CSCI 5448 Spring 2016

PACKET SNIFFER

Apoorva Bapat
Nehal Kamat
Sunil Murthy

Click Here for Project Demo

Use Cases

Capture Packets

- Allows the user to be displayed the packets being transmitted in real time
- Actors: All users
- Pre-conditions: Users have opened GUI
- Post-conditions: Users should have basic knowledge of packet formats and should be able to read them
- Frequency of Use: High (main function of packet sniffer)
- Flow of Events:

  $\boxed{\text{Open application}} \rightarrow \boxed{\text{Click “Start”}} \rightarrow \boxed{\text{Select Network Interface}} \rightarrow \boxed{\text{Press Okay}} \rightarrow \boxed{\text{Transmitted packet details are displayed on the UI}}$

Filter Packets

- Enables users to view information of packets of their preference
- Actors: All users
- Pre-conditions: Users should start the application and select the type of packets to filter
- Post-conditions: Users should be displayed only those type of packets that have been filtered out by the user
- Frequency of Use: High
- Flow of Events:

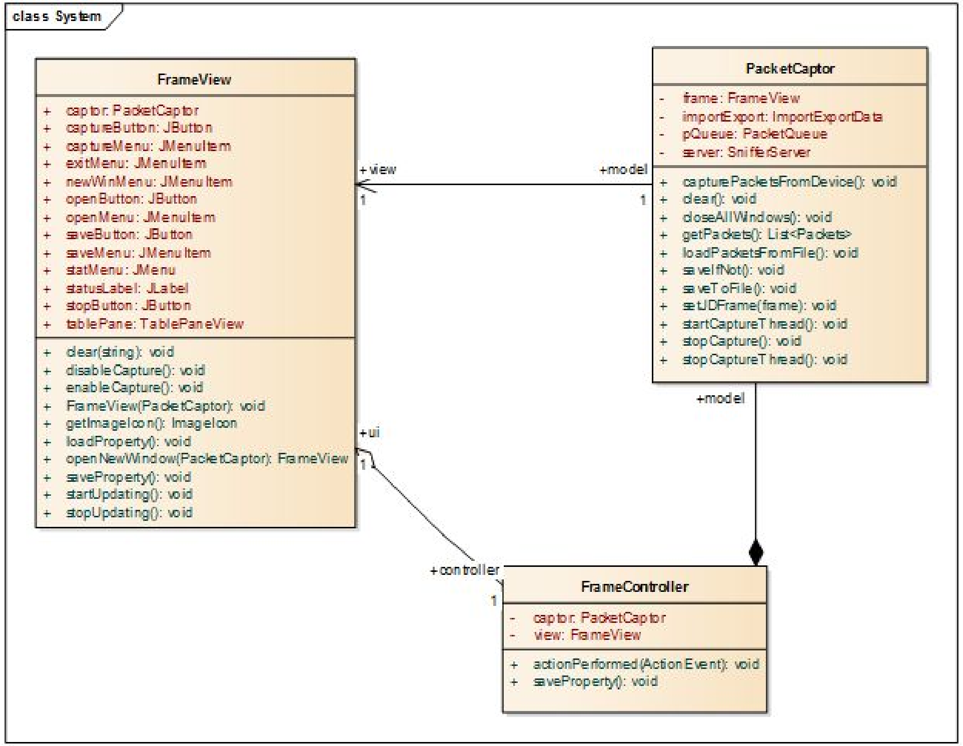
  Start application -> Click Monitor button -> Enter filter type -> Click OK -> System displays only filtered packet information

Display Network Statistics

- Enables user to view real time statistics of the information being transmitted along the network
- Pre-conditions: Users should start the applications and start monitoring packets
- Post-conditions: Users should be displayed real-time statistics of all transmitted packets such as number of a particular type of packet, origin and destination
- Frequency of Use: High
- Flow of Events: $\boxed{\text{Start application}}$ -> $\boxed{\text{Start monitoring packets}}$ -> $\boxed{\text{Click on “Statistics” in the menu bar}}$ -> $\boxed{\text{select Cumulative}}$ -> $\boxed{\text{Select Overall Stats}}$

Design Patterns

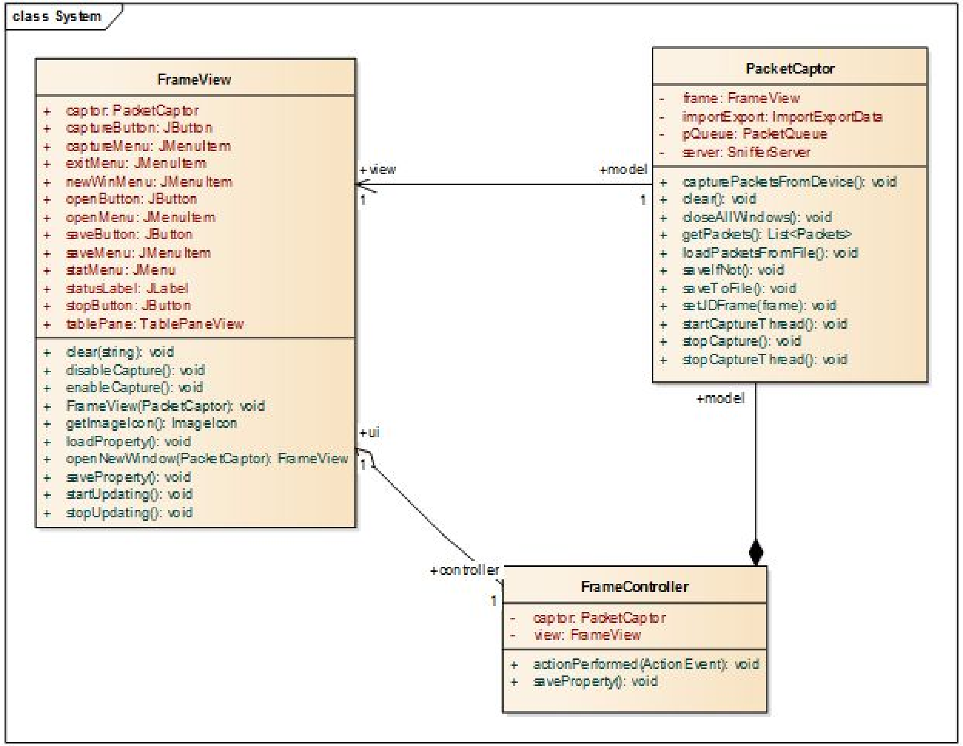
Model View Controller



- The PacketCaptor class, FrameView class and FrameController class form a model-view-controller architecture.

- The PacketCaptor class is the model which contains the basic data obtained from the calling of function and has all the necessary functions that are required to be called according to a user's requirements.

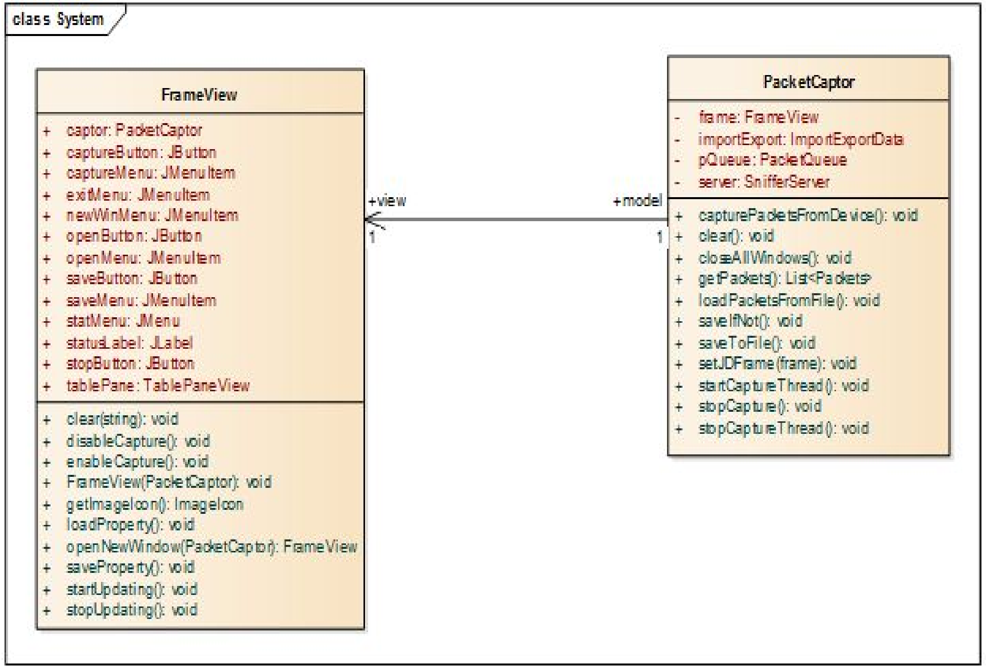
- The FrameView class serves as the View component that helps to render the window or frame of the window of the graphical user interface that the user will interact with (interaction such as inputting text, selecting options, etc.)



- The FrameController class serves as the controller component of the MVC, that helps to bind the data flow between the view and the model and to help keep the implementation of the 2 classes independent of each other.

- The FrameController class relays a user action or input in the view to the model and the model calls a function on that action. The result is replicated in the view and any modifications made to the view are replicated in the model, through the controller class.

Observer Pattern



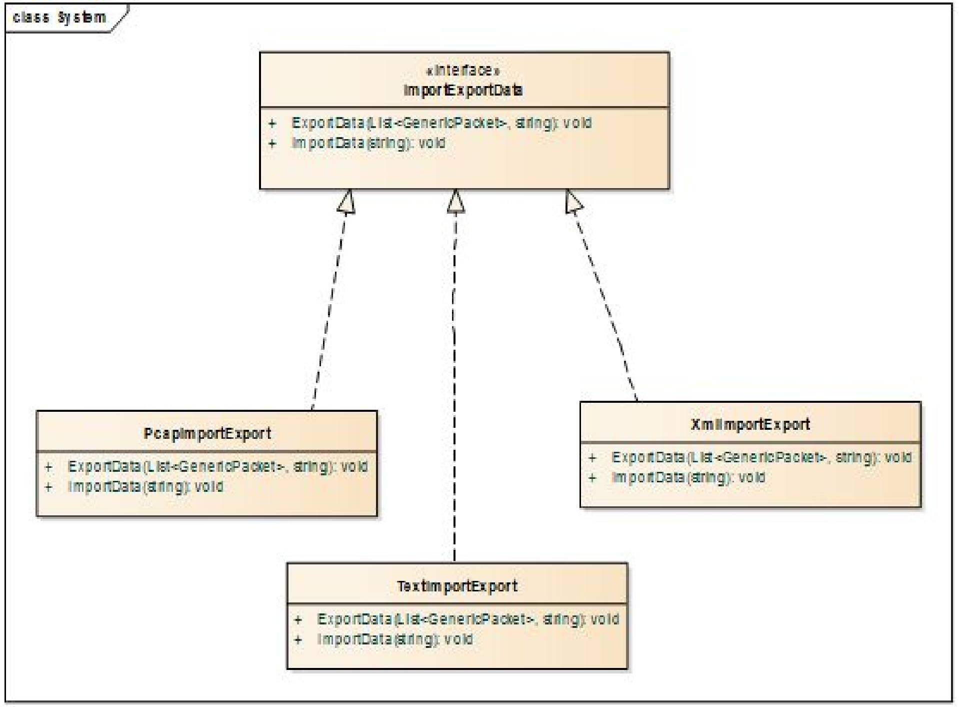
- The Observer pattern is basically a pattern that signifies one class observing the behavior of another and reacting to some stimuli that the observed class has to offer.

- The PacketCaptor class observes the FrameView class for any action that the user performs (interaction) with the view and takes corresponding action.

- The FrameView class observes the value of the variable or return value of a function in the PacketCaptor class corresponding to the user action and takes on a value according to what the PacketCaptor has for that entity.

- Basically, both classes monitor each other and keep their connected data in sync (not through direct connection but with the help of a connector)

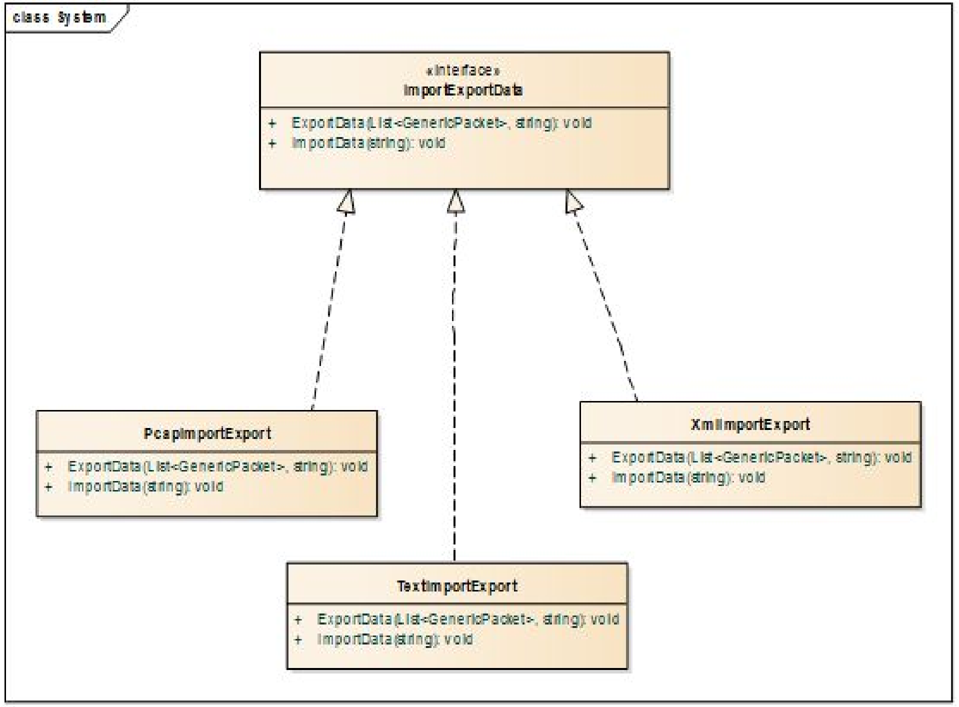
Strategy Pattern


class System
«interface»
ImportExportData
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void
PcapImportExport
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void
XmlImportExport
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void
TextImportExport
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void

- User gets to save file with any of the extensions shown in the figure above

- System resolves the input file type when the user loads any log file stored with one of the above extensions

- The ImportExportData class helps to save information of the packets in the session to a log in the following 3 formats: Pcap, txt or XML.


class System
«interface»
ImportExportData
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void
PcapImportExport
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void
XmlImportExport
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void
TextImportExport
+ ExportData(List<GenericPacket>, string): void
+ ImportData(string): void

- The strategy pattern comes into play when loading a log of packet information having any of the 3 given extension choices into the the program.

- The strategy pattern helps the function to resolve the extension properly and load the data in human-readable form.

Going Ahead

Future Work

- Implement Client Server
- More packet types
- More options to analyze packet information
- Adding option to mark packets

THE END