netObservator

Opening a Device for pcap

```
#include “pcap.h“


pcap_if_t *alldevs
char error[PCAP_ERRBUF_SIZE]; //=256
if (pcap_findalldevs(&alldevs, error) == -1)
    setErrorMessage(error);


pcap_if_t *device=alldevs;
for (int i = 0; i < selectedDeviceId; i++)
     Device = device->next;
```

Start Main Loop

```
#include “pcap.h“

pcap_t *handle = pcap_open_live(devs.getDevice(devId)->name,
                                BUFSIZ, // = 512 from pcap.h
                                0,
                                1000,
                                errorBuffer);


if(handle == NULL)
    // handle error
else {
    if (setFilter(handle))
        executeMainLoop(handle);
    pcap_close(handle);
}
```

pcap_open_live

```
pcap_t *pcap_open_live(char *device, //sniff on
                                       int snaplen, //max number sniffed bytes
                          /*BOOL*/ int promisc, //all packets in the network
                                       int to_ms, //timeout
                                       char *ebuf) //for error msg
```

Return Value: pcap_t* Handle to Session

Filtering expression

In general one does not want to sniff all of the traffic.

Thus compile a filter and set it:

```
int pcap_compile(pcap_t *handle, // handle to a device
                 struct bpf_program *fp, //compiled program
                 const char *str, //source of the filtering program
                 int optimize, // whether optimization is used
                 bpf_u_int32 netmask); // netmask of network

int pcap_setfilter(pcap_t *handle, struct bpf_program *fp)
```

Both functions have return Value: 0 success, -1 failure

Examples of Filtering Expressions

```
port 80 //only traffic with port 80

tcp //only traffic with protocol tcp

not dst host 1.2.3.4 //only traffic without
                     //destination host 1.2.3.4

tcp or udp //only traffic with protocol tcp or udp

tcp and host 1.2.3.4
```

Running the Main Loop

```
int result;

do {
      result = pcap_next_ex(handle, &header, &packetData);
      // readCode 1 = packet read, 0 = timout, -1 = error
      if (result > 0) {
          // extract infos from the packet
      }
      else if (result < 0) {
          // handle error
      }
} while (result >= 0);
```

The Packet Header

pcap_pkthdr **header:

- struct timeval timestamp (long tv_sec, long tv_usec) seconds and microseconds since Epoch (01.01.1970 00:00)
- bpf_u_int32 caplen (length of captured packet)
- bpf_u_int32 len (lenght of packet)

(long = „signed“ 32 bit integer)

(bpf_u_int32 = Berkeley Packet Filter unsigned integer 32 bit)

u_char **packetData

u_char is a byte

- Ranging from 0 till 255 ($2^8 = 256$)

packetData consist of

- IpHeader
- UdpHeader
- Payload

u_short = unsigned 16bit integer, may consist of
                  2 u_chars

Example of PacketData in ASCII

```
L....E.....r..E.......6.U..:........#...j..]$P..X........L...H..W....)).:^.o..f.p'Y.E...7t.\......+..
..............#.........h2....................0...0.........H7.T....0...*.H........0I1.0...U....US1.0...U....Google
Inc1%0#..U....Google Internet Authority
G20...160817185643Z..161109182900Z0f1.0...U....US1.0...U....California1.0...U....Mountai
n View1.0...U....Google
Inc1.0...U....*.google.com0Y0...*.H.=....*.H.=....B.....p.G.V...6..i].&.K9@.f....^..".`.6LL..)Bh....f
v.+o.4.^.$..q`......0...0...U.%..0...+.........
+.......0..i..U.....`0..\..*.google.com..*.android.com..*.appengine.google.com..*.cloud.google.
com..*.google-
analytics.com..*.google.ca..*.google.cl..*.google.co.in..*.google.co.jp..*.google.co.uk..*.goo
gle.com.ar..*.google.com.au..*.google.com.br..*.google.com.co..*.google.com.mx..*.google.
com.tr..*.google.com.vn..*.google.de..*.google.es..*.google.fr..*.google.hu..*.google.it..*.goo
gle.nl..*.google.pl..*.google.pt..*.googleadapis.com..*.googleapis.cn..*.googlecommerce.co
m..*.googlevideo.com..*.gstatic.cn..*.gstatic.com..*.gvt1.com..*.gvt2.com..*.metric.gstatic.co
m..*.urchin.com..*.url.google.com..*.youtube-
nocookie.com..*.youtube.com..*.youtubeeducation.com..*.ytimg.com..android.clients.googl
e.com..android.com..g.co..goo.gl..google-
analytics.com..google.com..googlecommerce.com..policy.mta-
sts.google.com..urchin.com..www.goo.gl..youtu.be..youtube.com..youtubeeducation.com0..
.U........0h..+........\0Z0+..+.....0...http://pki.google.com/GIAG2.crt0+..
```

IpHeader

- u_char ProtocolVersionAndInternetHeaderLength; //4bit + 4bit field
- u_char typeOfService; //various Def's. during the years: current precedense
  //Explicit Congestion Notification
- u_short totalLength; // of entire packet header & data
- u_short identification; // for fragments of an IP packet
- u_short flagsAndFragmentOffset; // infos about position as datagram fragment
- u_char timeToLive; // hop count, each router passed decrements by #secs or 1
- u_char protocol; // for example TCP or UDP
- u_short checksum; // error checking of ipHeader
- ipAddress sourceAddress;
- ipAddress destinationAddress;
- u_int optionAndPadding; // optional, for example about routing and security

UdpHeader

- u_short sourcePort;
- u_short destinationPort;
- u_short length; // of UdpHeader + Data
- u_short checksum; // for UdpHeader & Data

Design Pattern

What is a Pattern?

- A Design Pattern is a general reusable solution to a commonly occurring problem within a given context.

What does that mean?

- Template to solve a problem
- Can be applied in many comparable Situations
- Tested, Proven, Best Practices

Anti Pattern

- Spaghetti Code
- Magic Numbers / Magic Strings (instead of constants with hopefully meaningful names)
- God Object (allmighty and omniscient)
- Golden Hammer (Solution for everything)
- Design Pattern itself (human as compiler, from critics as a sign for a lack of abstraction)

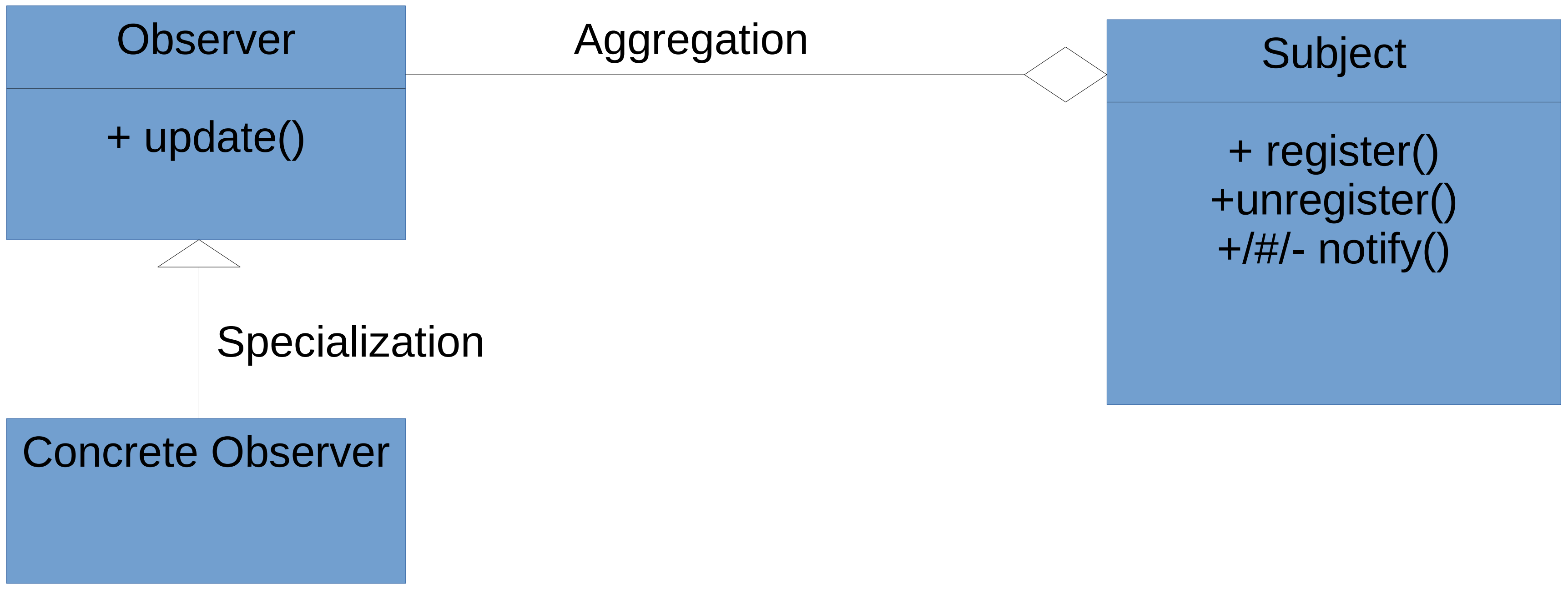
Some Pattern History

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (Gang of Four) published „Design Pattern“ in with 23 classical Design Pattern in 1994

„Program to an interface, not to an implementation“

„Composition over Inheritance“

The Observer Pattern



From Subject.cpp

```
Subject::register(const Observer *obs) {
     observers.push_back(obs)
}

Subject::unregister(const Observer *obs) {
     if (std::find(observers.begin(),observers.end(),observer) !=
observers.end())
          observers.erase(pos);
}

Subject::notifyObservers() {
     SubjectState state = getSubjectState();
     for (Observer *obs: observers)
          obs->update(state);
}
```

Pros and Cons

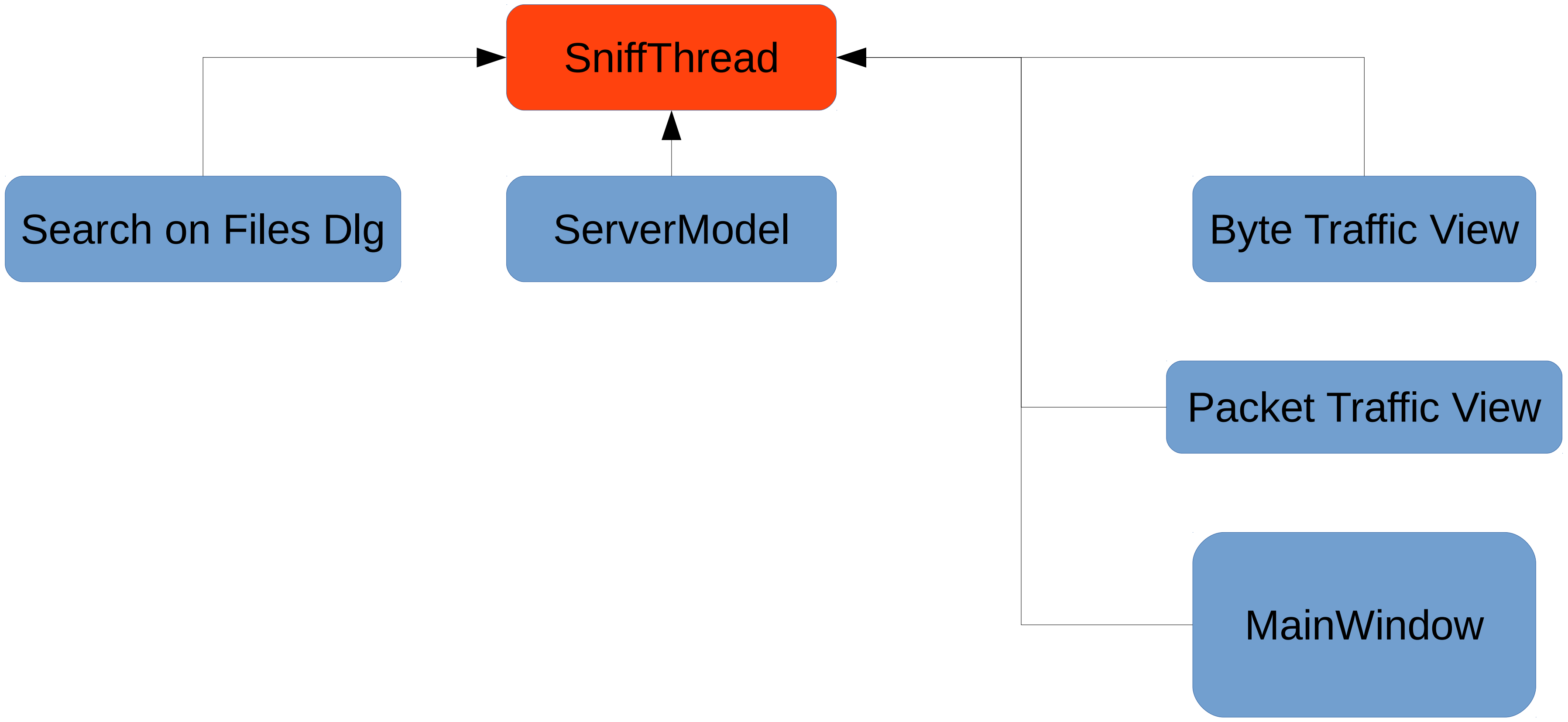
Pro:

- Loose Coupling
- No modification needed to (un)register
- Any time one can (un)register

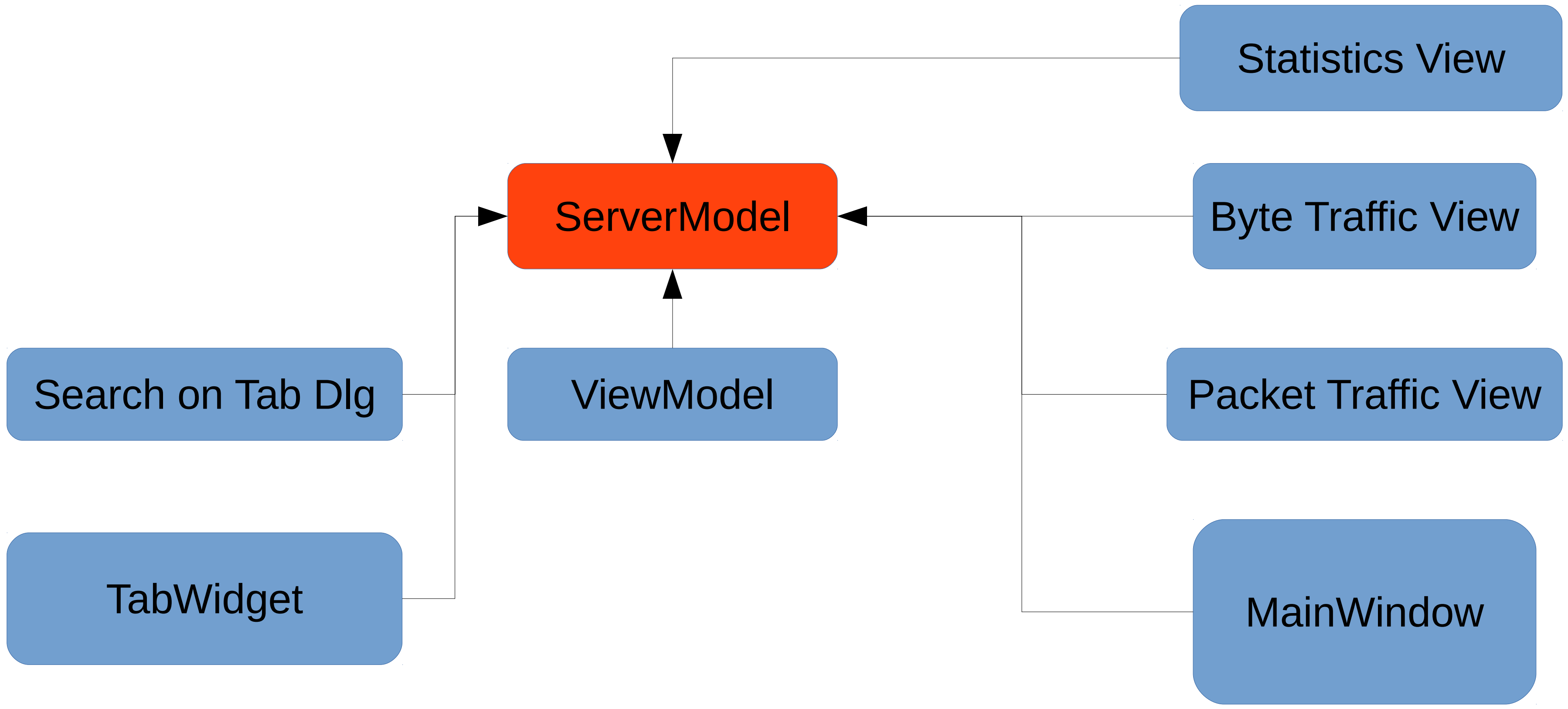
Contra:

- In the case of many subjects it can be very complicated (Cascades, Circles of Notifications, undetermined Behaviour), if one is not careful
- May cause Performance Problems

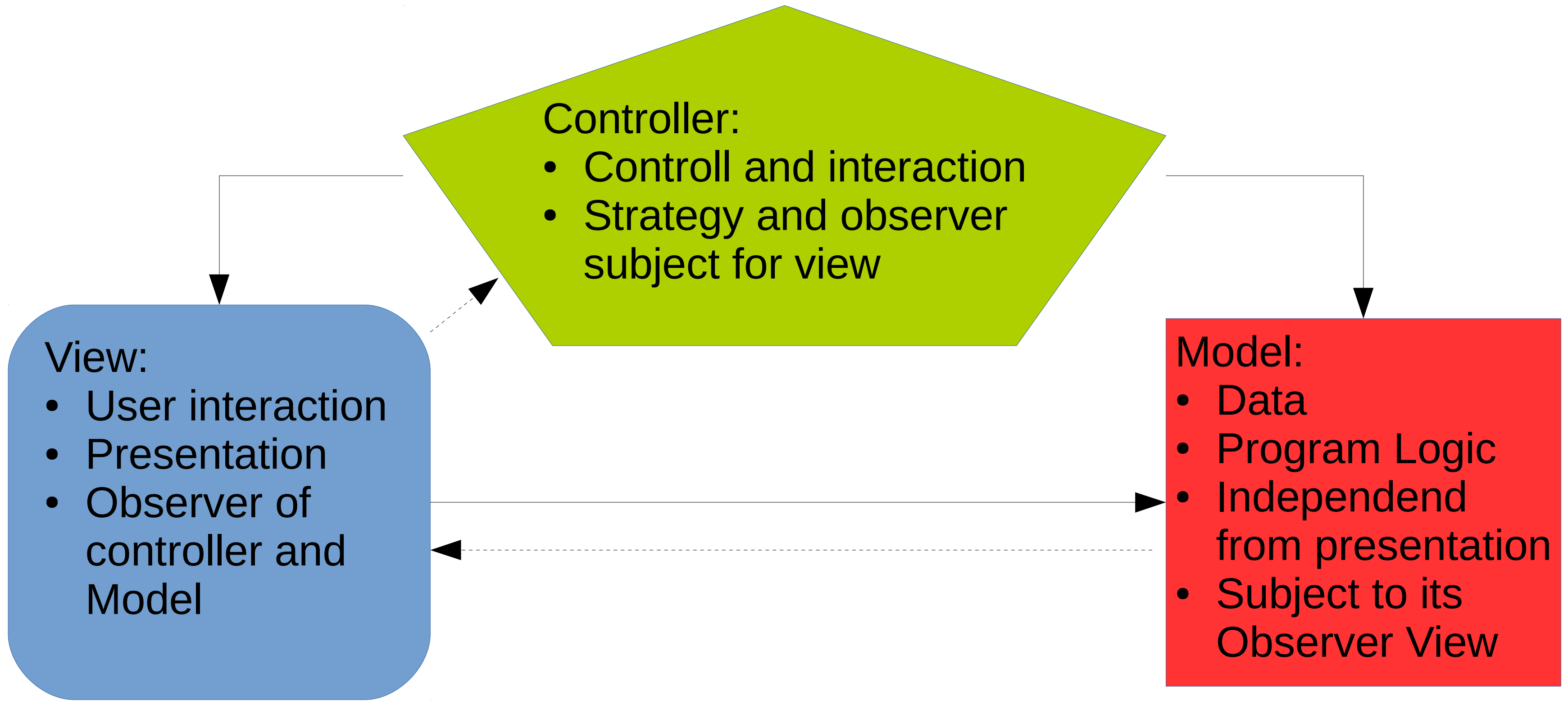
Observers in netObservator I



Observers in netObservator II



Model View Controller


Controller:
• Controll and interaction
• Strategy and observer subject for view
View:
• User interaction
• Presentation
• Observer of controller and Model
Model:
• Data
• Program Logic
• Independend from presentation
• Subject to its Observer View

(Dis)Advantages of the MVC

- Code is separated into 3 Modules. Each with a well-defined concern
- Presentation and Logics are loosely coupled
- Developers may specialize to work on one component
- High Complexity
- May cause excessive updates on the view observers

On the Controller

In general commands from the MainWindow use:

```
void Controller::getCommand(Command command) {
   if ( /* conditions to block the command */ )
       SetErrorMessage(...);
   else
       executeCommand(command);
}
```

For

```
struct Command {
   CommandCode code;  //enum class
   QStringList arguments;
};
```

On the View

Breakin the Rulez!!!

Each Tab stores its own

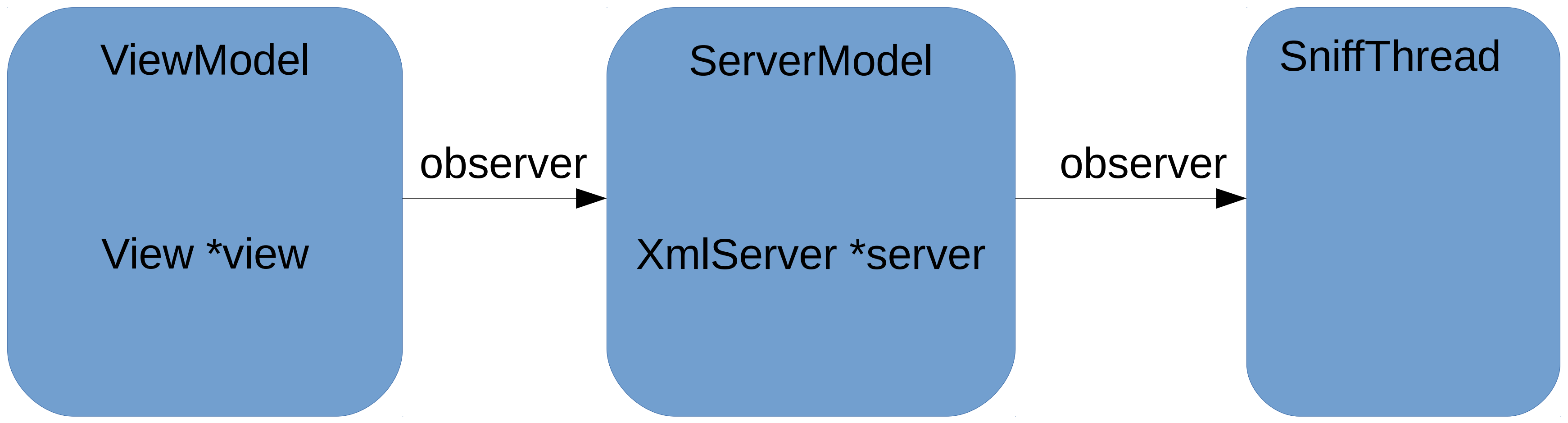
- XmlServer *server;
- View *view;

*server and *view are inserted in the Model, if the current tab changes:

```
void Controller::setModelView(modelView *mv) {
   model->view.set(&mv->view);
   … //some lines, which may hopefully be removed soon
   model->server.set(&mv->model);
   model->sniff.setPacketInfoPresenter(&mv->view.tablePacketInfo);
}
```

On the Model

General Architecture:



Why Server in Tab / Why not?

Why:

- TabWidget manages the content of the tabs perfectly. Thus I do not have to do. (Reducing Complexity)
- Quiet simple architecture. The controller only needs to put the components into the right places.

Why not:

- Very blunt procedure.
- Each time the Model „knows“ only the content of one tab.