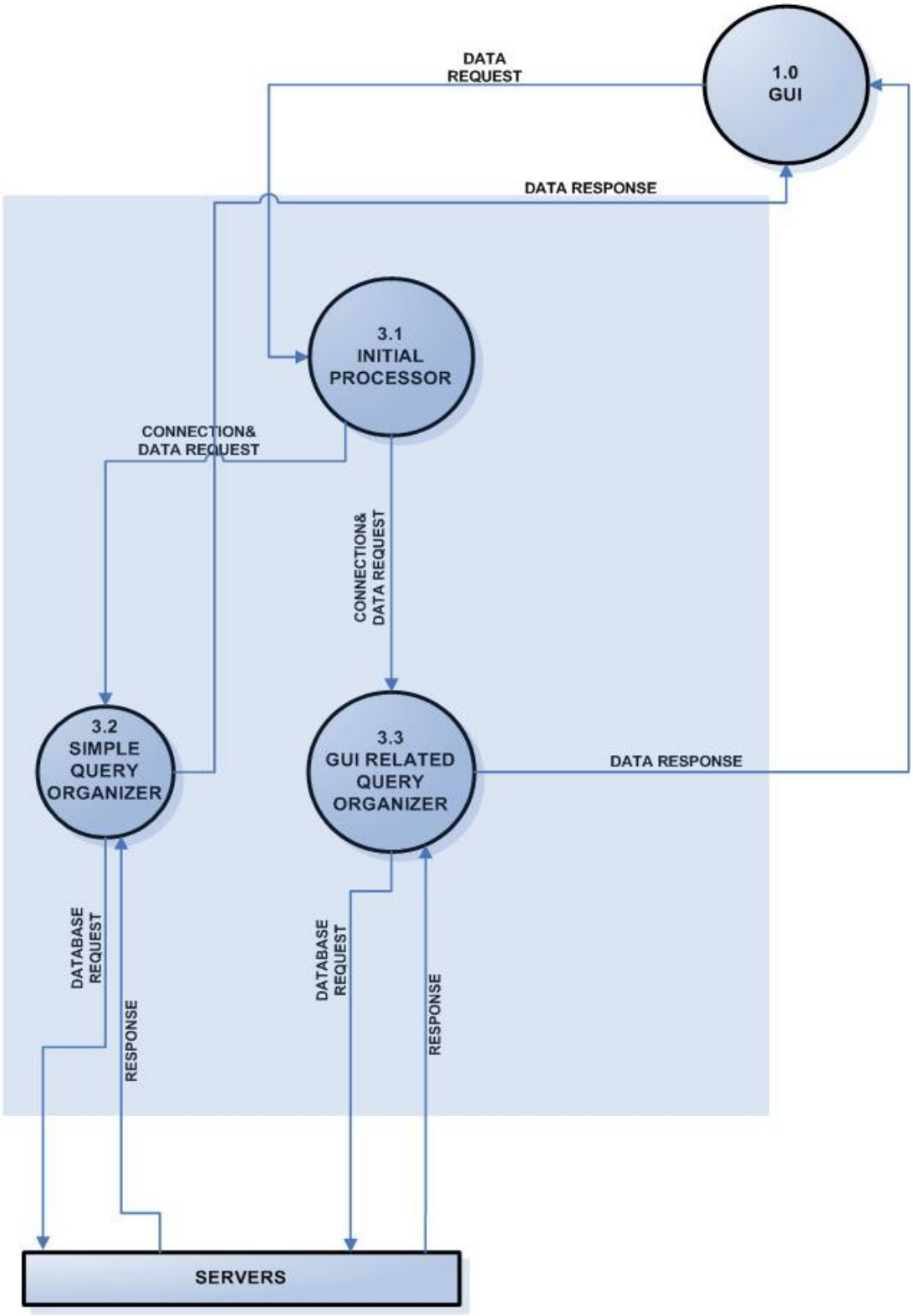
4.1.1.3 DFD GUI Level 2



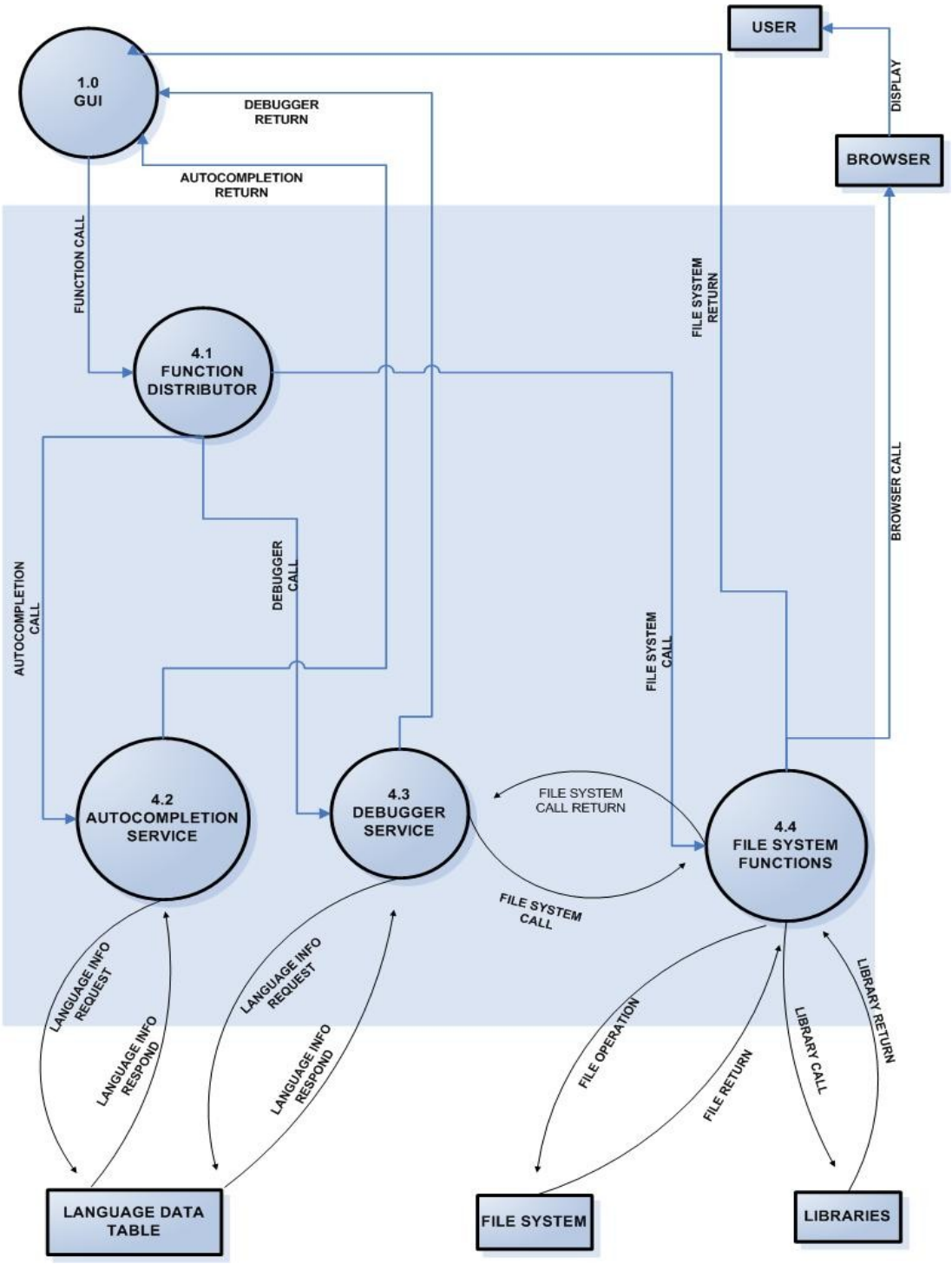
4.1.1.4 DFD ERROR HANDLER Level 2



4.1.1.5 DFD APPLICATION SERVICE Level 2



4.1.1.6 DFD LOCAL ENGINE Level 2



4.1.2 Process Specifications (PSPEC)

CEYLAN

This is our AJAX Development Studio consisting of components GUI, Editor, Error Handler, Application Server.

4.1.2.1 GUI

(**G**raphical **U**ser **I**nterface) A graphics-based user interface that incorporates movable windows, icons and a mouse. The ability to resize application windows and change style and size of fonts are the significant advantages of a GUI vs. a character-based interface. This process includes all the graphics view provided by our development studio. Menu, tree view, query execution and view, code-design-split view, etc. are all passed to user via GUI. GUI provides interaction between user and system. It takes user actions and displays response in appropriate manner.

4.1.2.1.1 ACTION LISTENER

Action listener listen user input. User input may be in two types: code input, operation invocation. User input is send to Editor (1.3). Operation invocations such as save, exit, connect to database ...etc is send to function handler with necessary information.

4.1.2.1.2 DISPLAY MANAGER

Display Manager is the main module which handles the main view. It takes highlighted code and operation results and puts them in appropriate place on the screen.

4.1.2.1.3 EDITOR

Editor takes code input and highlights it. It sends code to error handler and takes returned notification, highlighting is performed with this information.

4.1.2.1.4 FUNCTION HANDLER

Function handler takes operation attempt & additional info and divides operations into two: local operations and nonlocal operations as database connections query executions …etc.

4.1.2.2 ERROR HANDLER

Error handler takes the written code and makes necessary operation on it. If there exists some error it returns error in two ways: it may return an error message or it may show the error on screen via different highlight. While examining the code, error handler, writes some important information to LANGUAGE DATA TABLE. This information may be accessed by other operations.

4.1.2.2.1 LEXER

Lexer makes one pass and tokenizes the input. Returns errors and puts necessary information into the Language Data Table.

4.1.2.2.2 PARSER

Parser takes tokenized input and creates parse tree. Returns errors and puts necessary information into the Language Data Table.

4.1.2.3 APPLICATION SERVICE

CEYLAN allows users to make several database connections and monitor them. And in addition to monitoring CEYLAN provides Query Execution Tool, which gives users an interface for execute their queries on databases.

4.1.2.3.1 INITIAL PROCESSOR

Takes data request and decides whether this request is coming from user(query execution tool) or software (to create gui compenent, tree view). And

according to connection information, it creates a connection and pass them other components (3.2, 3.3).

4.1.2.3.2 SIMPLE QUERY ORGANIZER

Takes connection and data request, executes query on database and returns the query result.

4.1.2.3.3 GUI RELATED QUERY ORGANIZER

Takes connection and data request, executes query on database and returns information in expected structure.

4.1.2.4 LOCAL ENGINE

Local Engine controls operations on local machines 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.

4.1.2.4.1 FUNCTION DISTRUBITOR

Function distributor takes function call and examines the type of the call. It may be auto completion attempt, debugger call or file system call. Function call is reorganized and sends to appropriate component.

4.1.2.4.2 AUTOCOMPLETION SERVICE

When auto completion is called, this component takes last token from Language data Table and decides uncompleted parts, returns auto completion information, with the help of information from language data table. Returns to GUI.

4.1.2.4.3 DEBUGGER SERVICE

Debugger accesses file by file system calls and access language data table to get necessary information for debugging. Returns to GUI.

4.1.2.4.4 FILE SYSTEM FUNCTIONS

Operations like: save, load, open, create project, insert library function …etc is handled by this component. Returns to GUI.

4.1.3 Data Dictionary

Name	INPUT
Description	This user provided data establishes one part of the interaction.
Output from	User
Input to	GUI 1.0
Format	[keyboard action | mouse action]

Name	DISPLAY
Description	This is the graphical user interface provided to user by GUI and browser.
Output from	GUI 1.0 Browser
Input to	GUI 1.0
Format	[java swing objects]

Name	CODE STREAM
Description	User code written in text area.
Output from	GUI 1.0
Input to	ERROR HANDLER 2.0
Format	[HTML | JAVASCRİPT | CSS | PHP]

Name	NOTİFICATION
Description	[LEXER ERROR | PARSER ERROR]
Output from	ERROR HANDLER 2.0
Input to	GUI 1.0
Format	Formatted [String]

Name	TABLE ENTRY
Description	[TOKEN | AUTOCOMPLETION INFO]
Output from	ERROR HANDLER 2.0
Input to	LANGUAGE DATA TABLE
Format	Formatted [Token object | String]

Name	DATABASE REQUEST
Description	Queries that will executed on database
Output from	APPLICATION SERVICE 3.0
Input to	SERVER
Format	Formatted [SQL Query]

Name	DATABASE RESPONSE
Description	Query return from database.
Output from	SERVER
Input to	APPLICATION SERVICE 3.0
Format	Formatted [String]

Name	DATA REQUEST
Description	Database related operation invoke.
Output from	GUI 1.0
Input to	APPLICATION SERVICE 3.0
Format	Formatted [String]

Name	DATA RESPOND
Description	Database operation result
Output from	APPLICATION SERVICE 3.0
Input to	GUI 1.0
Format	Formatted [String]

Name	FUNCTION CALL
Description	Function call from user
Output from	GUI 1.0
Input to	LOCAL ENGINE 4.0
Format	Formatted [String]

Name	FUNCTION RETURN
Description	Function return from system
Output from	LOCAL ENGINE 4.0
Input to	GUI 1.0
Format	Formatted [String]

Name	FILE OPERATION
Description	File operation attempt on local machine
Output from	LOCAL ENGINE 4.0
Input to	FILE SYSTEM
Format	Formatted [String]

Name	FILE RETURN
Description	File operation return
Output from	FILE SYSTEM
Input to	LOCAL ENGINE 4.0
Format	Formatted [String]

Name	LIBRARY CALL
Description	Library function call
Output from	LOCAL ENGINE 4.0
Input to	LIBRARIES
Format	Formatted [String]

Name	LIBRARY RETURN
Description	Library function call return
Output from	LIBRARIES
Input to	LOCAL ENGINE 4.0
Format	Formatted [String]

Name	LANGUAGE INFO REQUEST
Description	Language related data request
Output from	LOCAL ENGINE 4.0
Input to	LANGUAGE DATA TABLES
Format	Formatted [String]

Name	LANGUAGE INFO RESPONSE
Description	Language related data response
Output from	LANGUAGE DATA TABLES
Input to	LOCAL ENGINE 4.0
Format	Formatted [String]

Name	TOKEN
Description	Language tokens, variables, keywords …etc
Output from	LEXER 2.1
Input to	LANGUAGE DATA TABLE
Format	Formatted [TOKEN OBJECT]

Name	AUTOCOMPLETION INFO
Description	Information will used in autocompletion: rules, last word …etc
Output from	PARSER 2.2
Input to	LANGUAGE DATA TABLE
Format	Formatted [String]

Name	TOKENIZED CODE STREAM
Description	Tokenized user code.
Output from	LEXER 2.1
Input to	PARSER 2.2
Format	Formatted [String]

Name	PARSER ERRORS
Description	Rule related syntax errors
Output from	PARSER 2.2
Input to	GUI 1.0
Format	Formatted [String]

Name	LEXER ERRORS
Description	Sytax errors based on wrong typing, lexical errors
Output from	LEXER 2.1
Input to	GUI 1.0
Format	Formatted [String]

Name	LEXER ERRORS
Description	Sytax errors based on wrong typing, lexical errors
Output from	LEXER 2.1
Input to	GUI 1.0
Format	Formatted [String]

Name	OPERATION ATTEMPT & ADDITIONAL INFO
Description	User operations like save, load …etc
Output from	ACTION LISTENER 1.1
Input to	FUNCTION HANDLER 1.4
Format	Formatted [String]

Name	OPERATION RESULT
Description	User operation return
Output from	FUNCTION HANDLER 1.4
Input to	DISPLAY MANAGER 1.2
Format	Formatted [String]

Name	HIGHLIGHTED CODE & NOTIFICATIONS
Description	Highlighted code that includes errors in different color.
Output from	EDITOR 1.3
Input to	DISPLAY MANAGER 1.2
Format	Formatted [String]

Name	CONNECTION& DATA REQUEST
Description	Database connection object and data request.
Output from	INITIAL PROCESSOR 3.1
Input to	SIMPLE QUERY ORGANIZER 3.2
Format	Formatted [CONNECTION OBJECT | String]

5. USER INTERFACE DESIGN

5.1 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

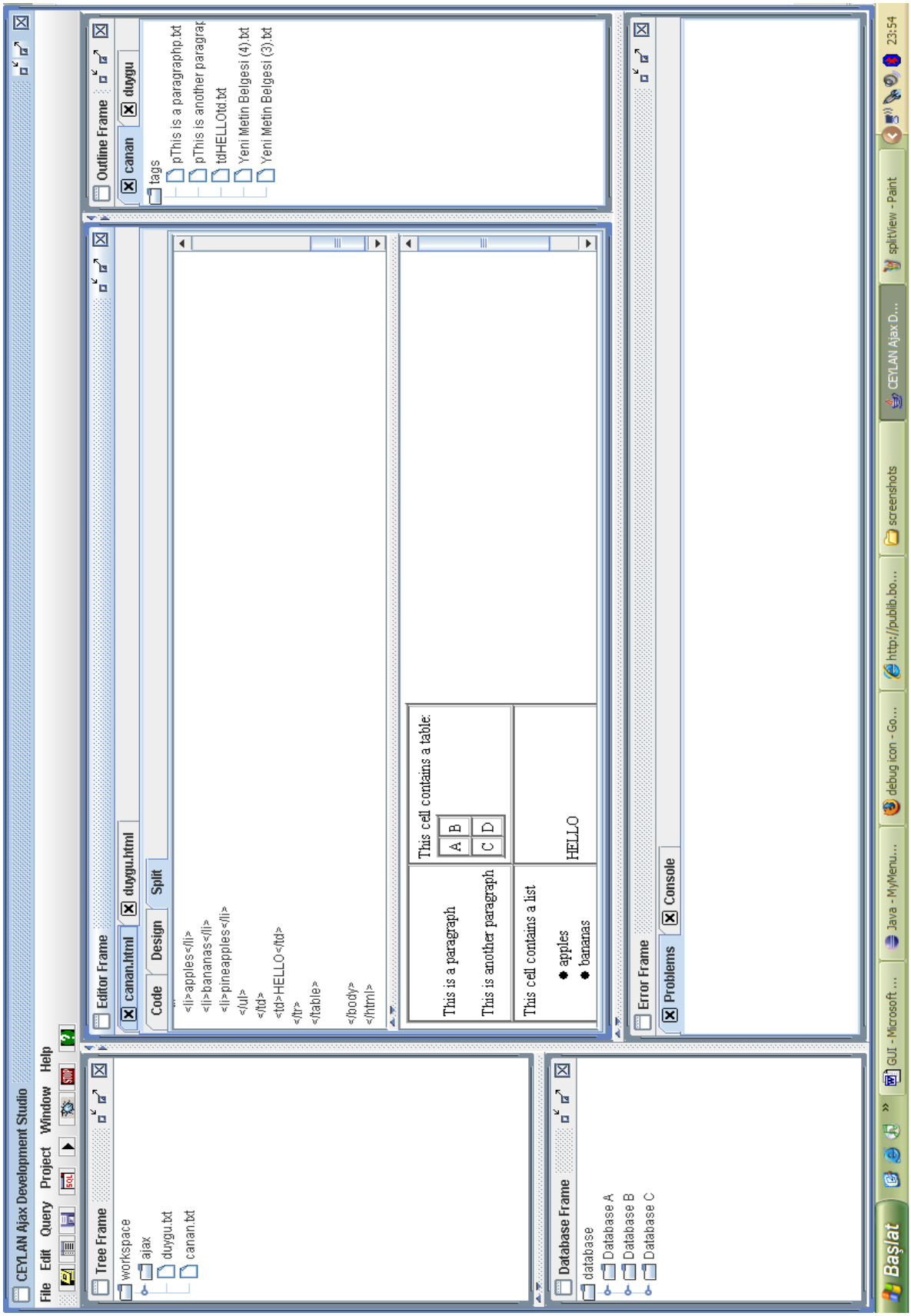
There three ways to handle user errors:

- 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.

5.2 Screenshots and Actions

5.2.1 General Editor View



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.

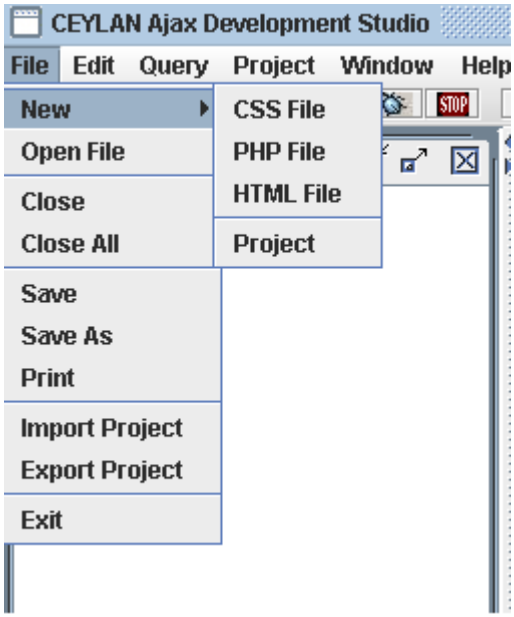
In editor view, user sees the files that s/he had opened. Each file comes with its own tab including its three different view options:

Code View: In code view, user can see his code tabbed and highlighted according to the type of his/her file. Our editor will provide the user with highlighted code(not applicable now since it is strongly connected to interpreted which will be implemented later).

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

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

5.2.2 File Menu View



New: By clicking on the new button user creates 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 view.

Open File: By selecting open menu, there comes a file chooser pop-up menu. S/he chooses one of his/her existing files. The file is opened in a new tab with its code, design and split view.

Close: When user chooses close menu, the file s/he is working on is closed.

Close All: When user chooses close menu, all the files that are open are closed.

Save: When user chooses save menu, the file that the user had created or changed is saved. The file is saved under workspace directory.

Save As: When user chooses save as menu, a copy of the file that the user had created or changed is saved with a different name. The file is saved under workspace directory.

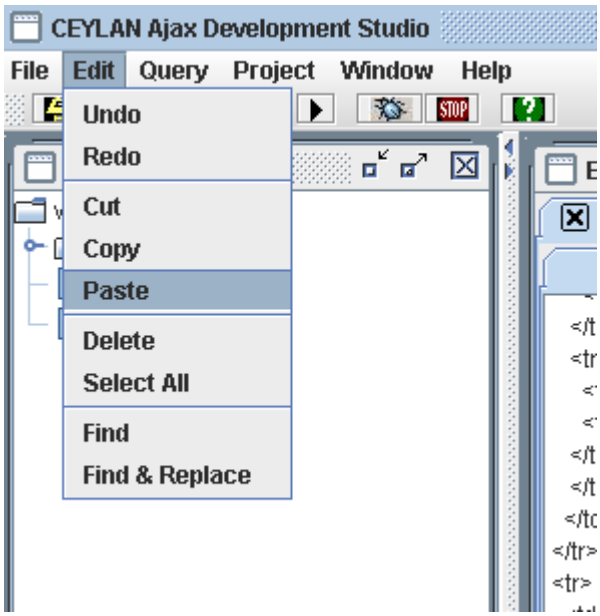
Print: When user chooses print menu, the file he is working on is printed via printer currently attached to the computer.

Import Project: By selecting import project menu, there comes a project chooser pop-up menu. S/he chooses one of his/her existing projects or other external projects and imports them to his/her workspace.

Export Project: By selecting export project menu, there comes a project chooser pop-up menu. S/he chooses and exports one of his/her existing projects to another workspace on computer.

Exit: When user chooses exit menu, user exits from Ceylan Ajax Development Studio.

5.2.3 Edit Menu View



Undo: By choosing undo option, user takes reverse action on what s/he had done.

Redo: By choosing redo option, user takes forward action on what s/he had done.

Cut: When user chooses cut menu, the item that is selected is cut.

Copy: When user chooses copy menu, the item that is selected is copy.

Paste: When user chooses paste menu, the item that is in the buffer is pasted on where the cursor is.

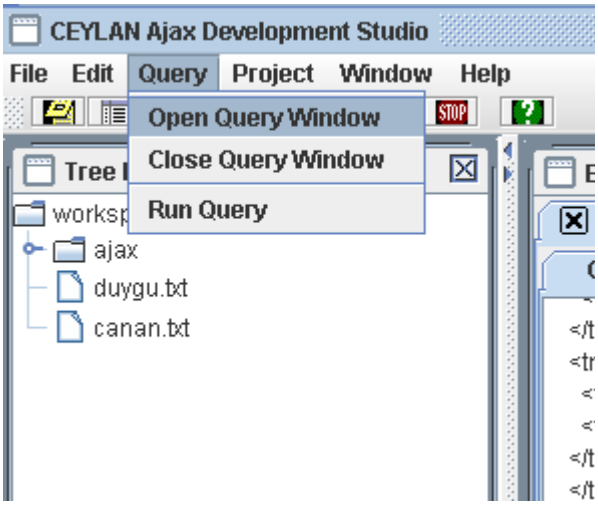
Delete: When user chooses delete menu, the item that is selected is delete.

Select All: When user chooses select all menu, the items on the current container are all selected.

Find: When user selects find menu, a pop-up window is opened and user enters the keyword s/he is searching for in the current document. The current document is parsed and the keyword is shown to user highlighted.

Find & Replace: When user selects find & replace menu, a pop-up window is opened and user enters 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 keyword wanted to be replaced.

5.2.4 Query Menu View

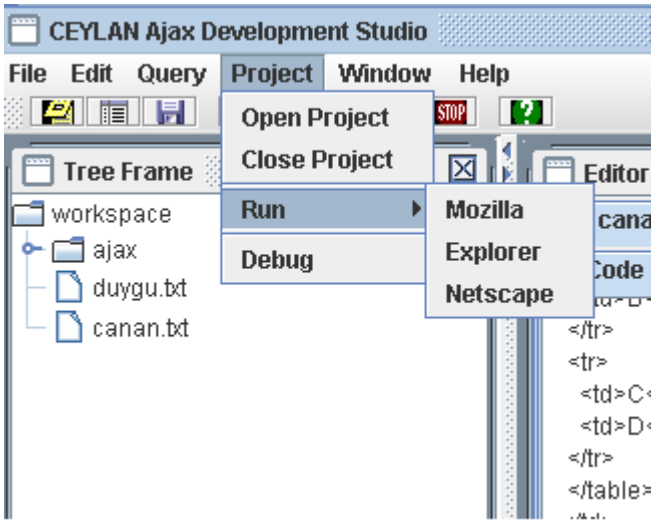


Open Query Window: When user selects open query window menu, query window is opened. We haven't decided on it yet whether it will be pop-up window or an inner separate window. For now we are implementing pop-up window.

Close Query Window: When user selects open query window menu, query window is closed.

Run Query: When user selects run query menu, the query s/he typed on query window will be executed and the result will be shown again on query window.

5.2.5 Project Menu View



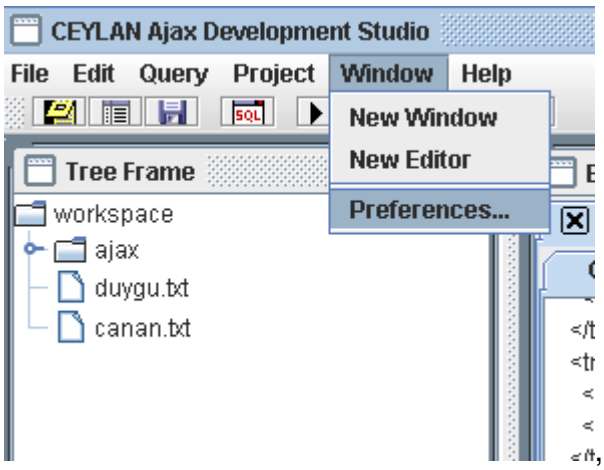
Open Project: When user selects open project menu, a pop-up window showing users projects will be opened and user will select one of his saved projects.

Close Project: When user selects close project menu, the project s/he is working on currently will be closed.

Run: When user selects run menu, the code will be evaluated(parsed, interpreted),and will be shown on browser. User can invoke run with different browsers which are Mozilla, Explorer and Netscape for now.

Debug: When user selects debug menu, until we implement the debugger, Mozilla add-on debugger Firebug will be opened.

5.2.6 Window Menu View

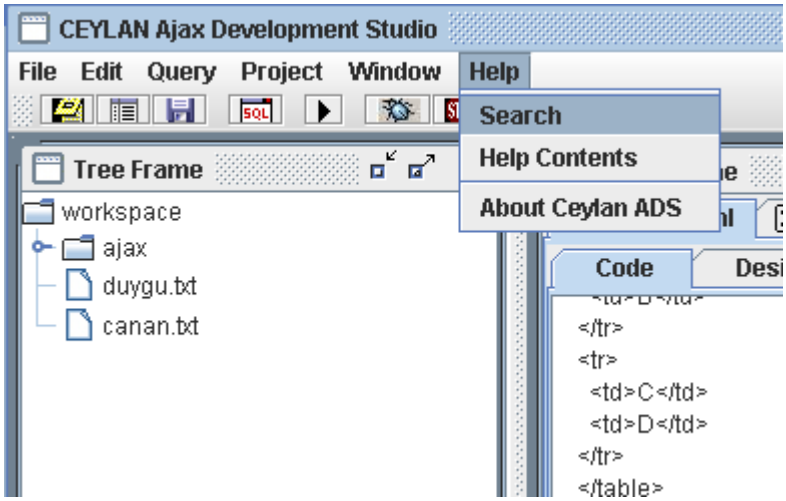


New Window: When user selects new window menu, a new window just like the whole frame will be opened.

New Editor: When user selects new editor menu, a new window just like editor frame will be opened in the same split area.

Preferences: For now, it is a pop-up window via which we setup the window properties.

5.2.7 Help Menu View

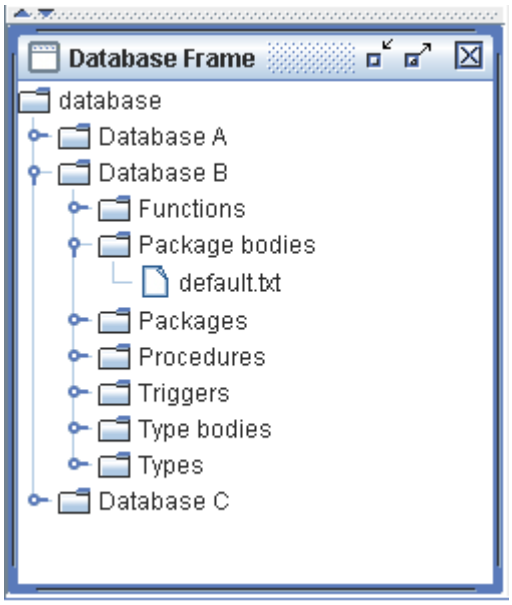


Search: When user chooses search menu, a pop-up window containing help topics will be opened.

Help Contents: When user chooses help contents menu, a pop-up window(the same one stated above) including search indexes near help topics will be opened.

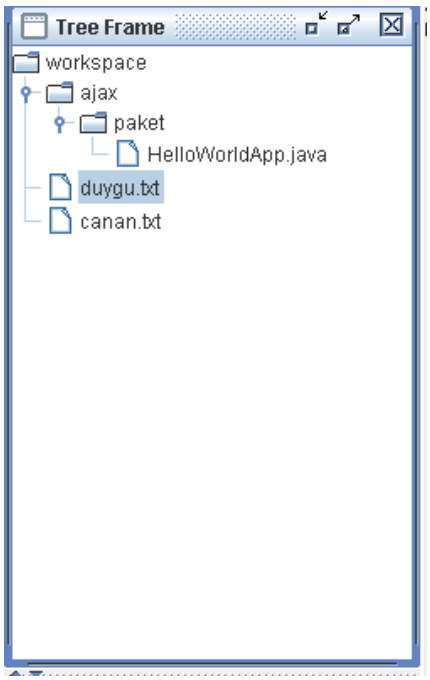
About Ceylan ADS: General information about Ceylan Ajax Development Studio will be displayed on a small display window.

5.2.8 Database View



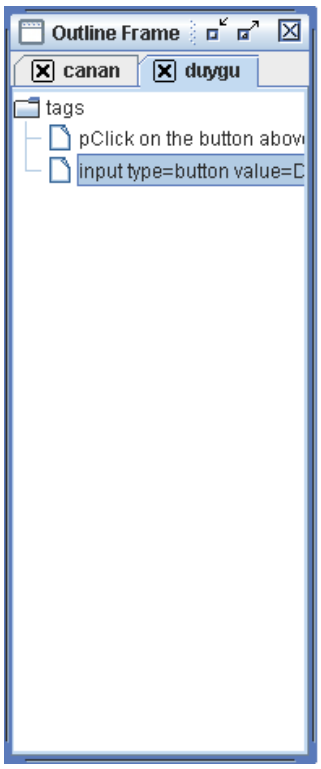
Since our Ceylan Ajax Development Studio will provide users with the opportunity to connect to different databases at the same time, we had the need to show all the databases s/he is connected showing all its functions, package bodies, packages, procedures, triggers, type bodies and types.

5.2.9 Project Workspace View



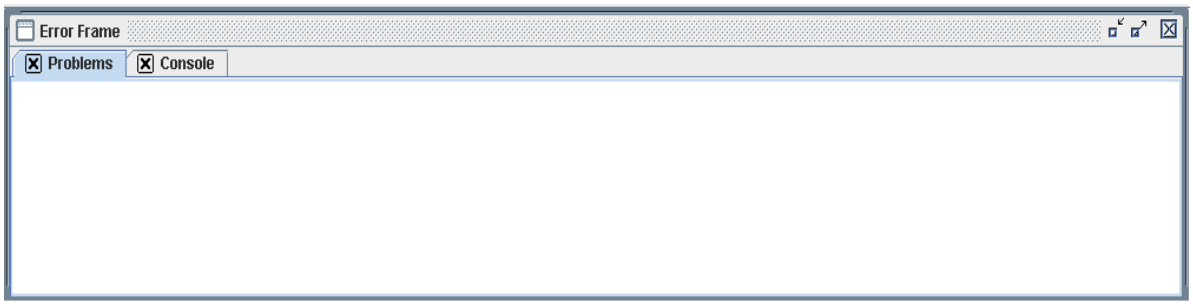
In project workspace view, user views his/her all files in his/her working space as tree view. When s/he clicks on one of his/her files on the tree, it is opened in editor frame with its own tab including code, design and split views.

5.2.10 Outline View



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. Since we need to parse the file to detect the tags in files, we put default tags for the time being.

5.2.11 ErrorView



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

6. TESTING

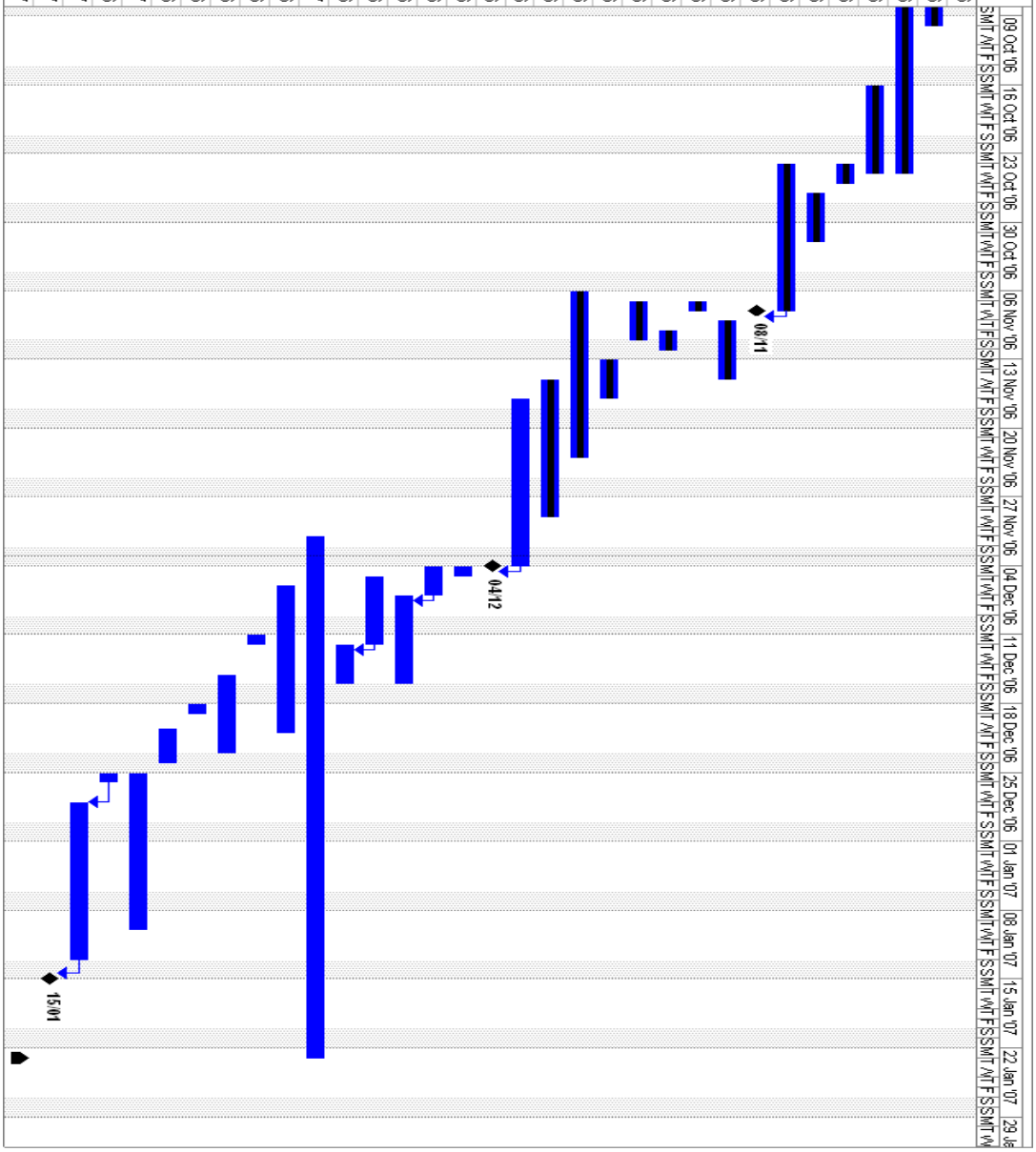
6.1 Unit Testing

- **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 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.

7 GANTT CHART

ID	Task Name	Duration	Start	Finish
1	Grouping and Selecting Topic	6 days	Tue 26/09/06	Tue 03/10/06
2	Proposal Report	5 days	Tue 03/10/06	Mon 09/10/06
3	Internet Research	14 days	Thu 05/10/06	Tue 24/10/06
4	Market Research	7 days	Mon 16/10/06	Tue 24/10/06
5	Template for Analysis Report	2 days	Tue 24/10/06	Wed 25/10/06
6	Doing Research	3 days	Fri 27/10/06	Tue 31/10/06
7	Writing Analysis Report	11 days	Tue 24/10/06	Tue 07/11/06
8	Milestone1	1 day	Wed 08/11/06	Wed 08/11/06
9	Building Web Site	4 days	Thu 09/11/06	Tue 14/11/06
10	Meeting	1 day	Tue 07/11/06	Tue 07/11/06
11	Preparing A Prototype of GUI	1 day	Fri 10/11/06	Sat 11/11/06
12	Analysis of Similar Products	4 days	Tue 07/11/06	Fri 10/11/06
13	Template for Initial Design Report	4 days	Mon 13/11/06	Thu 16/11/06
14	Interviews With Customer	13 days	Mon 06/11/06	Wed 22/11/06
15	Initial Design Report Research	10 days	Wed 15/11/06	Tue 28/11/06
16	Writing Design Report	11 days?	Fri 17/11/06	Sun 03/12/06
17	Milestone2	1 day	Mon 04/12/06	Mon 04/12/06
18	Meeting	1 day	Mon 04/12/06	Mon 04/12/06
19	GUI Design & Implementation I	3 days	Mon 04/12/06	Wed 06/12/06
20	GUI Design & Implementation II	7 days	Thu 07/12/06	Fri 15/12/06
21	Prepare Presentation	5 days?	Tue 05/12/06	Mon 11/12/06
22	Team Presentation	4 days	Tue 12/12/06	Fri 15/12/06
23	Start to Build up Demo	37 days?	Fri 01/12/06	Mon 22/01/07
24	Database Connection	11 days?	Wed 06/12/06	Wed 20/12/06
25	Meeting	1 day	Mon 11/12/06	Mon 11/12/06
26	GUI Design & Implementation III	6 days?	Fri 15/12/06	Fri 22/12/06
27	Meeting	1 day	Mon 18/12/06	Mon 18/12/06
28	Template for Final Design Report	2.5 days?	Wed 20/12/06	Sat 23/12/06
29	Final Design Report Research	12 days?	Mon 25/12/06	Tue 09/01/07
30	Meeting	1 day?	Mon 25/12/06	Mon 25/12/06
31	Writing Final Design Report	12 days	Thu 28/12/06	Fri 12/01/07
32	Milestone3	1 day?	Mon 15/01/07	Mon 15/01/07
33	Prototype Demo	1 day?	Tue 23/01/07	Tue 23/01/07