Software Design Document

Network Programming Project

Student Number: 013338170
Student Name: Yi Ding

Table of contents

1. Introduction

This document provides a formal description of the software design for the network programming project. Its goal is to list and illustrate system design in detail. The document contains 7 sections including: introduction, task description, software architecture, operation logic & protocol specification, file description, signal handling, and user interface.

The essential task details are clarified and described in Task description. Software architecture section provides a system overview, as well as the descriptions for all system components, the major tasks of each component, basic design principles, and the main data structures in use. In the section of Operation logic & Protocol specification, communications between components are specified and described in terms of message format and protocol operations to reflect software operation logics. Content and structure of files in use are covered in File description. Signal handling section lists all signals and the corresponding actions. Syntax and semantics of interactive commands are described in User Interface section.

This software design description will accurately reflect the structure of the final software system.

If particular aspects of system requirements described in this document are deemed as infeasible during implementation phase, reworking of relevant parts of the document will be undertaken, so that the project can be completed.

2. Task description

2.1. Overview

The goal of this software project is to design an interactive system providing online distributed dictionary service. The system is implemented in C language. It is mainly composed of two parts: the client side and the server side. The client side communicates with server side via net connection and obtains desired services according to the commands issued by the end users.

In our design, multiple clients are allowed to use the services concurrently. There are 3 servers forming a distributed system to provide the dictionary services. The basic structure is as illustrated in Figure 1.

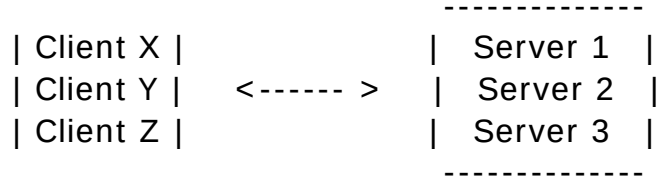

Figure 1: basic system structure - multiple clients interact with server farm.

2.2. Dictionary service

The distributed dictionary system provides end users with 4 operation choices: Explain - will get a term from the end user and reply with corresponding explanation; Remove - will delete an existing term provided by user from the database; Insert - will add a new term with its explanation into the database; Update - will allow the user to change the former term explanation to a new one. Necessary checks are performed by server side to ensure the validity of each operation such as not to insert the same term twice.

2.3. Server side

In our system, there are 3 servers forming a distributed server farm. Each server obtains a database which is a normal text file to store dictionary terms. The dictionary contents are not duplicated among all the servers. Every term is stored only on one of the servers. Servers can communicate with each other to check the database status. Servers provide concurrent access which allows multiple clients to connect them at the same time.

Transparency of the server system is maintained by making the servers connect with each other in a peer-to-peer way. The servers can exchange and pass commands to each other inside the server farm. Each server node acts both as “client” which issues request and also “server” which accepts requests and do the computation tasks plus reply to the “client”. The server farm provides a uniform interface to outside so that the client is not aware of actual operations done inside the server farm.

Necessary locking mechanisms are adopted in order to keep the database in coherent state and to achieve serialization.

2.4. Client side

The client side software provides a user interface and is responsible for connecting users to the server side. The client program will pass input from users to the server side and obtain replies. The final results will be printed on the screen to enable interactions between the user and the system.

Client can connect to any one of the servers in the system. The connection details are specified in a configuration file on the client machine.

Client program adopts necessary input checking mechanism to ensure the validity of all the user inputs.

3. Software architecture

3.1. System component design

There three logic parts participating in a service session according to our system design: 1) end user, 2) client program, and 3) server program. The relationship among different parts is demonstrated in the Figure 2.

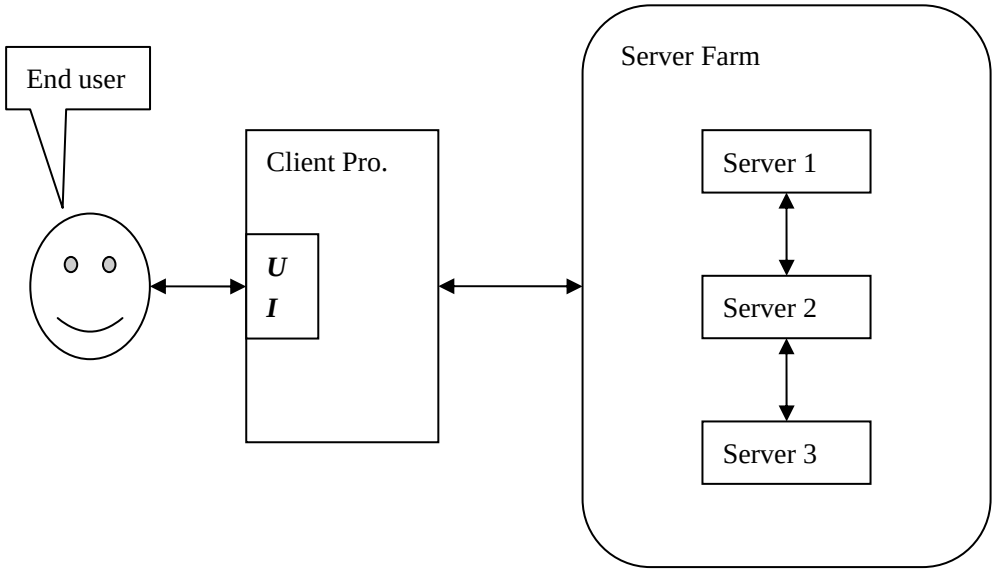

Figure 2: Components of dictionary system and their inter-relationship.

For every service session, end user will use the client program to obtain desired services from the server program following the working flow arrows in Figure 2. Four types of services are offered by our system: 1) Explain - will get a term from the end user and reply with corresponding explanation; 2) Remove - will delete an existing term provided by user from the database; 3) Insert - will add a new term with its explanation into the database; 4) Update - will allow the user to change the former term explanation to a new one.

The client program provides an interactive user interface to the end user in order to facilitate intercommunications. At the right beginning of every session, the client program will read its own ‘config.txt’ file to obtain necessary information of the system. Then it will connect to one of the servers in the server farm through TCP based connection. User requests will be analyzed and forwarded to the server side. The necessary information is exchanged between client program and server program in such a way that at any given time, client program will only connect to a single server program running in the server farm. All the database operations are fulfilled by cooperative operations of all three server programs. The client program will accept the reply from the server program and after processing, report result to the end user via printing the information on the screen. The log file is used by the client program

The server farm in our system is made up of three server programs running at the same time on different machines. Within the range of server farm, each server

program can connect to other servers through TCP socket. All three server programs will exchange necessary information through the channels and thus providing the distributed dictionary service to end user. The dictionary service provided by our system is regarded as distributed in that all the interactions among servers are transparent to the end user. The TCP connection between one server and the client program will maintain during the whole operation session until the user chooses to quit the service. The server program will check, exchange, and collect information within the server farm. After necessary processing, results will be sent back to client program via the established connection. From the user point of view, every user can connect to the “dictionary database”, issue commands, and obtain desired services.

In the server system, each of the server programs contains a database to store terms and related explanations. Our design requires that every term and its explanation will only be stored on one of the server programs only. There is no duplicated copy on other machines. To enhance the performance, distributed hash table algorithm (DHT) is adopted so that every term is hashed into a key and depending on the key map, system will know which server will hold the term and its explanation. Besides the DHT, when there is request arrived either directly from client program or indirectly from another server, this server program will first check its local hash table in which the status of the database is reflected. Based on the information from hash table, server will check its database and the final result will be sent back via a ‘Database Reply’ message to the client program or the server connecting to it.

In our design, the connections maintained by every server include the TCP connection to a fixed client program and another optional TCP connection to any of the server program within the server farm. The intercommunications between server and client, and as well as among different servers will be discussed in detail in Section 4 Operation logic & Protocol specification.

3.2. Data Structure

Three types of data structure are used in our system: buffer, DHT, and ordinary hash table. Both client program and server program will maintain a read/write buffer to store the information passing to or coming from TCP sockets. Client program has a buffer to store the input information from end user. The buffer in system is simply a continuous memory space in the form of array in OS.

DHT is used by server program to locate the term in the server farm. The idea of DHT is to distribute the hash table to different locations. And in our dictionary system, the key will reflect the location of every term that which server in the server farm should hold this term and its explanation. The DHT hashing key in our system is by adding up the ASCII code number of all the characters in a term string and mod the sum value to 3. If for instance, a term is “table”, the sum of each of the characters of the string is 116+97+98+108+101= 520. And the sum 520 mod 3 = 1. Therefore, the term

“table” and its explanation will be stored in server 2. (1+1 by considering the notation usage) Every term will have a unique DHT key, and this key value will determine its location in the server farm. By using the DHT, it saves the time that may be waste on randomly picking up a server and carry on searching operation. The uniqueness of DHT key thus promoting the operational performance. For

Besides the DHT, every server also has a hash table of its own. The usage of hash table on a local machine is to fast detect the whether a term does exist on the server or not. In our design, the hash table size is 10, with index from 0 – 9. The key value is the sum of ASCII code of each character of a string, and mod the final sum to 10. Hence, every term will have a location in the hash table according to its key. The value stored in each index position is the number of terms with the same hash key that exists on this machine. After obtaining the term from either the client or from another server, the term will be calculate with local hashing method. The hash table information will be saved in a local file “hash.txt”. Every time before conducting a search operation, the client program will check the hash table information at first to figure out whether it is necessary to further search the ‘data.txt’ file. If the value stored in the hash table is 0 for that key, meaning there is no such term on this machine, the server program will not process any further search operation. As DHT, local hash table is adopted for performance consideration. The hashing method is flexible and can be changed to maximum the efficiency. Detailed description of hash table file is covered in Section 5 File description.

4. Operation logic & Protocol specification

Communications between different components in our system are based on TCP. There are two levels of communication: 1) first level communication between client program and server program; 2) second level communication between two server programs in the server farm. The messages passed between different components are normal C character string starting with a flag character at the beginning of each string. Every message string is ended with ‘\n’. Both the client and server program will parse the string according to the scenario and react differently following the protocol specifications.

4.1. Client – Server communication

The client – server communication is the first level communication in the system. The client at any time will connect to one and only one server based on the information from the configuration file on the local machine. The default server will maintain the TCP connection will the client program during the whole service session. The messages are exchanged through TCP sockets on both client and server. According to the DHT information, if the operation can be fulfilled by the default server, system will not enter the second level communication. Client program can issue requests and receive replies from the default server. Based on the replies, client will translate the

answers to human readable format and present them to the end user.

4.1.1. Message format

The message used in the first level follow a general rule that every message string starts with the flag character indicating the service type or the exception type:
'E' = Explain;
'R' = Remove;
'I' = Insert;
'U' = Update;
'X' will indicate an error with extra payload to specify details.

Following the flag, the payload information will be organized according to the flag character and type of the message.

1) Request from the client program:
For 'E', the format of string is "E: keyword\n"
For 'R', the format of string is "R: keyword\n"
For 'I', the format of string is "I: keyword: explanation\n"
For 'U', the format of string is "U: keyword: explanation\n"
There is no message starting with 'X' from client program.

2) Reply from the default server as 'Database Reply':
For 'E', the format of string is "E: explanation\n"
For 'R', the format of string is "R\n"
For 'I', the format of string is "I: keyword\n"
For 'U', the format of string is "U: keyword\n"
The above four replies indicate successful operation.

For 'X', the string will contain extra information to convey error types.
"X:E:keyword\n" – meaning no such keyword found
"X:R:keyword\n" – meaning no such keyword, thus can't delete
"X:I:keyword\n" – meaning the keyword already exists in the database
"X:U:keyword\n" – meaning no such keyword, thus can't update

4.1.2. Protocol operation

The server program will listen to a predefined socket since it starts to run. After the client connecting to the server and establishing a TCP channel, the data transmission phase starts.

Client program will pack the end user request to form a valid request message according to our message format defined in 4.1.1. The message is sent via socket to the server. When receiving a request by read from the receiving buffer, the server program will parse the string and determines further actions:
1) If the flag of request message is 'E', server will first calculate the DHT key value

to check whether this term is possible to be handled locally. If yes, the server program will use local hash table to check whether the term is stored on this machine or not. The local hash table is stored on the local file ‘hash.txt’. If the value in the index is larger than 1, meaning there is a possible term on the machine with the same hash key. The server will read the ‘data.txt’ file to locate the term. If the index value is 0, an error message will be sent following the format specified in 4.1.1. If the term is found in the ‘data.txt’, the server will start to prepare a reply message. The explanation will be appended after the ‘E’ flag according to the format specified in 4.1.1. and server will send the reply to the client program.

2) If the flag is ‘R’, server will calculate the DHT key value to check whether this term is possible to be handled locally. If yes, the server program will use local hash table to check whether the term is stored on this machine or not. If in the hash table, the index value is 0, meaning no such term existing in the database, server will reply with an error message “X:R:keyword\n”. If index is larger than 0, server will search the database file to check the term. If the term is found, server will delete the term and its explanation and reply to the user with a successful operation message “R\n”. The corresponding index value in the hash table will be decreased by 1. If the term is not found in the database file, an error message will be sent.
3) If the flag is ‘I’, server will calculate the DHT key value to check whether this term is possible to be handled locally. If yes, the server program will use local hash table to check whether the term is stored on this machine or not. If in the hash table, the index value is 0, the server will append this new term and its explanation to the database file, and send a successful operation message “I: keyword\n” to the client program. At the same time the corresponding hash index value will be increased by 1. If the hash index value is greater than 0, server program will search the whole database file to find out whether there is already such a term existing in the database file. If no match, server will follow the normal procedure to append new term and reply with a message indicating the successful operation plus increasing the hash index value by 1. If there is a match, the server will send an error message “X:I:keyword\n” to the client program.
4) If the flag is ‘U’, server will calculate the DHT key value to check whether this term is possible to be handled locally. If yes, the server program will use local hash table to check whether the term is stored on this machine or not. If the index value is 0, server will send an error message “X:U:keyword” to client indicating that no such term existing in the database. If the index is greater than 0, server will search the database to find a match. If a match is found, server will update the explanation of the term and send back a successful operation message. If no match, an error message will be sent.

When the client program receives a reply message, it will first check the flag to determine which action should be followed:

1) If the flag is ‘E’, meaning the operation is fulfilled, the client program will parse

the explanation in the message and display to the end user.

2) If the flag is ‘R’, meaning the operation is fulfilled, client program will display a message to user that the command is fulfilled.
3) If the flag is ‘I’, meaning the operation is fulfilled, client program will display a message to user that the command is fulfilled.
4) If the flag is ‘U’, meaning the operation is fulfilled, client program will display a message to user that the command is fulfilled.
5) If the flag is ‘X’, client will further check the second parameter in the message.
   If ‘E’, client program will inform the user no such a term in database;
   If ‘R’, client program will inform the user no such a term in database;
   If ‘I’, client program will inform the user there is already such a term in database;
   If ‘U’, client program will inform the user this term is successfully updated in database;

4.2. Server – Server communication

The server – server communication happens only under the condition that the request from client can not be fulfilled directly by the default server. The cases include: 1) For ‘Explain’, the term is not stored on the default server and should be, if existing, on the other machine according to the DHT key value; 2) For ‘Remove’, the target term is not stored on the default server and should be, if existing, on the other machine according to the DHT key value; 3) For ‘Insert’, the DHT key indicates that the new term should be stored, if not existing, onto another server according to the DHT key; 4) For ‘Update’, the DHT key indicates that this term may be stored, if existing, on the other machine based on the DHT key.

From the DHT information, the default server will connect to a chosen server and send a request message. Based on the reply from the chosen server, the default server will forward the message further to the client program. From the connection point of view, the default server is acting as both a requester and a responder in the system. When communicate with another server, the default server is regarded as a requester, while when communicating with client, the default server is a responder. The chosen server will fork() a child server process to handle the service session after the accept() function call.

4.2.1. Message format

The messages used in the second level follow a general rule that every message string starts with the flag character indicating the service type or the exception type:
‘E’ = Explain;
‘R’ = Remove;
‘I’ = Insert;
‘U’ = Update;
‘X’ will indicate an error with extra payload to specify details.

Following the flag, the payload information will be organized according to the flag character and type of the message.

1) Request from the default server:
For ‘E’, the format of string is “E: keyword”
For ‘R’, the format of string is “R: keyword”
For ‘I’, the format of string is “I: keyword: explanation”
For ‘U’, the format of string is “U: keyword: explanation”
There is no message starting with ‘X’ as a request message.

2) Reply from the chosen server as ‘Database Reply’:
For ‘E’, the format of string is “E: explanation”
For ‘R’, the format of string is “R”
For ‘I’, the format of string is “I: keyword”
For ‘U’, the format of string is “U: keyword”
The above four replies indicate successful operation.

For ‘X’, the string will contain extra information to convey error types.
“X:E:keyword” – meaning no such keyword found
“X:R:keyword” – meaning no such keyword, thus can’t delete
“X:I:keyword” – meaning the keyword already exists in the database
“X:U:keyword” – meaning no such keyword, thus can’t update

4.2.2. Protocol operation

Every server in the server farm will always hold a predefined port for other servers to connect. The port number is different from the port used to setup connection with client program. A TCP socket is listening ever since the server program is run. When there is a need to connect to a chosen server, the default server will initiate connection at first by calling connect() function. After the TCP channel is established, the message will be exchanged between two servers.

The default server packs the request message which origins from the client program and sends to a chosen server. The format of the request follows the specification in 4.2.1. When receiving a request by read from the receiving buffer, the chosen server will parse the string and determines further actions:

1) If the flag is ‘E’, the chosen server will use local hash table to check whether the term is stored on this machine or not. If the value in the index is larger than 1, meaning there is a possible term on the machine with the same hash key. The server will read the ‘data.txt’ file to locate the term. If the index value is 0, an error message will be sent following the format specified in 4.1.1. If the term is found in the ‘data.txt’, the chosen server will start to prepare a reply message. The explanation will be appended after the ‘E’ flag according to the format specified in 4.1.1. This message will be sent to the default server.

2) If the flag is ‘R’, the chosen server will use local hash table to check whether the term is stored on this machine or not. If in the hash table the index value is 0, meaning no such term existing in the database, server will reply with an error message “X:R:keyword\n”. If index is larger than 0, server will search the database file to check the status. If the term is found, server will delete the term and its explanation and reply to the user with a successful operation message “R\n”. At the same time, the corresponding hash index value will be decreased by 1. If the term is not found in the database file, an error message will be sent.
3) If the flag is ‘I’, the chosen server will use local hash table to check whether the term is stored on this machine or not. If in the hash table the index value is 0, the server will append this new term and its explanation to the database file, and send a successful operation message “I: keyword\n” to the default server. At the same time the corresponding hash index value will be increased by 1. If the hash index value is greater than 0, server program will scan the whole database file to find out whether there is already such a term existing in the database file. If no match, server will follow the normal procedure to append new term and reply with a message indicating the successful operation plus increasing the hash index value by 1. If there is a match, the server will send an error message “X:I:keyword\n” to the default server.
4) If the flag is ‘U’, the chosen server will use local hash table to check whether the term is stored on this machine or not. If the index value is 0, server will send an error message “X:U:keyword” to default server indicating that no such term existing in the database. If the index is greater than 0, server will search the database to find a match. If a match is found, server will update the explanation of the term and send back a successful operation message. If no match, an error message will be sent.

When the default server receives a reply message, it will first check the flag to determine which action should be followed:

1) If the flag is ‘E’, meaning the operation is fulfilled, the default server will further forward it to the client program;
2) If the flag is ‘R’, meaning the operation is fulfilled, the default server will forward this message to the client program.
3) If the flag is ‘I’, meaning the operation is fulfilled, the default server will forward this message to the client program.
4) If the flag is ‘U’, meaning the operation is fulfilled, the default server will forward this message to the client program.
   If the flag is ‘X’, default server will forward the error reply to the client program for further processing. The details of client program response are defined in 4.1.2.

5. File description

Four types of files are used in our system including: configuration file, database file,

log file, and hash table file. All four types are normal text files with '.txt' suffix.

The configuration file 'config.txt' is used by the client program to obtain necessary information to connect to the server side. The file is checked every time when client program starts. The contents contained in the 'config.txt' include port number, server IP address. The format of the file is as the following: "IP:Port\n" where the IP address and port number is divided by a colon ':'.

The database file 'data.txt' is where server side will save all the explanations to each term. The format of the file is as the following: "Term:Explanation\n" where every term and its corresponding explanation will take one line and end with '\n'.

The log file 'log.txt' is mainly for testing purposes. Both the client and server have their own log file locating on their machine. The format of log file is as the following: "Time/Operation/Result\n" where every operation that is logged will take one line space. For testing, every successful operation will be logged. The error message will trigger a log operation to save the error result for future analysis.

The hash table file 'hash.txt' is used by every server in the system. The usage of a hash table is to enable fast detection of whether a certain term does exist in the database. This is done by hashing the term to a key. The hashing method we use is to add up all the ASCII value of each character in the string and mod the total sum to 10. Therefore, every term will be mapped into an integer from 0 – 9. Our hash table file is thus simply a 10-line file that every line indicating the index key. And in each line, the value will reflect how many terms have been saved in the database which shares the same hash key. If a new term is added, the value corresponding to the key will increase by 1. This value will be deduced by 1 when the 'Remove' operation is done.

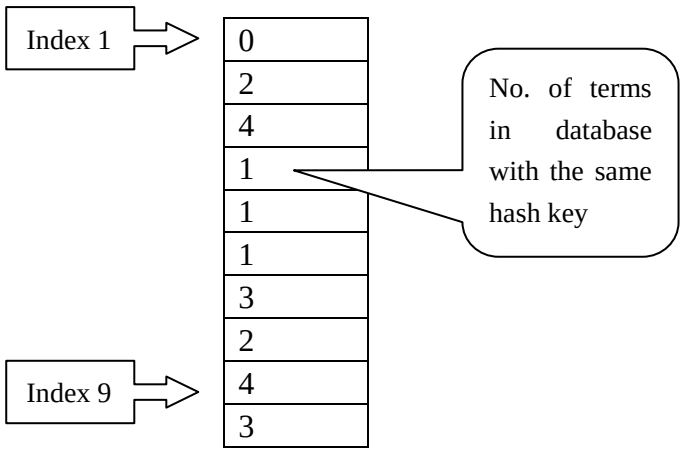

Figure 3: Hash table file format.

6. Signal handling

In our system, special signals are caught and handled by the signal handler embedded in source codes. Two signals are treated in our design as SIGINT and SIGCHLD. SIGINT will be caught once the user wants to end the program during the normal operation procedure. The signal handler will take over and terminate the TCP connection by calling shutdown() function and exit the whole program by return '-1'. Because server program will fork() child processes to carry out tasks, to avoid zombie status in the system, SIGCHLD is caught by a signal handler, and wait() function is adopted to handle terminated processes.

7. User Interface

In our system, the client program will provide an interactive user interface to the end user. The UI is designed in such a way that two levels of interfaces are adopted for different operation purposes. The first level is the overview interface where all the valid commands are listed on the screen with necessary notes to guide the user. To lower the complexity of interactions between end user and client program, all the commands are simply denoted by integer numbers as: 1. Explain; 2. Remove; 3. Insert; 4. Update; 5. Quit. The user only needs to type in a single number to issue the command according to the needs. A simple graph in Figure 4 demonstrates the first level UI.

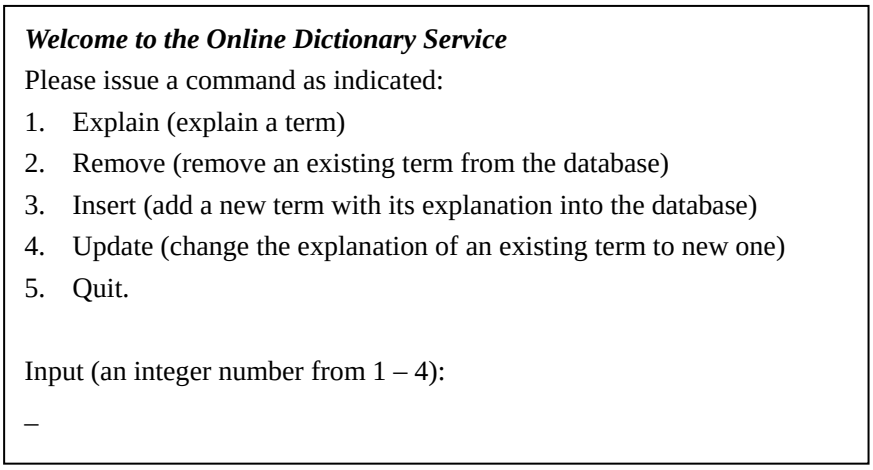

Figure 4: First level User Interface.

After the user issues a valid operational command, the UI will enter the second level. If the input at this level is incorrect, UI will stay at the first level with a warning message popped up. The warning message will instruct the user how to issue commands correctly. In the second level UI, the instructions on the screen are dependent on the previous input. A diagram in Figure 5 describes the relationships between first level and second level UI.

Take the 'Explain' and 'Insert' commands as examples to illustrate interaction syntax and semantics:

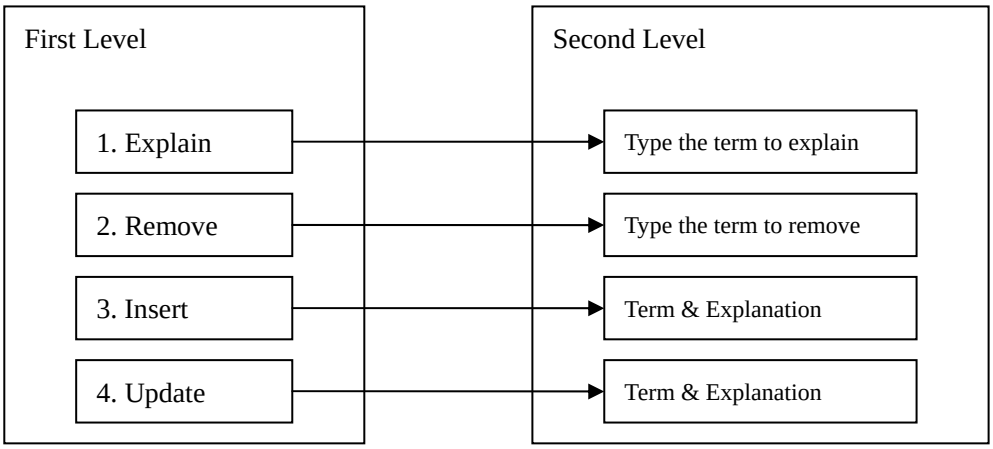

Figure 5: Relationship between two levels of UI concerning operation commands.

For ‘Explain’, in the second level, user will be required to type in the term. After the system’s processing, client program will reply to the user with either an error message or an explanation sentence for the term which the user wants to know. After one round of successful operation, UI will go back to the first level to start a new session. The same operation sequence applied to ‘Remove’ command.

For ‘Insert’, the user will be required first to type in the term. After system checking, user will be notified whether it is possible to insert such a term. If allowed, user will be asked further to type the explanation for that term. The user will be notified the successful operation result. ‘Update’ will follow the same operation sequence as ‘Insert’.

Client program will check all the inputs issued by the user. If there is any error input during the second level operation, a warning message will pop up. The users are allowed to re-input only three times. If the input times go over that limit, the program operation will stop and UI will go back to the first level. User thus should start again from the beginning.

One special type of message is the ‘Database Reply’. If the term entered by the user during ‘Explain’ operation does not exist, the ‘Database Reply’ will inform the user the current dictionary database status. The same rule will apply to the ‘Remove’ operation that if the term does not exist, user will be informed. For ‘Insert’ operation, the ‘Database Reply’ message will inform the user whether the term already existing in the database or not. If the term already exists, user will be hinted by a message “Term already existing in database. Can’t insert the same term twice! Try Update if needed.” On the other hand, the ‘Update’ operation will get a message if the term does not exist in database as “Term does not exist. Can’t update! Try Insert if needed.” The UI will if necessary, go back to the first level according to the information obtained from ‘Database Reply’.