

**Hot Cocoa Communication Protocol
COCOPRO**

**Dylan Habersetzer
Dylan Quach
Sabina Miani
Erickson Nguyen
Tyler Wood**

5 March 2021

Introduction

The Hot Cocoa Communication Protocol (COCOPRO) is designed for the implementation of the server client network functionality allowing multiple clients to connect and simultaneously edit persistent spreadsheets. Clients will be able to collaborate on spreadsheets to edit. Undo and cell revert functions are allowed, but deletion of spreadsheets is strictly prohibited. It is assumed the client-side functionality is already established; only networking implementation is necessary for communication with the server. However, it stands that the client must be able to provide sufficient information and structure to connect, send, receive, and display correct data to and from the server. Packets sent through something other than the aforementioned fully functioning client will be allowed assuming the client has proper syntax on connection.

COCOPRO Architecture

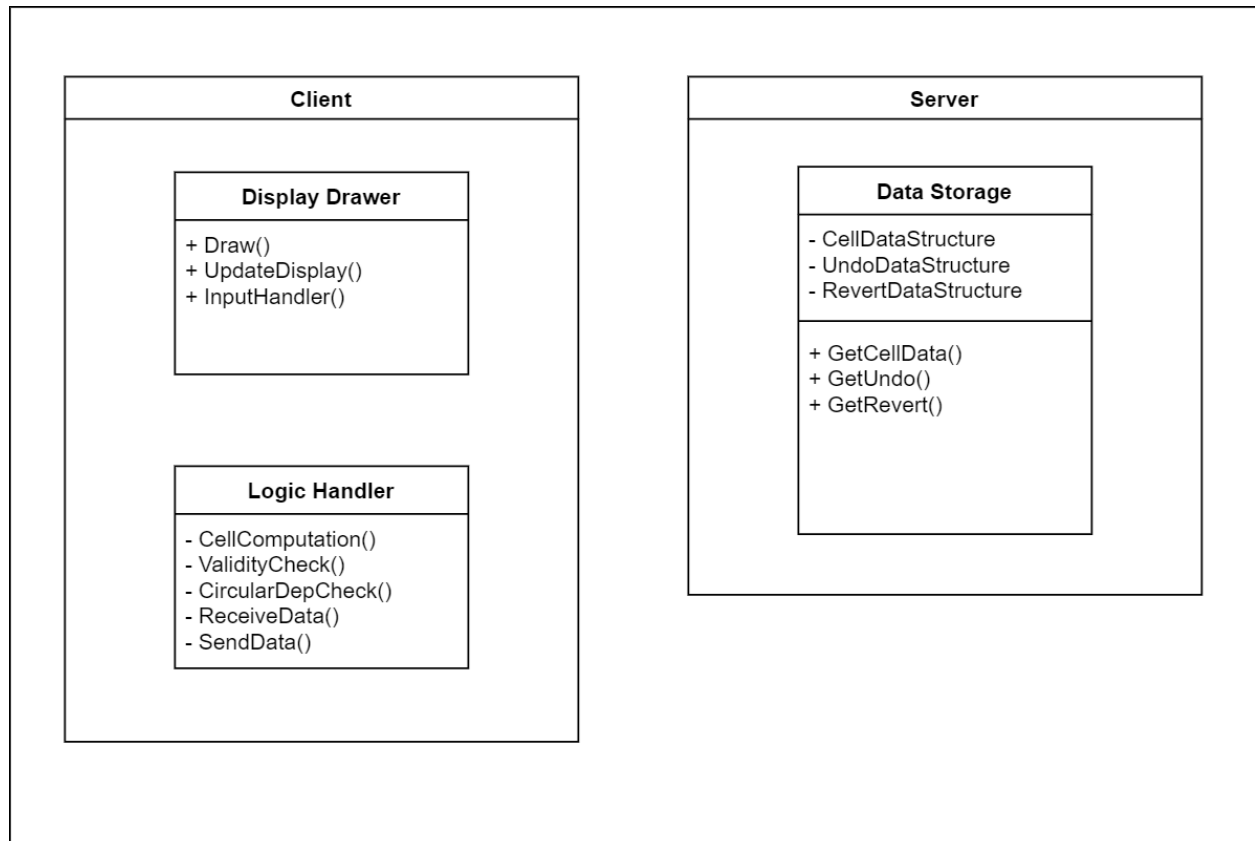


Figure 1. Client Server Architecture with Outlined Functional Components

The general project architecture separates the concerns of the client and the server. The server will contain all the data required to maintain spreadsheets, as well as the means to inform any clients of updated spreadsheets. This means the server contains cell contents, cell revert snapshots, as well as records of the entire spreadsheet to be

used when 'undo' is called. For further information regarding cell reverts and spreadsheet undo's, see Figure 2. Furthermore, the server will send all the users the most up-to-date spreadsheet. The client will handle drawing and displaying the interface, mathematical logic, and content errors such as circular dependencies. The server will check for incorrect formatting errors. However, both the client and server will check for communication errors. The client connects to the server via a spreadsheet name and a viable IP address. The interaction between the server and client will be limited to send and receiving of data stored within the server. The client will handle all forms of mathematical or logical errors. Furthermore, the client will handle updating the display when there are any changes.

In regard to how multiple spreadsheets' data will be stored, each spreadsheet object will have a block of memory allocated for use. To access a specific spreadsheet's data, a version name or number will be used as an identifier to gain access to a specific spreadsheet.



Undo		The last edit done on any cell in the spreadsheet is restored to the previous content value.
Revert		Restore currently selected cells content to whatever it last was

Figure 2. Explanation of Undo and Revert Actions

Communication Protocol

In order to send information between the client and the server, the COCOPRO sets a specific protocol to connect the client and server to each other. XML will be used to convey data utilizing TCP/IP data protocol, preventing packet data loss. Making stable connections between the client and the server is paramount to separating the protocol into two stages, the initial handshake and the edit/send phase.

To start the initial handshake, the server starts and opens a listener capable of accepting clients. After turning on the client and before initiating the connection, the client has to input the IP address of the spreadsheet server as well as a string representing the name of the spreadsheet they wish to access. If the name does not exist, the server will create a new spreadsheet. The client will use the specified port number to connect to the server. First, the client will send an XML message consisting of the name of the spreadsheet they wish to access. So long as the provided information is correct and the server receives the request, the server will respond by

sending all of the current contents of the specified spreadsheet to the user in XML format ending the handshake (see *Initial Client-Server Handshake* for further detail).

Assuming the connection is stable, the protocol will maintain the edit/send phase after the handshake ends. During this phase, clients can request changes, undo the last edit made on a spreadsheet, and revert a single cell to a previous state. While the client is connected to the server, the server will be listening for any type of communication from any clients already connected or attempting a connection while sending relevant information to other clients whenever spreadsheet edits are performed.

In order for the spreadsheet to determine what kind of change the client wants to make, messages utilize true and false statements to tell the server whether a user requested a specific type of change. The entire message is sent in XML, the encapsulating tag being `<spreadsheet>`. The spreadsheet tag also has an attribute called *version*, which indicates what the name of the spreadsheet is. Depending on the change that the client wishes to make, the change will be sent as an XML Boolean tag labeled as `<cellEdit>`, `<undo>`, and `<revert>`. The changes made to the spreadsheet will depend on whether these labels are read as true or false. The server edits the spreadsheet before sending messages to other clients updating their versions of the spreadsheet. The message sent to each client has the cell name, as well as the new contents of that cell (see *Client-Server Networking* for further detail).

Initial Client-Server Handshake

To connect the server and client, both must be started before attempting a connection. The server will be held on port number 56665. The communication process will use TCP with UTF-8 to assure whole message packets are received. Each user must provide the IP address of the server and the name of the spreadsheet to which they wish to connect via the client.

Assuming the server has been started and is continuously (non-blocking) listening for new client sockets, a client can be started. The server address and a unique spreadsheet name, between 6 and 30 characters in length containing only letters, numbers, and underscores, must be specified by the client before the client can send a request to the server to open the specified spreadsheet. The client must send the spreadsheet name in XML format. If there are any connection errors, they will be handled quietly (see *Errors* for further explanation), otherwise, the server will open the designated spreadsheet. If the name does not belong to a previously created spreadsheet, a new blank spreadsheet will be created on the server. After the name exists on the server belonging to a spreadsheet, the server will send the contents of the spreadsheet in XML format (see Figure 3 and 6). Only the non-empty cells will be sent to the client. If the spreadsheet is blank, an XML message as shown in Figure 3 will be

sent. Once the full spreadsheet is received, the handshake is complete, and the client may begin sending edit commands to the server.

```
<spreadsheet version = "nameOfSheet">
</spreadsheet>
```

Figure 3. XML Example of Sending an Empty Spreadsheet

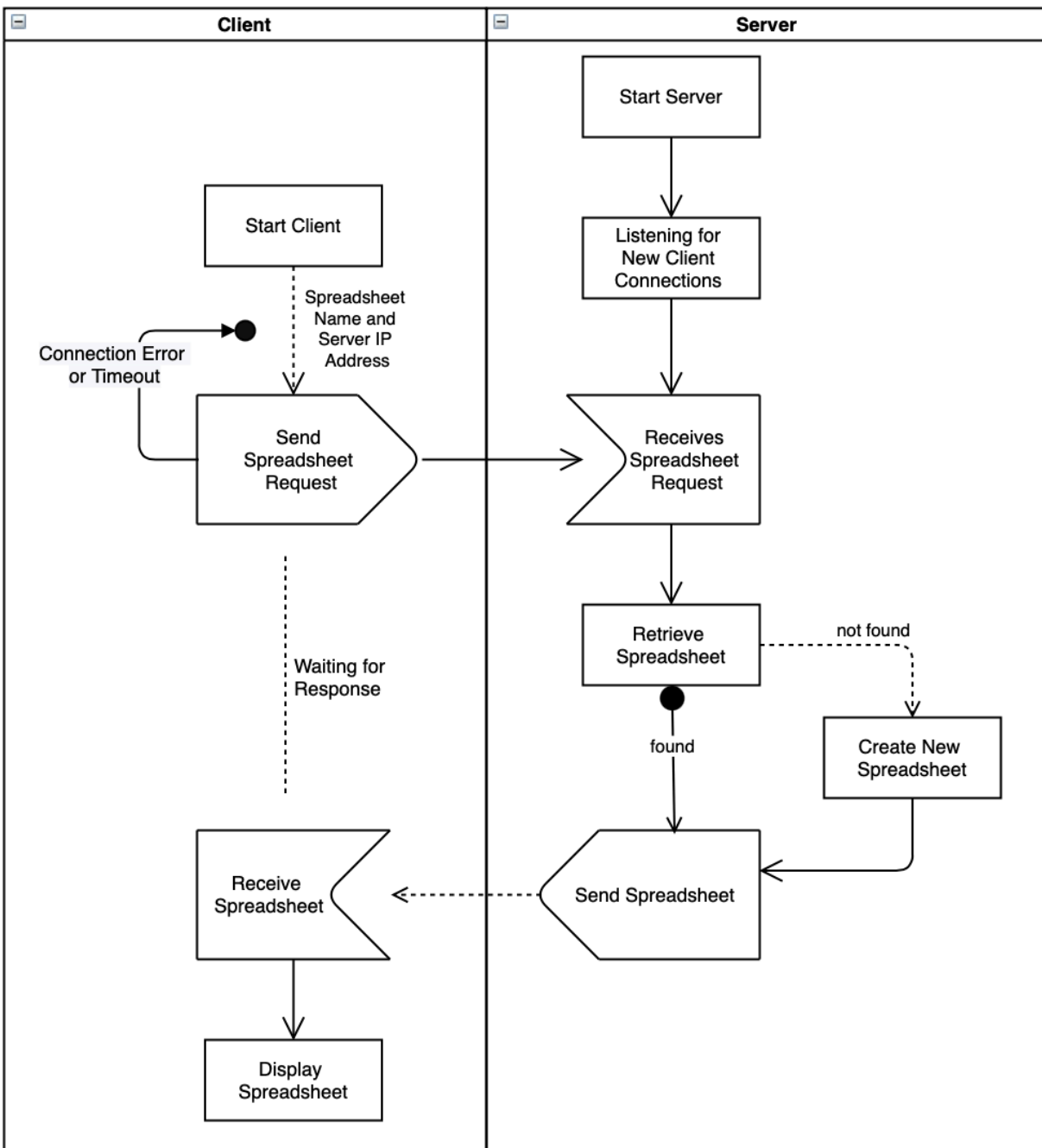


Figure 4. UML formatted Diagram of the Initial Handshake

Client-Server Networking

When edits are being sent from the client, there is one XML send to the server for each client command: cell edit, undo, revert. Each send will contain the name of the spreadsheet being edited, the type of command, and the cell information: name and contents if edited (see Figure 5).

```
<spreadsheet version = "nameOfSheet">
  <cellEdit> true/false </cellEdit>
  <undo> true/false </undo>
  <revert> true/false </revert>
  <cell>
    <name> nameOfCell (e.g. A1, Z7, etc.) </name>
    <contents> string or int go here </contents>
  </cell>
</spreadsheet>
```

Figure 5. XML Example of Client Commands sent to the Server

When edits are sent from the server to each client connected to the same spreadsheet, one XML send will be sent. An XML send includes the spreadsheet version name and the cell information being changed, the name and the contents of the cell. An example of a message can be found in Figure 6.

```
<spreadsheet version = "nameOfSheet">
  <cell>
    <name> nameOfCell </name>
    <contents> string or int go here </contents>
  </cell>
</spreadsheet>
```

Figure 6. XML Example of a Message sent from the Server to a Client with Spreadsheet changes

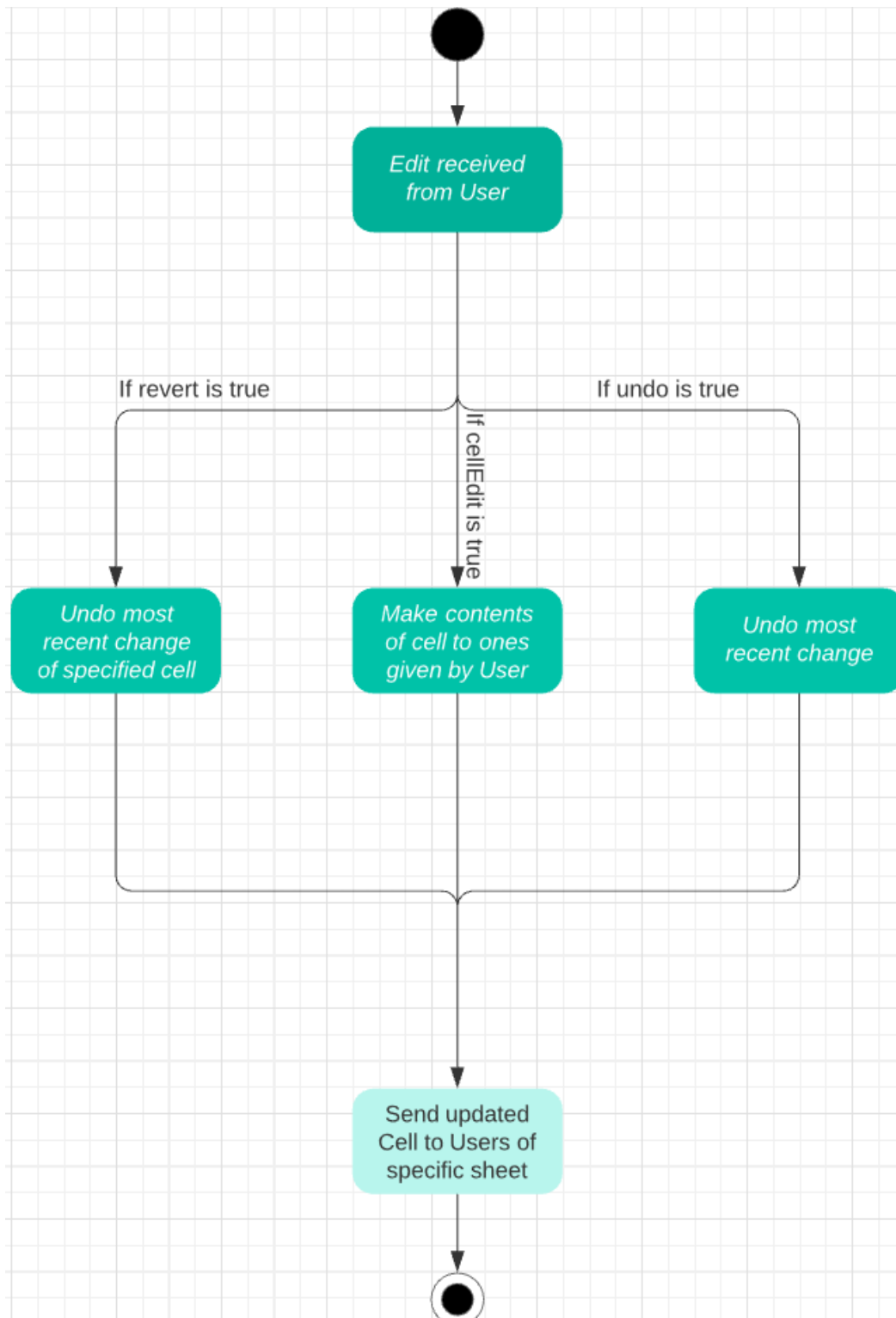


Figure 7. An Overview of the Interaction Between Clients and the Server

The edits sent by the client will be sent to every client (including the client sending the edit) currently using that spreadsheet. For example, if clients 4, 11, and 17 are

simultaneously editing the spreadsheet “Names” and client 4 requests a change to a cell, the server will only send the change to clients 4, 11, and 17, and no other clients on the network (see Figure 8).

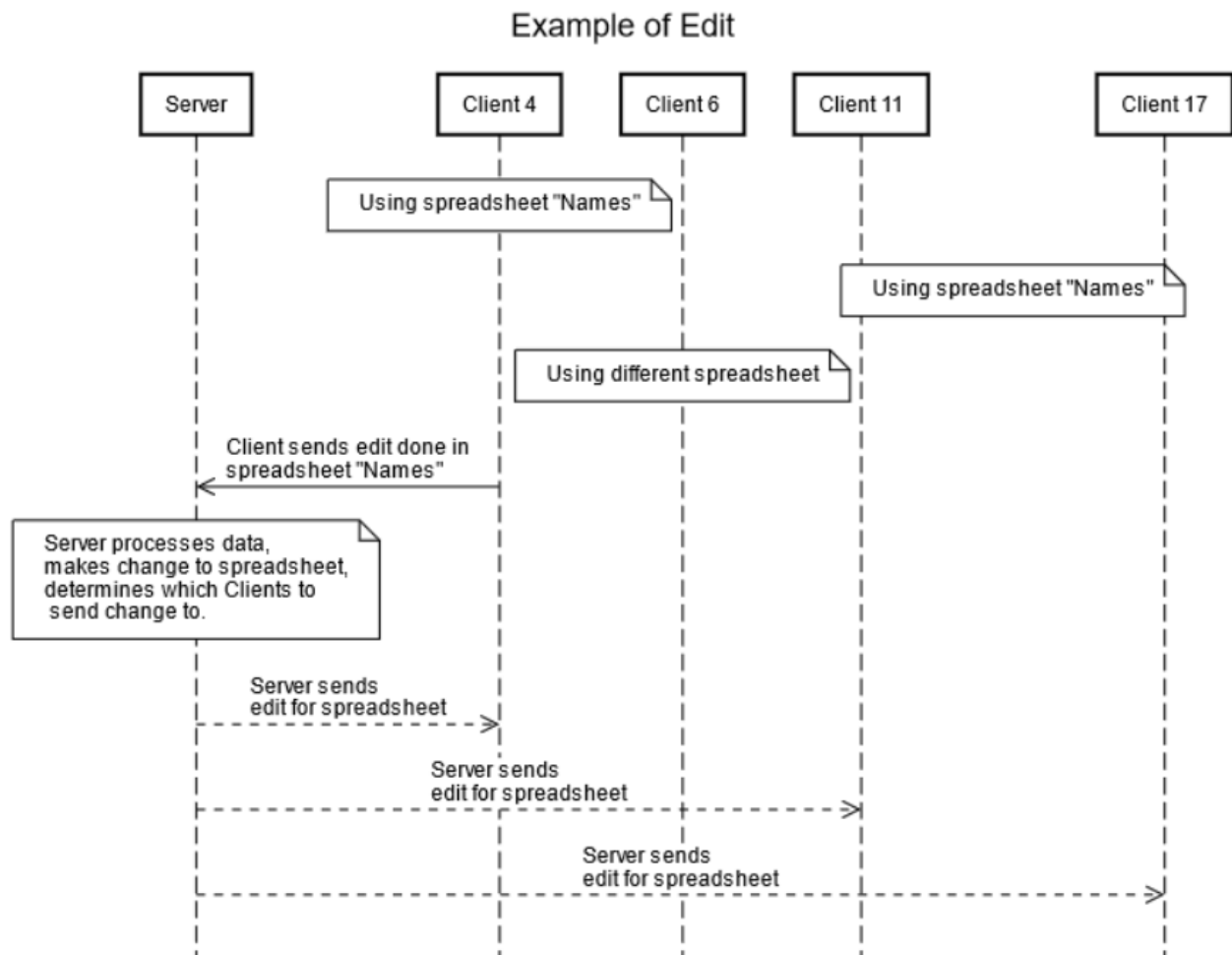


Figure 8. UML Documentation based on Multi-Client, Single-Server COCOPRO Spreadsheet Functionality

Errors and Connection Handling

Handling errors will be contained within the server and/or client depending on the scope of the error. Connection errors will be handled quietly on both the server and the client without blocking further communication between the server and any other connected clients. Spreadsheet cell validity and error checking will be handled within the client. As shown in Figure 8, if packets sent are invalid, they will be ignored.

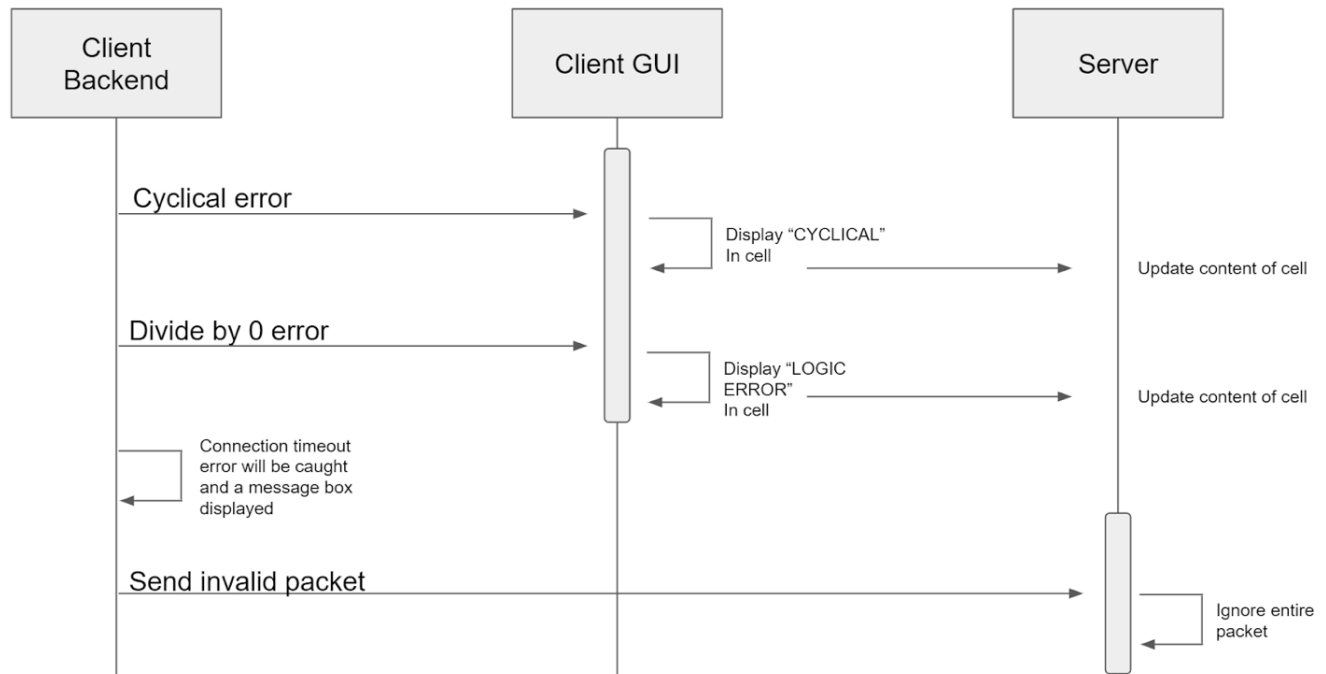


Figure 9. UML Documentation for Processing Connection and Logical Errors

Connection Errors

A client will attempt to connect to a server for 20 seconds before displaying a message box saying, "Connection Timeout Occurred". Any other error that occurs on the server should be non-blocking, meaning all processes should continue running after the error is caught and handled.

If the client closes, the spreadsheet will persist on the server and the server will not send any information regarding the close to any remaining clients on the spreadsheet; the close is quiet on both client and server. If the server closes, all data saved while that server was open will be lost, meaning that while the server remains open, all spreadsheet information will be saved. Any clients connected to the server at the time of its closing will be disconnected and a message box will appear for the client displaying an appropriate server disconnected message. Any spreadsheet the client was working on will not be saved, including the cell contents they may have been editing.

Packet Errors

TCP guarantees all packets will be sent to the server in the correct order they were sent; however, it is possible for the packet being received by the client to be incomplete. In this case, the client should ignore the packet and run a validation method (similar to the message the server sends during the initial handshake to a new client with the full spreadsheet) to check the client spreadsheet for identical contents for every cell in the

spreadsheet (see Figure 3). If other packet errors occur, the server will run the validation method.

A packet is considered invalid and will be ignored if any of the following are true:

- The packet has invalid XML syntax
- The XML packet does not contain all the fields required to update cell contents
- The packet tries to edit a cell, undo, or revert at the same time

Spreadsheet Errors

Circular dependencies are handled by the client displaying “CYCLICAL” in all the cells tied to the dependency. Logical and mathematical errors are also handled similarly; relevant messages will be displayed in the cell when the error is identified by the client (see Figure 9).

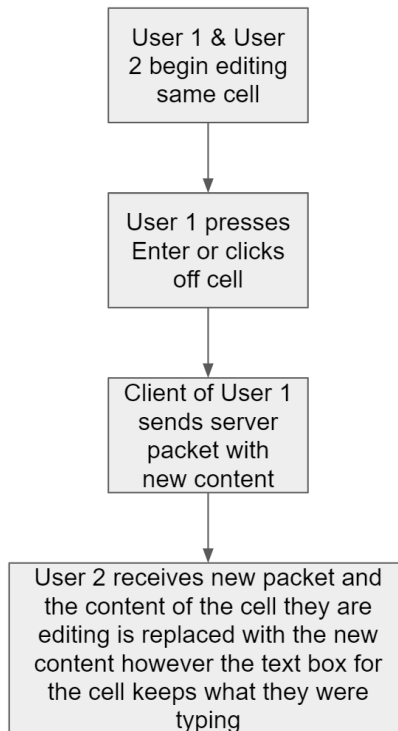


Figure 11. Visual Representation of Actions taken While a Cell is Edited by Two Users Simultaneously

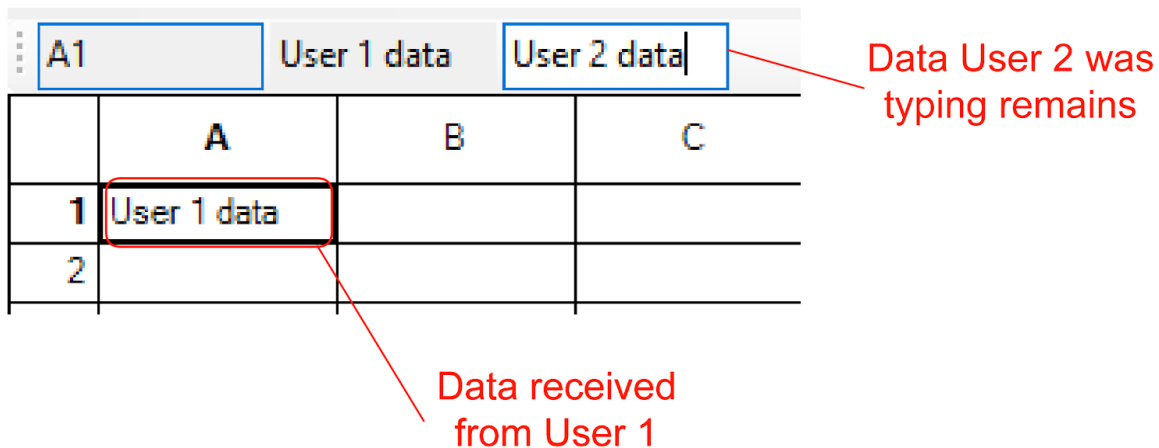


Figure 10. Example of Two Users Editing Cell A1

This type of editing guarantees users will not lose what they were typing during the duration the edit from another client took place. The content stored on the server appears in the actual cell as well as in the content label to the left of the text box (see Figure 11).

Figures

Figure 1. Client Server Architecture with Outlined Functional Components	2
Figure 2. Explanation of Undo and Revert Actions.....	3
Figure 3. XML Example of Sending an Empty Spreadsheet.....	5
Figure 4. UML formatted Diagram of the Initial Handshake	5
Figure 5. XML Example of Client Commands sent to the Server	6
Figure 6. XML Example of a Message sent from the Server to a Client with Spreadsheet changes.....	6
Figure 7. An Overview of the Interaction Between Clients and the Server	7
Figure 8. UML Documentation based on Multi-Client, Single-Server COCOPRO Spreadsheet Functionality	8
Figure 9. UML Documentation for Processing Connection and Logical Errors	9
Figure 10. Visual Representation of Actions taken While a Cell is Edited by Two Users Simultaneously.....	11
Figure 11. Example of Two Users Editing Cell A1	11

All figures were created in collaboration of the authors namely Dylan Habersetzer, Dylan Quach, Sabina Miani, Erickson Nguyen, and Tyler Wood.