Collaborative Spreadsheet Design Document

Wesley Gormley, Ryan Saunders, William Shupe

March 27, 2014

1 Project Overview

The goal of this project is to create a collaborative spreadsheet program that maintains cell information at a central server. Users are authenticated before accessing spreadsheets. Spreadsheets can be created and edited by a user and shared with multiple other users. Users will retrieve the spreadsheets from a remote server and view the spreadsheet on a GUI. Modifications to the spreadsheet can be made by the shared users simultaneously. When a cell is modified, that change will be propagated to all active users viewing the cell's spreadsheet. Circular dependencies will be handled server-side. Users can undo previous modifications of the spreadsheet.

The server will manage modifications to spreadsheets and user filesystems —a list of the user's accessible spreadsheets. Authentication will be done through the server. Updates to spreadsheets will be coordinated through the server. The server supports concurrent changes from multiple users. The server stores the spreadsheet information inside a database, whether local or remote (such as an SQL database).

2 Project Implementation

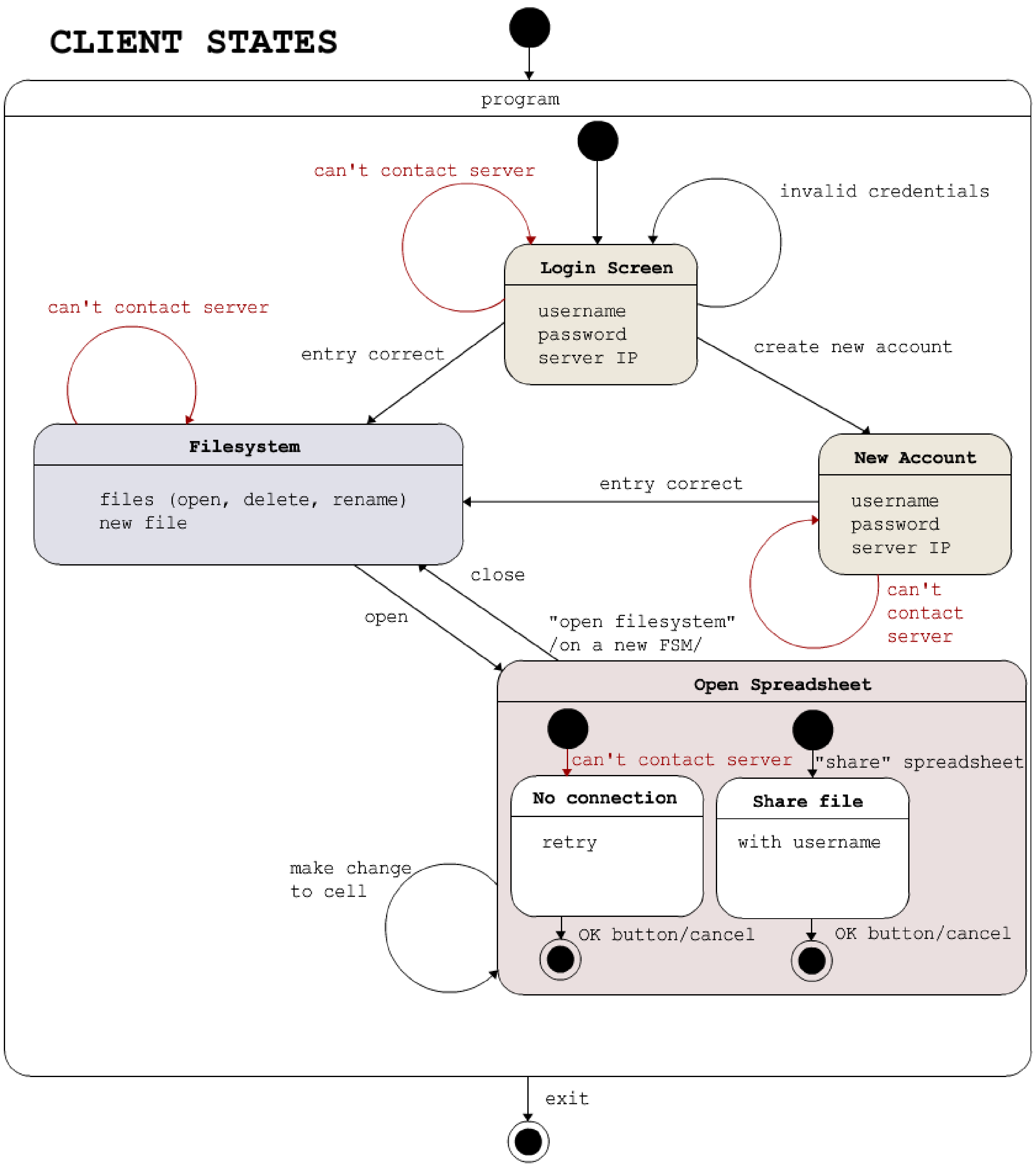
The project follows a client-server model with possibly an additional remote database accessed solely by the server. The spreadsheet project will be implemented using the existing C# spreadsheet design with specified modifications, a C++ server, and a database. More detailed information about the client and server side of the project can be found in the below sections.

Note: In the report, the term "active client" will be used to denote a client who has a given authorized spreadsheet currently open and should be able to update and receive updates for that spreadsheet.

2.1 Client

2.1.1 State Diagram for Client

The reader is referred to the state diagram of the client program in Figure 2.1 for a general understanding of the specifications laid out in section 2.1.



[Figure 2.1]

2.1.2 Starting Program and User Authentication

The client-side program initiates with a means to input a username, password, and server IP address. The client program should assume the server is running on port 44441. The login credentials will be verified server-side. A user should also be able create a new account by providing a unique username and password. The server will verify that the username is indeed unique. The username and password must only contain letters and numbers —no spaces. This should be handled client-side.

When a client provides the server with the credentials it should first create a TCP connection with the server. This connection should be maintained until the client closes the program or a DBFAILURE message is sent from the server. At any time the client may cancel the connection attempt if the server has not yet confirmed the entry.

The messages that will be used to communicate between client and server are specified in section 2.3.1 and 2.3.2.

2.1.2 Filesystem

Once the user's credentials are validated, the server will send the client a list of all the spreadsheet files the user has access to. The client will have the option to rename, delete, or open a spreadsheet by communicating the requests to the server.

When a client opens a spreadsheet, the client sends the spreadsheet id to the server and will anticipate to receive multiple messages from the server which explain what cells contain content for that spreadsheet. The client should then display a spreadsheet GUI with cells that contain the information from these messages.

The client should also be able to create a new spreadsheet by requesting a new spreadsheet name to the server. The spreadsheet names do not need to be unique. Spreadsheets are identified by a spreadsheet id. When the client creates a new spreadsheet name, the server will send the user with a spreadsheet id for that spreadsheet. The client should then display a fresh spreadsheet GUI.

If the client renames a spreadsheet, the change will be sent to the server. The spreadsheet id will remain the same.

If the client deletes a spreadsheet, the client will notify the server. The filesystem should no longer display that spreadsheet for that user.

The server will always respond to one of these requests with some type of confirmation message. If the expected confirmation message does not come through the socket in a given time, the client should notify the user that it cannot reach the server. See section 2.1.4 for a more detailed explanation of the timeout.

The messages that will be used to communicate between client and server are specified in section 2.3.1 and 2.3.2.

2.1.3 Spreadsheet

With a spreadsheet GUI open, the client will be able to modify cells, share the spreadsheet, undo operations, or receive updates about the spreadsheet's cells. A user should be able to modify the content of a cell. If the content of the cell is changed, and the user commits the changes, the change should be sent to the server. The actual changes are only made to the spreadsheet once the server sends back to the client a CELL type message with the changed cell name and with the expected changes to the cell's content. For example, if the client sends an update, *"UPDATE 001 A1 5"* —where 001 is the spreadsheet id, A1 is the cell name that is being modified, and 5 is the new content of that cell— then the client must wait for the message, *"CELL 001 A1 5"* before the change should be made to the cell. If the client does not receive a matching response in a given time, the client should notify the user. See section 2.1.4 for a more detailed explanation of the timeout. If the server discovers a circular dependency with the pending update, the server will send a DEPEND type message to the client. The client should notify the user, and not allow that change to be made on the spreadsheet.

The client should also be able to automatically update cells' content as such updates are received from the server. Many active users could be modifying the spreadsheet at one time, and the changes should be replicated on each client.

Users may undo a previous modification on the spreadsheet. The client will contact the server requesting the undo. The server will send back a cell update of the last modified cell with its previous content. In other words, the undo is handled by the server. The user is free to undo operations made on the spreadsheet as far back as the spreadsheet has been opened by the first active user for that session.

A user may share the spreadsheet with another user by specifying a username to share with. The client will send the share request to the server and the server will add the given spreadsheet to that shared user's authorized spreadsheet list.

The messages that will be used to communicate between client and server are specified in section 2.3.1 and 2.3.2.

2.1.4 Unable to Connect to Server

Anytime the client sends a message back to the server, there is a corresponding response it expects from the server. If the connection between the server and client goes down, the server either never gets the client's message or the client never gets the server's message. Regardless, the client will be waiting for a response that is not coming.

To recover from possible connection failure, a timer for each message to the server should be started with a given timeout interval. When the timer goes off, the client should notify the user of a connection loss. The timeout interval value can be freely determined by the reader. There are many possible ways to calculate an effective timeout interval, and the reader is encouraged to search timeout interval calculations on the web.

2.1.5 Client Receives Error Message.

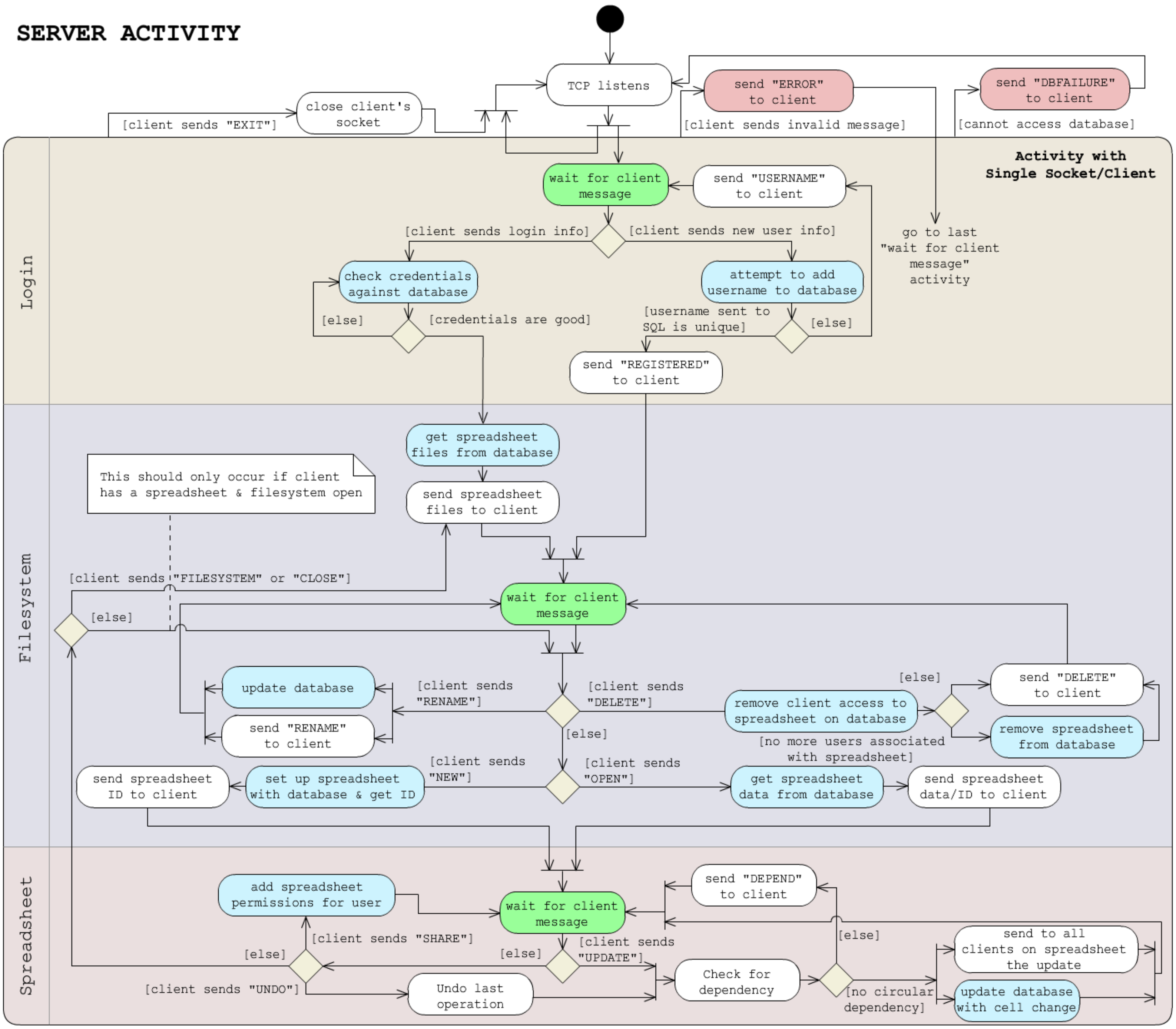
There are two possible error messages that the client can receive from the server. Each should be handled differently. The first one is if the server sends ERROR. In which case, the server is communicating to the client that the previously sent message from the client was invalid. How the client handles such errors is up to the reader, but should be handled in a helpful manner to the user. The second possible error is a DBFAILURE message sent from the server. This message is sent when server is unable to access its database. Since all the information that the client uses is accessed from the database, there is nothing the client or server can do until the database is working again. When the client receives this message, it should notify the user, close the TCP connection by sending EXIT to the server, and ask the user for login credentials again. If the user wishes to log in again, another attempt to access the database will occur.

[continues on next page]

2.2 Server

2.2.1 Activity Diagram for Server

The reader is referred to the activity diagram of the server program in Figure 2.2 for a general understanding of the specifications laid out in section 2.2.



SERVER ACTIVITY
TCP listens
close client's socket
[client sends "EXIT"]
send "ERROR" to client
[client sends invalid message]
send "DBFAILURE" to client
[cannot access database]
Activity with Single Socket/Client
Login
wait for client message
send "USERNAME" to client
go to last "wait for client message" activity
[client sends login info]
[client sends new user info]
check credentials against database
attempt to add username to database
[else]
[credentials are good]
[username sent to SQL is unique]
[else]
send "REGISTERED" to client
Filesystem
get spreadsheet files from database
send spreadsheet files to client
This should only occur if client has a spreadsheet & filesystem open
[client sends "FILESYSTEM" or "CLOSE"]
[else]
wait for client message
update database
send "RENAME" to client
[client sends "RENAME"]
[client sends "DELETE"]
[else]
remove client access to spreadsheet on database
[else]
send "DELETE" to client
remove spreadsheet from database
[no more users associated with spreadsheet]
send spreadsheet ID to client
set up spreadsheet with database & get ID
[client sends "NEW"]
[client sends "OPEN"]
get spreadsheet data from database
send spreadsheet data/ID to client
Spreadsheet
add spreadsheet permissions for user
wait for client message
send "DEPEND" to client
[client sends "SHARE"]
[else]
[else]
[client sends "UPDATE"]
[else]
[client sends "UNDO"]
Undo last operation
Check for dependency
send to all clients on spreadsheet the update
[no circular dependency]
update database with cell change

[Figure 2.2]

2.2.2 Connection to Client

Since the server is literally a server in the client-server model, the server must begin by continually listening for incoming TCP connection requests. The listening socket is bound to port 44441. When the server receives a TCP connection, it should accept the connection onto a concurrent connection socket, and listen again for another incoming request. Each socket should be on its own thread as to allow an asynchronous TCP server.

2.2.3 Connection to Database

The server contacts a database to retrieve or send information regarding the spreadsheets. It is up to the reader to decide on how the database should be implemented. Two possible suggestions would be to have a remote SQL database, or to keep a database file system on a local disk.

2.2.4 Authorizing User

Once a TCP connection is established between the server and client, the server waits for a client to either send new account credentials or log in credentials. Usernames must be unique. If the client sends new account credentials, the server will query the database for the provided username. If the provided username already exists, the server will inform the client of an invalid username. If the username or password contains characters that are not letters or numbers (which should have been checked client-side), the server will send to that client an ERROR message. Otherwise, it will add the username and password to the database, and inform the client that the username was unique. If the client sends log in credentials, the server will query the database to see if the credentials are valid. If not, it will let the user know invalid credentials were provided. Otherwise it will provide the user with a list of his/her authorized spreadsheet files.

Once the user is verified and “logged in”, the server should keep track of which user is on which socket.

The messages that will be used to communicate between server and client are specified in section 2.3.1 and 2.3.2.

2.2.5 Managing Spreadsheet Files

Once a user has successfully logged in and is given his/her list of spreadsheet files, the client can send many different commands to perform actions on these spreadsheet files. The client will have the options to rename, delete, or open a spreadsheet file by communicating the requests to the server. When a client asks for a spreadsheet to be opened, the server will respond by sending a chain of messages with cells for that spreadsheet that have saved content.

When the client asks for a new spreadsheet, the server will send the user with a spreadsheet id for that spreadsheet. The spreadsheet id should be unique for all spreadsheets stored in the database.

When the client notifies the server that it has renamed a spreadsheet, the spreadsheet name for the spreadsheet with the specified id will be updated on the database.

When the client notifies the server that it has deleted a spreadsheet, one of two actions will occur. The server will query the database asking how many users are associated with that spreadsheet. If other users are associated with the spreadsheet, only the user who has requested the delete will no longer be associated with the spreadsheet. If no other users are associated with the spreadsheet, it is to be removed from the database.

The server will always respond to one of these requests with some type of confirmation message.

The messages that will be used to communicate between client and server are specified in section 2.3.1 and 2.3.2.

2.2.6 Open Spreadsheets

The server should keep track of which spreadsheets are opened, and which clients are actively connected to those open spreadsheets. A client can send requests to open a new filesystem, share a spreadsheet, close a spreadsheet, undo changes, and make changes to the spreadsheet.

A user can access multiple spreadsheets at a given time. When the client request the filesystem again, the user is allowed a second window. One window displays the spreadsheet previously opened, the other displays a new filesystem. The server should resend the list of spreadsheet files first provided to the user in section 2.2.5.

When the server receives a share request message from a client, the spreadsheet should be added to the provided username's list of authorized spreadsheets.

When the client is finished with a spreadsheet, a CLOSE message is sent and that client should be removed from the list of active clients for that spreadsheet.

2.2.6.1 Revision History and the Undo Functionality

A stack of the revision history —i.e. the undo history— should be kept for all spreadsheets. The revision history should start when the spreadsheet was first created. If an update to a particular cell is requested, but the update does not change the cell's content, it should not be added to the revision history. Otherwise, the cell that was modified, and the new content, should be placed in the revision history.

When a client requests an undo, the server pops the last modification off of the stack. Only one action at a time should be done on the revision history.

Though the revision history is stated to be a stack, the implementation is open to the reader.

2.2.6.1 Updating Cells and Circular Dependency

When the client modifies a cell in the spreadsheet, a message will be sent to the server. The server should check the spreadsheet for circular dependencies. If the update would create a circular dependency, a DEPEND message is sent back to that client. Otherwise, the server updates the database with the change and sends this update to all active clients for that spreadsheet. Checking for modifications to a spreadsheet should be done one at a time.

2.2.7 Invalid Message from Client

If at any time the server receives a message from a client that is invalid, an ERROR message should be sent to that client. Since a user never directly sends messages to a server, an ERROR message should only be sent if there is something wrong with the implementation of the client program.

If a meaningless message is sent, there is an obvious communication exception between the server and client. If the client is sending a message that is unexpected at a certain stage, then the client is either not correctly advancing the states of the client program, or a connection lost occurred during a state change and a duplicate message is sent. If the client sends duplicate messages, the server will handle them as specified in section 2.2.8.

To resolve this last issue (the only issue that needs to be resolved on the server end), the server will resort back to the last “wait for client message” activity stage as specified in Figure 2.2.

2.2.8 Connection Loss to Client and Database

Connection loss to a client isn’t handled on the server-side. To read about how connection loss is handled, the reader is referred to section 2.1.4. If a connection loss does occur however, the server can receive duplicate messages. The way a server handles messages from clients is such that there will be no issues if a message is received twice. The server will either send back an ERROR message (as specified in 2.2.7) or query the database a second time, which will cause no harm. The only exceptions for this is when the user requests a new spreadsheet twice. In this case, two new spreadsheets will be created; a minor issue.

Access to the database, on the other hand, is a rather large issue. Without access, the server cannot respond to almost all client requests. If access cannot be given to the database, a DBFAILURE message should be sent to all clients that are sending messages to the server. The server should expect back a EXIT command from the client, —that is, to close the socket— but the server may do so without the clients permission if the reader wishes. Any attempt to retry accessing database will be made when a users attempts to re-login.

2.3 Client/Server Communication

A TCP socket will be used to pass all messages from the server to the client and vice versa. The standard for these communications can be found below. The message protocol ensures that clients and servers can effectively communicate with each other without understanding the other's design —i.e. they are interchangeable.

All messages should be terminated with **\r\n** (CRLF). All messages should follow the formats exactly as explained in 2.3.1 and 2.3.2. For example, if the user wishes to send login credentials he would send the message, "LOGIN alice223 mypassword\r\n".

2.3.1 Client to Server Communications

CLOSE <spreadsheet id>

- Sent when a user closes a specific spreadsheet, letting the server know not to send anymore updates to the client about that spreadsheet's changes.

DELETE <spreadsheet id>

- Sent when a user is attempting to delete a spreadsheet he/she has access to.

EXIT

- Sent when a user closes their application, stating the server may close the connection.

FILESYSTEM

- Sent when a user is attempting to access the list of spreadsheets he/she is authorized to view.

LOGIN <username> <password>

- Sent when a user is attempting to login.

NEW <spreadsheet name>

- Sent when a user is attempting to create a new spreadsheet.

OPEN <spreadsheet id>

- Sent when a user attempts to open a specific spreadsheet.

RENAME <spreadsheet id> <new spreadsheet name>

- Sent when a user is attempting to rename a previously created spreadsheet.

SHARE <spreadsheet id> <username to share with>

- Sent when a user wants to share a spreadsheet with a different user.

UNDO <spreadsheet id>

- Server should undo the last modification to the spreadsheet.

UPDATE <spreadsheet id> <cell name> <cell content>

- Sent when a user changes the content of a cell.

USER <username> <password>

- Sent when a user is attempting to create a new user profile.

2.3.2 Server to Client Communications

CELL <spreadsheet id> <cell name> <cell content>

- Sent to notify a client that a change has been made to the spreadsheet and must be displayed locally.

CREATED <spreadsheet id>

- Sent when a new spreadsheet was created successfully, the internal spreadsheet id is also sent.

DEPEND <spreadsheet id> <cell name>

- Server found dependency with previous update from a client, send message to client to indicate a circular dependency was found and no update was made to that cell.

DBFAILURE

- Message sent when server is unable to access the database and no further work can be done until access can be restored.

DELETE

- Sent when a spreadsheet was successfully removed from the user's list of accessible spreadsheets.

ERROR

- The previously sent message from the client was invalid.

FILES <list of spreadsheet ids, names>

- Sent when a users requests to access his/her filesystem. A list of the spreadsheets that can be accessed by this user is returned. The spreadsheet name is separated from the spreadsheet id with a comma, and the spreadsheets are separated with a semicolon.
    - example, "FILES spreadsheet1, 00001; tax spreadsheet, 00014"

OPEN <spreadsheet id>

- Sent when a spreadsheet is opened. This will be followed by CELL commands for each cell that contains data in the opened spreadsheet.

PASSWORD

- Sent when a login attempt has failed.

REGISTERED

- Sent to the client when the new user has been registered.

RENAME

- Sent when a spreadsheet was renamed successfully.

REVERT

- Sent to notify a client that his/her undo message was successful. Immediately followed by a CELL message to all active users about the change.

USERNAME

- Sent when a new user attempts to register a username that already exists.

2.4 Modifications of Original Spreadsheet

Though much of the functionality of the original spreadsheet design remains the same, there are a few things to consider. First, the cells should be in a range from A1 to Z99. Any attempt to modify a cell outside this range should be rejected by the server and an ERROR message should be sent back to the client.

The menu buttons for open, save, and new should be replaced with an option to open a new filesystem. If the reader wishes to keep local spreadsheets, it is up to him/her to implement it without interfering with this protocol.

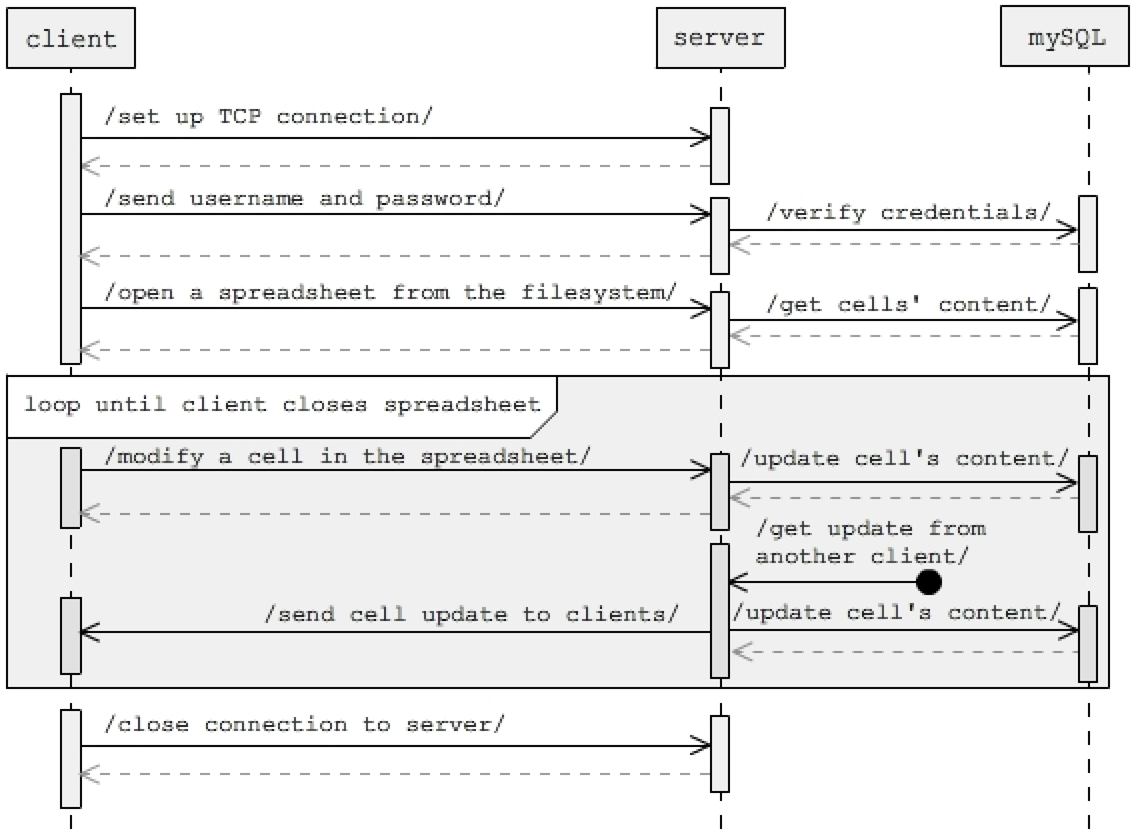
The individual clients should not check for circular dependency, rather they should rely on the server for this check.

Finally, the reader should be aware of how his/her spreadsheet “updates” cells. If the reader has implemented an ability to navigate the spreadsheet with shortcut keys, no update should be sent to the server unless a spreadsheet cell has actually been modified.

[continues on next page]

3 Typical Session Example

In order to help clarify any unclear standards for this design, an example client-server session will be explained below. The reader may refer to the Figure 3.1 for a visual understanding of the procedure.


client
server
mySQL
/set up TCP connection/
/send username and password/
/verify credentials/
/open a spreadsheet from the filesystem/
/get cells' content/
loop until client closes spreadsheet
/modify a cell in the spreadsheet/
/update cell's content/
/get update from another client/
/send cell update to clients/
/update cell's content/
/close connection to server/

[Figure 3.1]

3.1.1 Client Login

The server should be actively listening for incoming TCP connection requests at any time. The user, Alice, opens the client program and enters her username, password, and the IP address of the server. A TCP connection is made with the server. The message, *"LOGIN Alice ImaPassword12\r\n"* is sent from the client, and the server verifies the user's login credentials against the credentials on the database.

3.1.2 Client Filesystem

The server sends to the client her list of authorized spreadsheets, *"FILES spreadsheet1, 12201; spreadsheet2, 01562; excel spreadsheet, 00012\r\n"*. Alice's client window should display the possible files to choose from. Alice desires to open *"spreadsheet1\r\n"*, and her client sends the message, *"OPEN 12201\r\n"* to the server.

3.1.3 Client Spreadsheet

Once the server receives the message it will query the database for the spreadsheet's cell data. The server will then send a series of messages to the client about the cells' content, as an example, *"CELL 12201 A1 42\r\n"*. As the client receives these messages, it should propagate the opened spreadsheet's cells with the given content values.

When Alice modifies the content of cell A1, her client sends a message, *"UPDATE 12201 A1 43\r\n"* to the server. The server checks for circular dependencies on that spreadsheet, then contacts the database and updates the cell's content and sends the message, *"CELL 12201 A1 43\r\n"* to all clients currently viewing spreadsheet 12201. When Alice's client receives this message, it can confirm that the server received the update. If the client doesn't receive this message, a timeout may occur and Alice will be notified that there was a connection problem with the server.

Alice can also receive messages from the server about her open spreadsheet's cells if other active users are modifying that spreasheet. Anytime the server sends a CELL 12201 <cell name> <cell content> message to Alice's client her client will automatically update the spreadsheet's cell with the given server information.

Alice can also open a second window by sending the server, *"FILESYSTEM\r\n"* in which case the server will send back to Alice, *"FILES spreadsheet1, 12201; spreadsheet2, 01562; excel spreadsheet, 00012\r\n"*. The server will continue to send updates about spreadsheet 12201 to Alice.

Once Alice is done with spreadsheet 12201, she sends the message, *"CLOSE 12201\r\n"* to the server, and the server ceases sending Alice's client the updates about that spreadsheet.

3.1.4 Closing the Program

When Alice closes the program, her client will send, *"EXIT\r\n"* to the server, and the server will close the socket with Alice's client.

3.2.1 Conclusion of Example

Though there are many cases and possible scenarios for interaction between a client and server, other cases should follow a pattern similar to the one in section 3.1. For addition communication protocols the reader is referred to sections 2.3.1 and 2.3.2.

4 Conclusion

Though some of the details of this design seem overly complex, we assure the reader that the complexity is only in the thoroughness of what is required for the project. A great attempt was made to keep things as abstract as possible while maintaining the already in-place functionality of a spreadsheet. Though it could be debated on handling circular dependencies client side, by handling all dependencies one at a time on the server side, there is less room for exceptional cases and fewer server/client communications when confirming a cell change. Finally, the simplicity at which the program handles errors and connection issues helps keep the amount of exceptional cases to a minimum.