

SDMCET Dharwad

Model **V**iew **C**ontroller

Madhusoodan M Pataki [2SD14CS054]

Content

1. What is MVC
2. What this project is [not]...
3. Selection of the Platform, Language ...
4. Class Diagrams for Model and Controller
5. How things work.
6. A simple Protocol
 - a. Different Scenarios in protocol.
 - b. PDU internals.
7. Demo Screenshots
8. Pseudo Code

1. What is MVC?

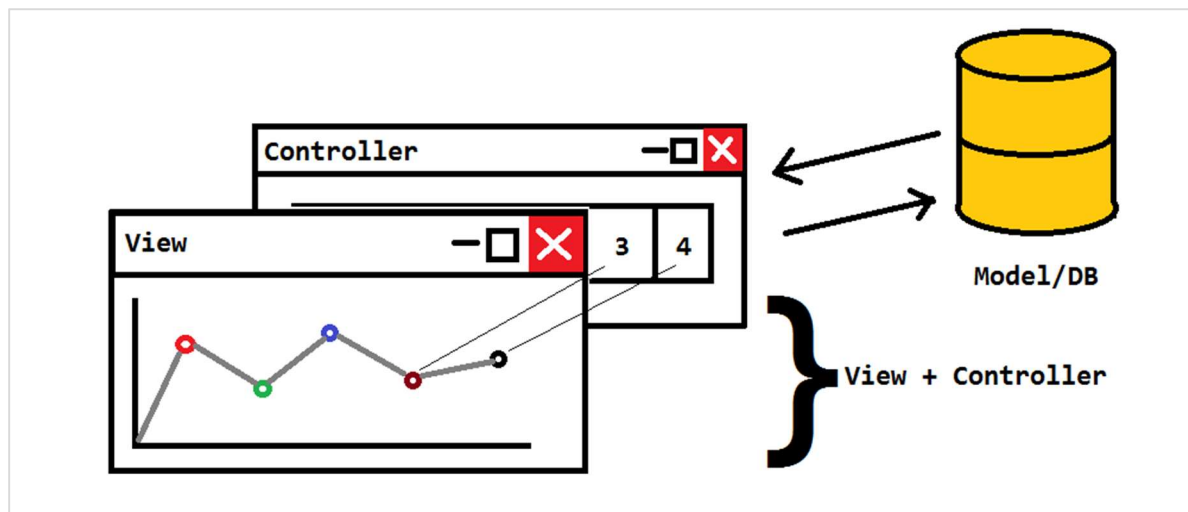


Figure 1: What is MVC?

Model

- Holds the data centrally.
- Holds communication links of all clients.
- Signals them if any changes occur.

View

- Holds a local copy of the data.
- Allows the actor to view some kind of representation of data.
- Updates itself whenever local data changes.

Controller

- Allows actor to modify local data¹.
- Notifies the Model about changes done.
- Upon confirmation of change notifies View about change².

1. Controller doesn't change the data directly.

2. No one can directly change local-data. Change must be notified from Model

3. What this project is [Not]?

This project is:

- Just a demo of MVC.
- Helps understanding how MVC works.
- Built just for max 10 clients and 16 data entries.
 - 1 Model : 10 Controllers/Views
 - 1 Model : 16 Data entries

This project is not:

- A software product to be sold.
- May fail sometimes due to some socket failures.
 - May not accept connection sometimes.
 - Abrupt termination of Model.
- No security measures are taken
 - Any-one can play with data.
 - Can cause buffer-overflow on Model.
 - Insert some absurd values in global data.
 - One person can connect many times and cause Denial-of-Service.
 - Can also make model to terminate.

3. Selection of the Platform, Language and Libraries...

Platform

- Windows Platform.

Language

- C, C++ Language.

APIs

- Win32 APIs for socket programming
- mGL-API (Widget Library developed by me for Windows for UI Elements).
- Many wrapper classes written by me around Winsock APIs

IDE

- Visual Studio 2015

Note: Executables are not in release version so needed to be compiled on the required machine only.

4. Class Diagrams for Model and Controller

Class diagrams help to visualize the entities in terms of their constituents and functionalities they provide. Here are the class diagrams of the Model and Controller.

1. Model

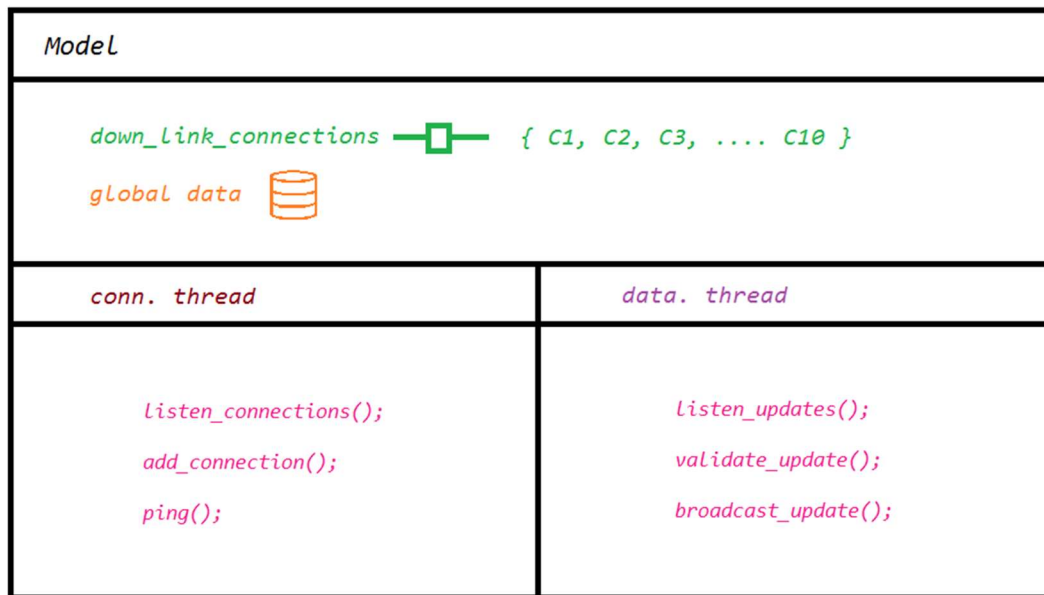


Figure 2: Class Diagram of the Model

2. Controller

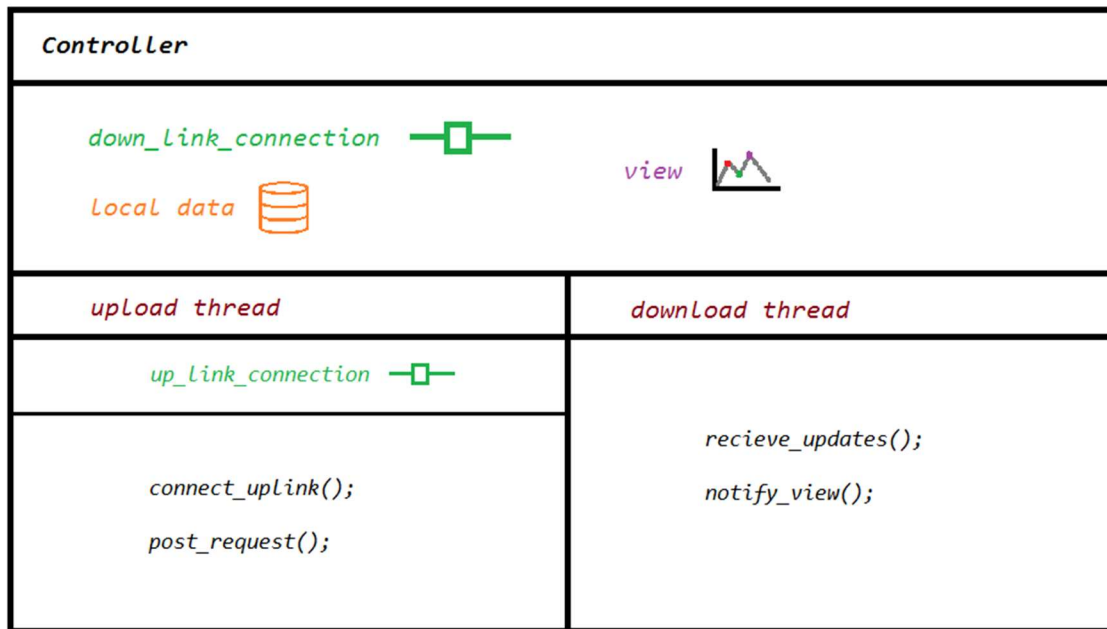


Figure 3: Class Diagram for Controller

5. How things work.

The model is a separate process running on any host device connected to a network. The Model consists of two threads conn thread (thread which receives new client connections) and data thread (thread which receives and broadcasts modifications to the data).

In order to connect to the Model the machine where client is executing needs to be first connected to the network where the Model is hanging. The client consists of two threads upload thread (used to post modifications) and a download thread (used to receive modifications). The Client starts executing by connecting to the Model by executing a connect protocol to connect its download thread to the Model. Whenever the client requires a modification to be done on the data it builds a Request and execute Addition/Modification protocol.

Whenever some client connects to the Model, Model accepts the connection and waits for the client to tell its name. After getting the name it returns a string to client telling whether it's registered for updates or not.

Whenever the Model gets updates from the client on its data thread it parses it checks whether the data is consistent, and if yes it sends it to all the clients which are connected to it.

The clients get the update on their download thread. It parses the string received and modifies its local data accordingly.

The protocol is explained in next section.

6. A simple protocol.

The communication between the components is established by a simple protocol which is explained below.

Ports involved

- | | |
|------------------|------|
| 1. UPLOAD_PORT | 5002 |
| 2. DOWNLOAD_PORT | 5003 |

UPLOAD_PORT:	Port used to send the updates
DOWNLAD_PORT:	Port used to get the updates from the server.

There are six scenarios identified in the protocol and they are as follows.

1. Successful connection to Model from client.
2. Unsuccessful connection from client to Model.
3. Successful Addition/Modification of the data from client.
4. Unsuccessful Addition/Modification of the data from the client.
5. Ping protocol from server.
6. Ping protocol from client.

The above cases are explained in further pages.

6.1, 6.2 [Un] Successful connection.

Explanation:

1. Model's conn. Thread waits for a new connection (It's always waiting for new client.)
2. Controller's download thread connects to it.
3. Now Model's conn. thread waits for name of the client to be sent.
4. Now Client sends his name. This may not be unique.
5. Server checks whether the list of client that it support is full. If it's full it sends an error statement back to the client, else it sends a null string and stores the connection.
6. The client checks the string returned by Model. If it's null it knows it's connected. It waits for the upcoming updates from Model, else it notifies the user about connection failure.
7. If some data is present in the Model it sends it back to new client using operation as 'M'any.

Below are sequence diagrams explaining the above protocol.

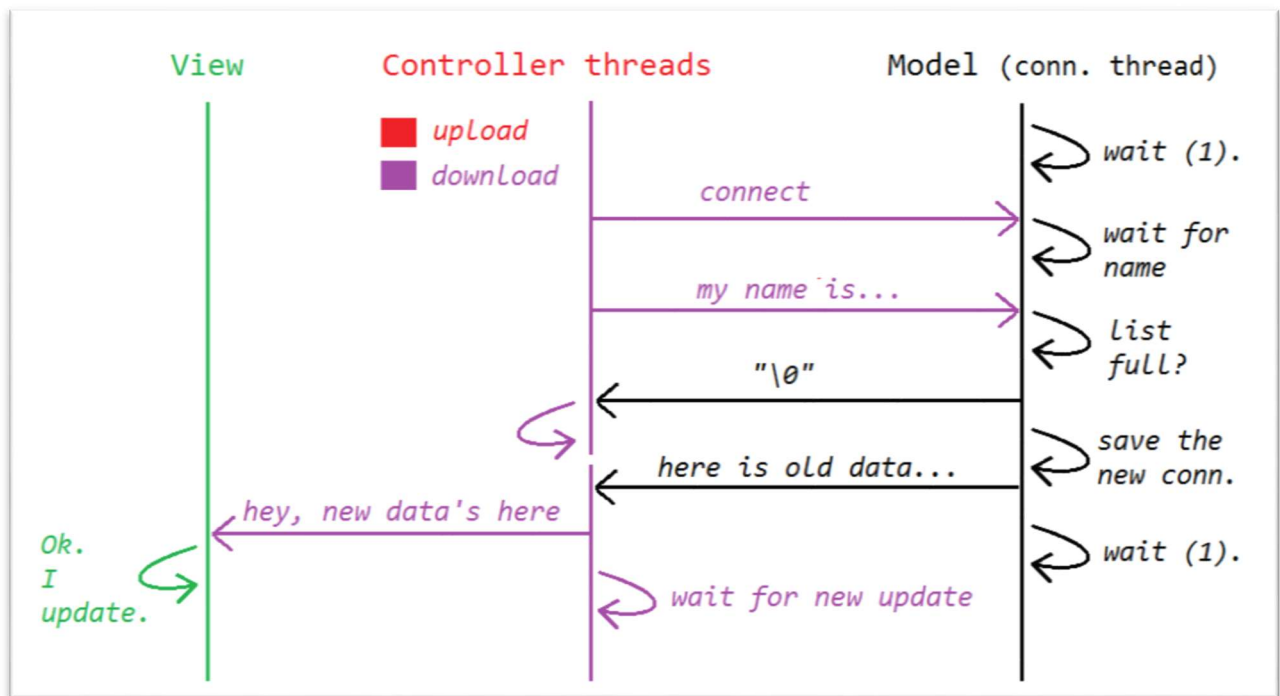


Figure 4: Sequence Diagram for successful connection

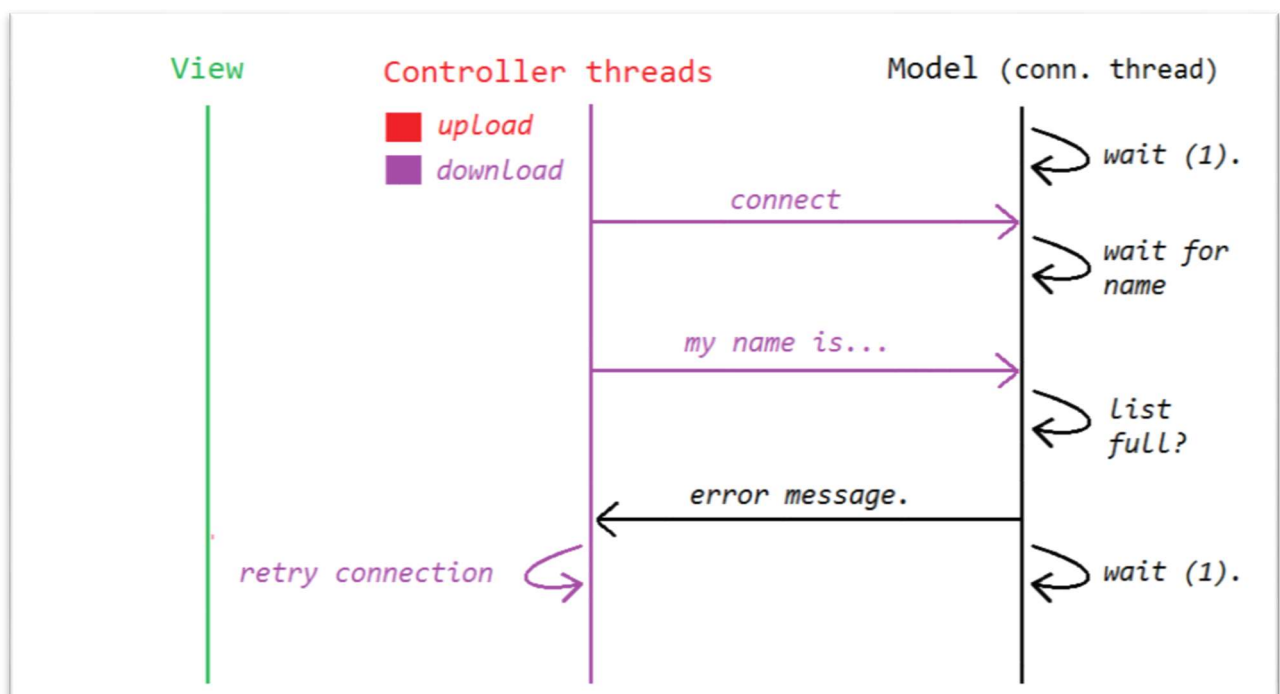


Figure 5. Sequence diagram for unsuccessful connection.

6.3, 6.4 [Un] Successful Addition/Modification of data.

Explanation:

1. The Model is always listening on the data thread for updates from clients.
2. The clients connects to Model's data thread and sends the PDU.
3. The Model validates the data (checks whether the values are greater than 0). If some invalid data is there it rejects the whole data, else does the update to the global data it is holding.
4. The Model sends the updates to all the clients via the connections that were stored while first connection (Client's download thread).
5. The Clients after getting the data also update their local data and notify the view.
6. View updates itself with the new data.
7. Model restarts to listen on data thread for new updates.

Below are sequence diagrams explaining the above protocol.

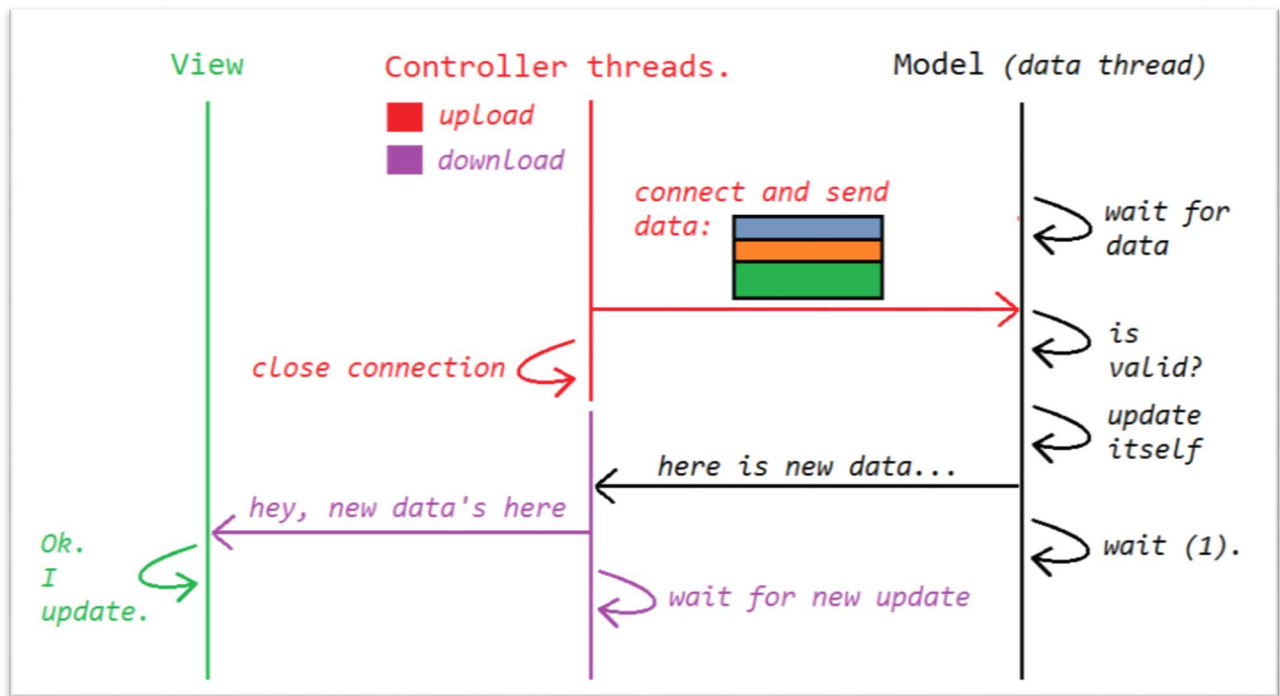


Figure 6: Successful Addition/Modification of Data

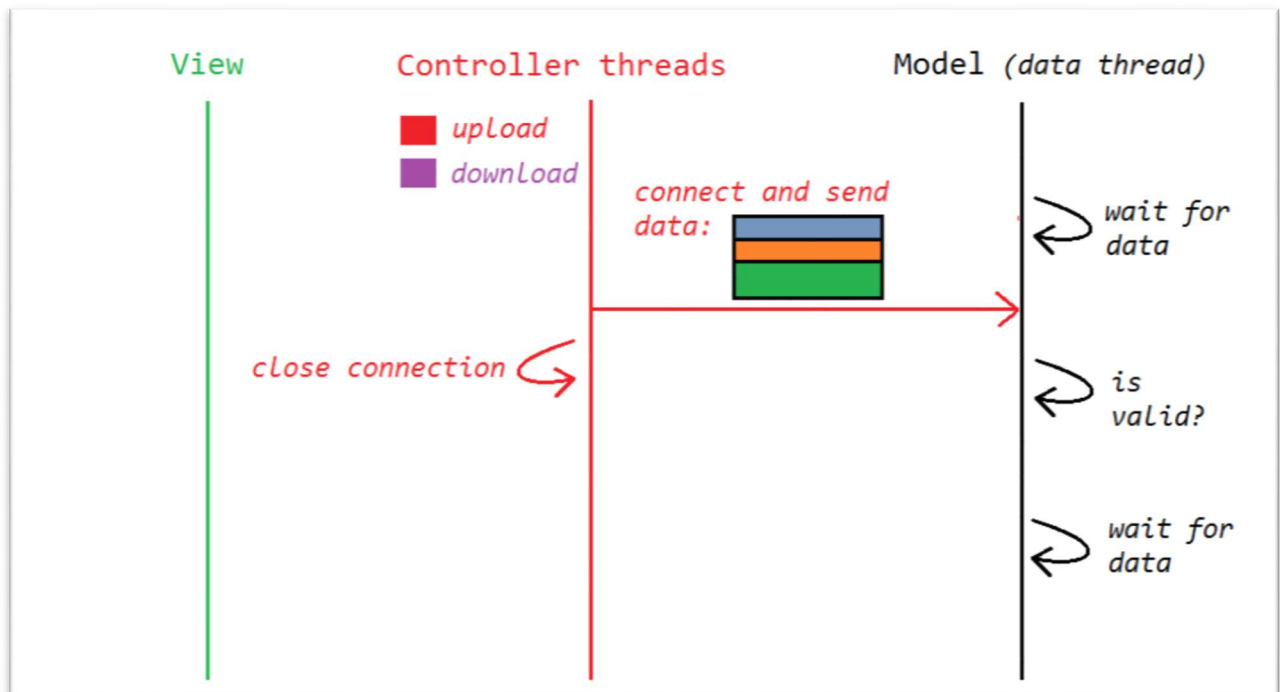


Figure 7: Unsuccessful Addition/Modification of data.

6.5 PDUs Transferred between components

The PDU (Protocol Data Unit) transferred between the client and the Model may be one of the following.

Name_Of_The_Client
Operation_Code
Number_Of_Operations
Key1 : Value1
Key2 : Value2
⋮
KeyN : ValueN

Figure 8. PDU for data transmission

ping

Figure 9. PDU for PING

The meaning of the fields from the above PDUs are

Figure 1:

1. Name_Of_The_Client: Each client connected to the server gives his name to model while connecting to it.
2. Operation_Code : This may be one of the following
 - a. 'A': Add data.
 - b. 'U': Update data.
 - c. 'M': Multiple Additions.
3. Number_Of_Operations: Number of operations in PDU of type specified above. This is usually 1 for 'A' and 'U' and >0 for 'M'.
4. KeyN : ValueN: Key value pairs of operations. The values in these are according to operations
 - a. For operation 'A'/'M': Key is label associated with the data (what data is all about). Value is the data itself.
 - b. For operation 'U': Key is the index of the data to be updated and Value is new data.

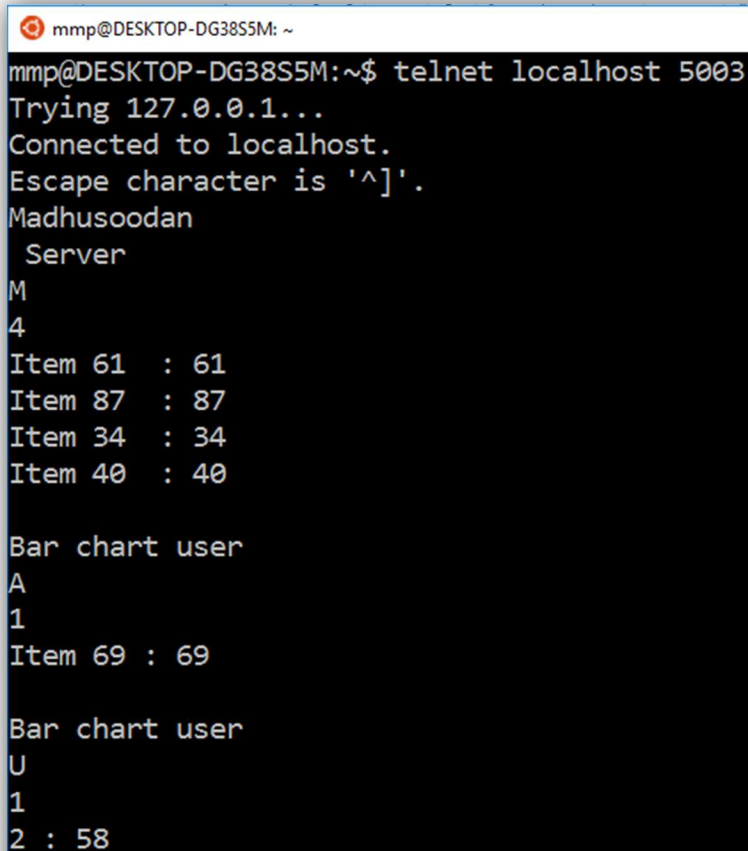
Figure 2:

1. ping: Literal string "ping" is sent for pinging the client or server.

Debugging the protocol

Open a telnet client (Open in Linux as it buffers the whole lines.) and type the IP of the host running Model application and port number as 5003. A connection will open to the model and nothing is echoed back. Now type a name for the client. The server will reply with an error message or null string. If it's null string then try adding data to Model from GUI based Clients. The PDUs are sent to telnet client also. Below is an Image of a debug session.

```
$: telnet localhost 5003
```



The screenshot shows a terminal window titled 'mmp@DESKTOP-DG38S5M: ~'. The user has entered the command 'telnet localhost 5003'. The terminal output shows the connection process: 'Trying 127.0.0.1...', 'Connected to localhost.', and 'Escape character is '^[''. The server then sends the name 'Madhusoodan' and the word 'Server'. The user enters 'M' and '4'. The server responds with a list of items: 'Item 61 : 61', 'Item 87 : 87', 'Item 34 : 34', and 'Item 40 : 40'. The user then enters 'Bar chart user' and 'A'. The server responds with '1' and 'Item 69 : 69'. The user enters 'Bar chart user' and 'U'. The server responds with '1' and '2 : 58'.

```
mmp@DESKTOP-DG38S5M: ~  
mmp@DESKTOP-DG38S5M:~$ telnet localhost 5003  
Trying 127.0.0.1...  
Connected to localhost.  
Escape character is '^['.  
Madhusoodan  
Server  
M  
4  
Item 61 : 61  
Item 87 : 87  
Item 34 : 34  
Item 40 : 40  
  
Bar chart user  
A  
1  
Item 69 : 69  
  
Bar chart user  
U  
1  
2 : 58
```

8. Demo Screenshots

Here are some screenshots of the Model and different Views.

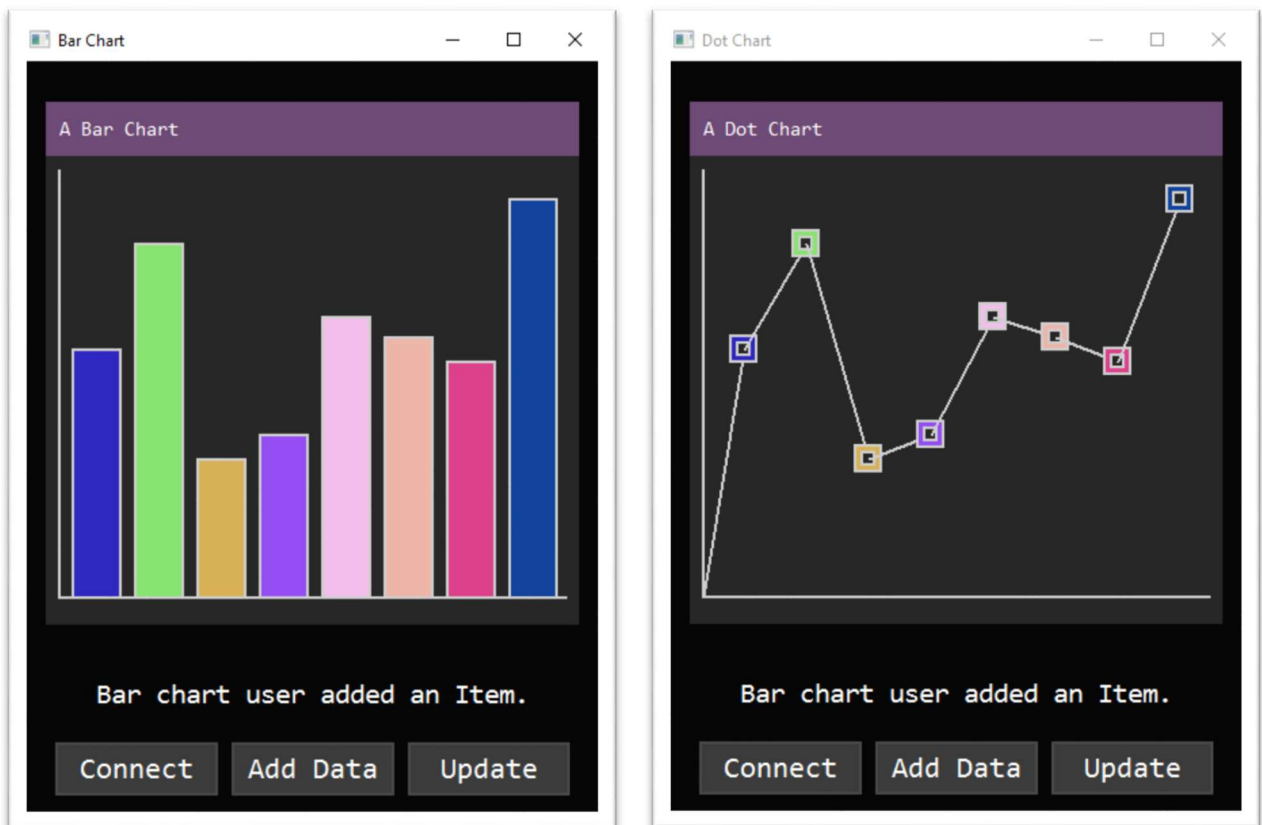
a. Charts populated with some data.



c. Model's log showing join of Client and updates.

```
C:\Users\Madhusoodan Pataki\Desktop\MVC Demos\Model.exe
Ready to accept clients
Ready to accept updates
Client [Bar chart user] got connected
Client [Dot chart user] got connected
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
```

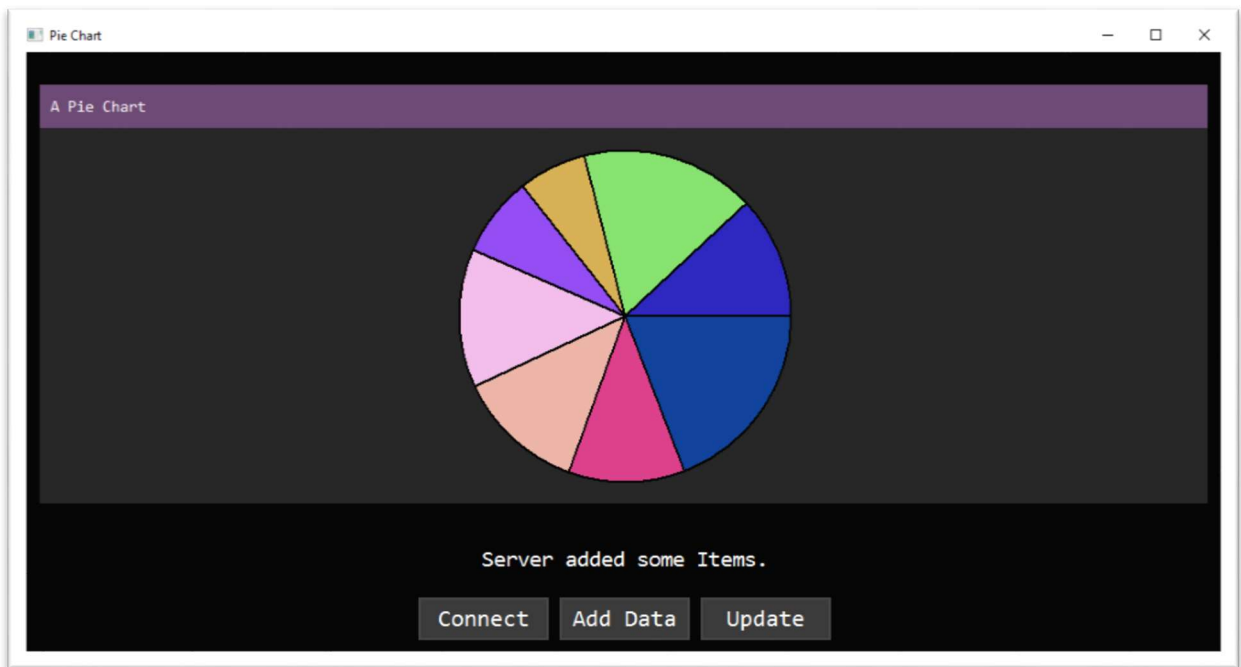

d. Now bar chart user add a data entity x ($20 < x < 100$) to Bar Chart



e. Model's snapshot when a new item got added

```
C:\Users\Madhusoodan Pataki\Desktop\MVC Demos\Model.exe
Ready to accept clients
Ready to accept updates
Client [Bar chart user] got connected
Client [Dot chart user] got connected
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
```

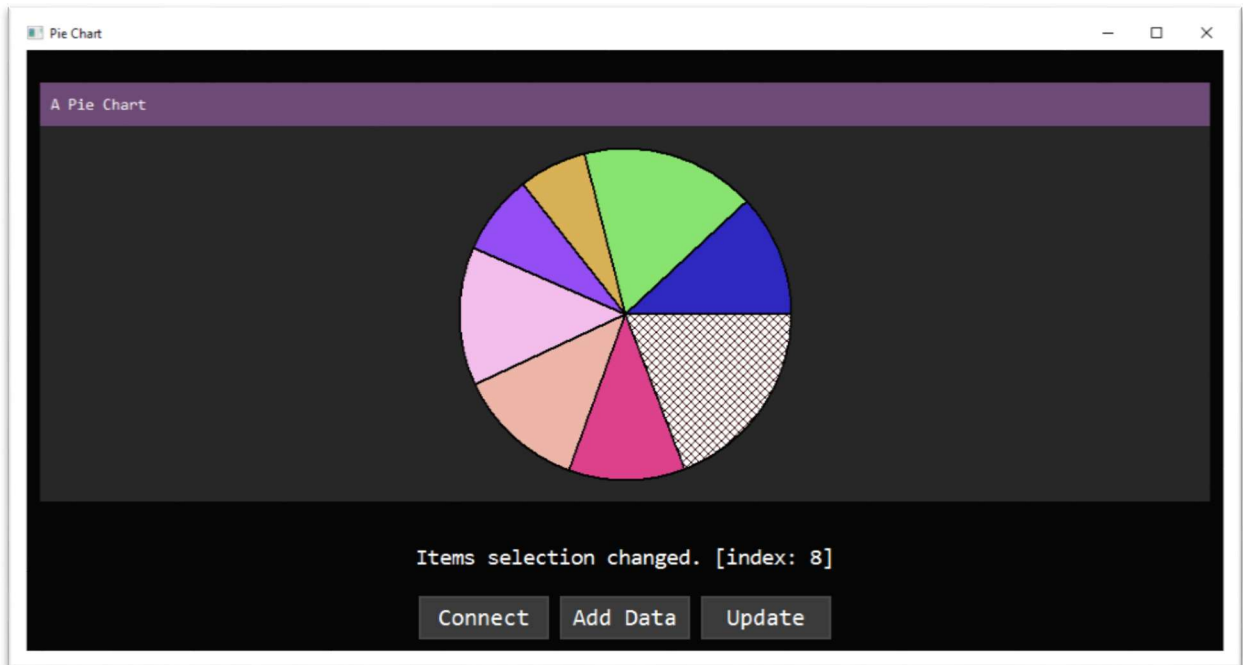
- f. A new client (Pie-Chart) joins in between and gets the data.



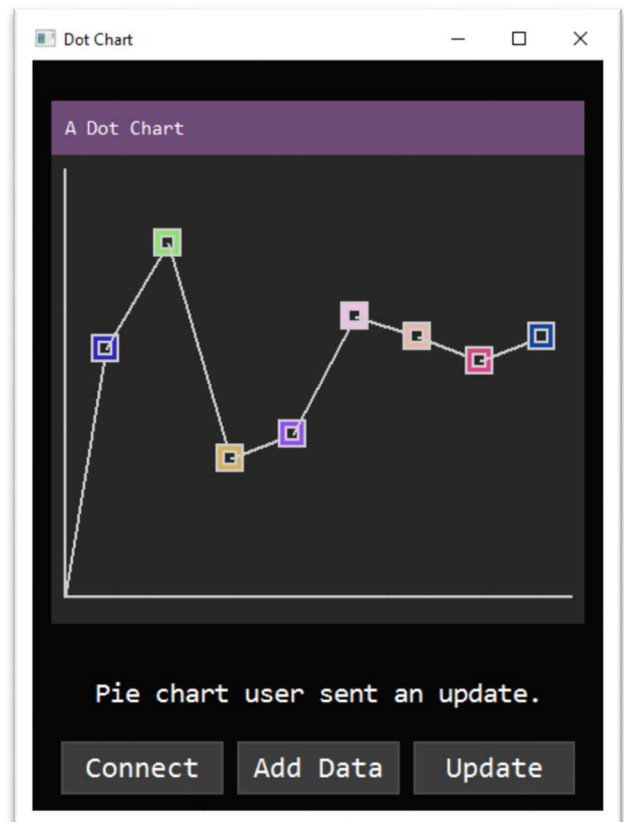
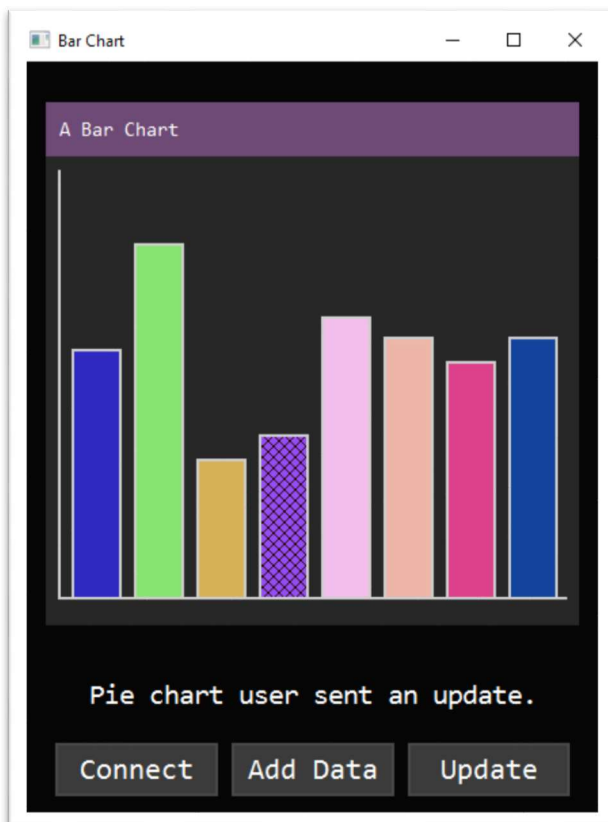
- g. Log of the Model after Pie-Chart user joins

```
C:\Users\Madhusoodan Patak\Desktop\MVC Demos\Model.exe
Ready to accept clients
Ready to accept updates
Client [Bar chart user] got connected
Client [Dot chart user] got connected
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Update recieved
Client [Pie chart user] got connected
Flushing off data to Pie chart user as Server
waiting to acquire lock on Data
locked Data
Done sending them
```

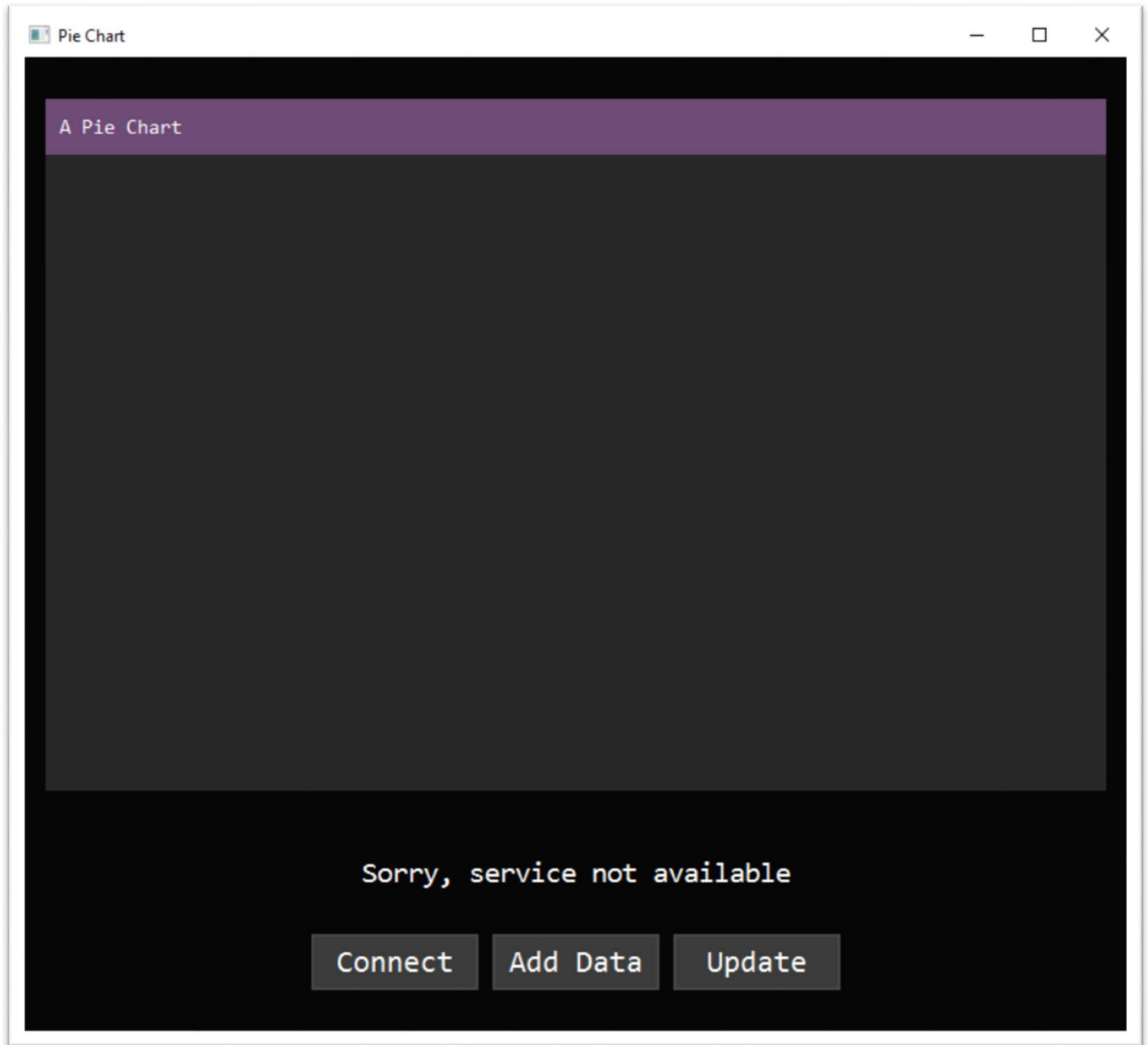
h. User selected a data region from Pie-Chart (hatched region)



i. User updates the selected region by clicking Update, which spreads to all.



- j. When eleventh client (>10) wants to connect it gets Service not available message.



9. Breaking down the system.

There are many ways to break into the system mentioned above. Take a look at sequence diagrams and you will come to know. Some of the methods that I have found are.

1. Connect to Model and don't tell your name. A server thread will wait for an infinite time (until it exits).
2. Connect to Model and tell a long name such that name buffer overflows.
3. Send data frequently to make server busy all the time.
4. Even if service is not available send modifications. Since there is no server side authentication this can be done.

10. Source Code

Source code and more can be found at

11. Outcomes

By doing this project, I

- Improved my design skills.
- Improved coding style.
- Earned more knowledge about MVC.
- Learnt why sequence diagrams are important.
- Appreciated CASE tools which make development cycle
- Thank Berkley for his sockets.