Lab 5 – Design Discovery Script

The document describes what your instructor would do, if he was solving the lab's problem. It is written in a non-formal way to mimic code exploration and design discovery.

1. I will first run the GUI, and find out which widget should trigger find request.

2. Humm, let's find a widget which performs similar function. How about file downloading functionality?

3. Let's explore the file download functionality.

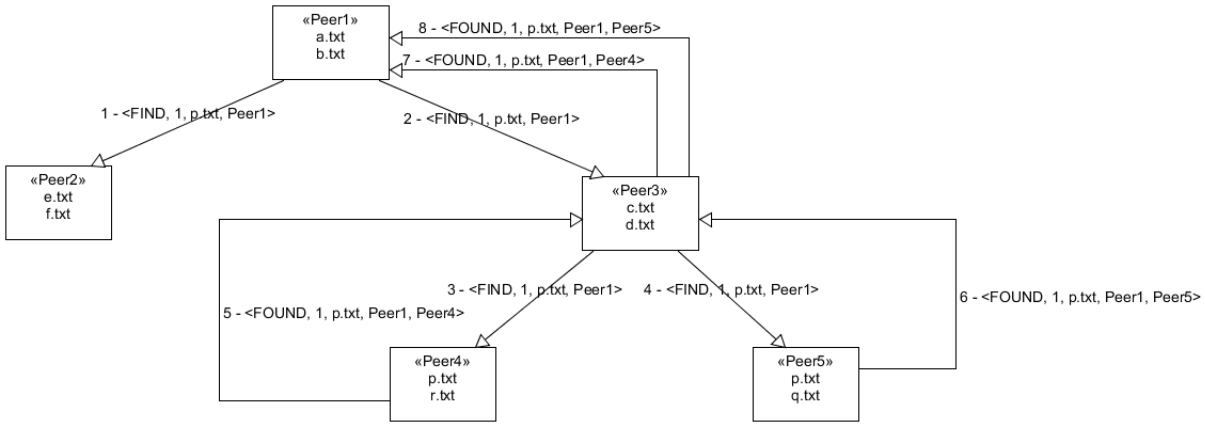
i. Let's go to P2PGUI, find download button, find all of the references of the button in the project.
ii. Let's look at the steps used (Getting host, file, calling get request on Mediator).
iii. Calling Get Request on Mediator, let's see how that is done …
    a. Get stream monitor, humm, this sounds like a class that monitors the socket stream. I will explore it a little later.
    b. Start a new sequence number for the request
    c. Create a get request packet
    d. Log the request
    e. Send the packet
iv. How does the receiving on the other side works?
    a. I think it's time to explore the StreamMonitor class now. StreamMonitor monitors each peer in a separate thread, it seems. Who creates it?
    b. ConnectionMonitor sounds like it listens for incoming connection, may be this one creates StreamMonitor, let's check it out.
    c. I see that if requestAttachOK is called via P2PMediator, then it creates StreamMonitor for monitoring incoming request from that socket.
v. Ok, StreamMonitor receives Get request now, what's next?
vi. packet.fromStream() in the run method of StreamMonitor parses that request. Let's take a look.
vii. The fromStream method first parses status and headers and populates the protocol, command, and the headers map for the packet.
viii. It then request the protocol object to provide it a hander to handle the request. Humm, where do we configure the handler? Perhaps before the GUI starts, let's take a look at the P2PApp class, which sounds like the entry point of this application.
ix. I see that that we configure protocol with handlers for different commands like GET, PUT, LIST, LISTING, etc. Interesting! So, everything can be attached to the framework!
x. Alright, so I know now the GetRequestHandler is called to handle the incoming get request. Let's explore further.
xi. The GET handler extracts necessary info from the packet and asks the mediator to process PUT request.
    a. The mediator gets the monitor for the peer in the requestPut method.
    b. The file object is created followed by setting up the packet for file transfer.
    c. Wait, we never read the bytes from the file? How does the transfer work?
    d. Let's explore packet.toStream(). Something must be happening there!

e. The stream builds the status line, header, and writes to the socket, wait, what!
f. Got it! Like request handler, to hook in some custom actions after sending a packet header, we can register a response handler for such post-action. Cool!
g. Let's get back to P2PApp and see what response handler is available for the PUT request.
h. There we go, we are registering PutResponseHandler for this task.
i. Let's explore the PutResponseHandler. Great, all it does is reads the file and dumps the data to the socket! That's how file gets sent!

xii. How does the file get received then?
xiii. There must be a PUT request handler, let's take a look.
xiv. Yes we have one. It just reads the bytes from stream and creates a new file to write that data. That is cool!
xv. It also calls fireDownloadComplete() on mediator. I am pretty sure this has to do something to notify the GUI about the download.
xvi. Indeed, when I explore the method, it just goes through all of the download listeners and tells them about the download. Perhaps, the P2PGUI has registered itself for listening for such events. Let's check P2PApp again.
xvii. Indeed, P2PGUI registers itself for various notifications to update the GUI based on the protocol events. Wow! Everything can indeed be attached to the framework! That is cool!

4. Now let's think about designing the FIND feature.

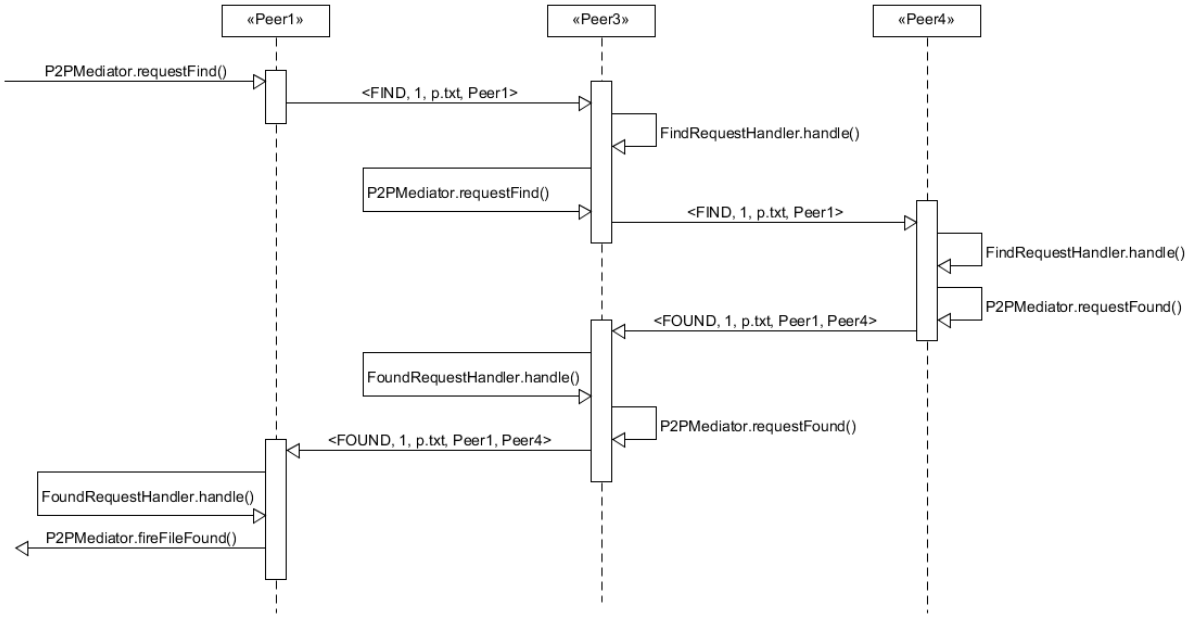
i. I need to create an action listener for the find button. Wait, I need to first decide on what protocol objects are needed. Here is what I am thinking, the exact sequence may not be true as the peers are running on different processes and threads, but this is how I can get the FIND feature to work: (Note to myself, Request: <COMMAND, SEQ_NO, FILE, ORIGIN-IP:PORT>, Reply: <COMMAND, SEQ_NO, FILE, ORIGIN-IP:PORT, DEST-IP:PORT>)



ii. I think I also need to maintain a stack of peers (can be a comma separated list of IP:Port) in the packet header to trace and remember who to reply back to. I can push the peer into the stack

when sending/forwarding the FIND request, and pop the top peer from the stack for sending the FOUND request. When the stack in the header is empty, I know that peer is the original peer that initiated the FIND request.

iii. Ok, let's translate the communication diagram to code now. Humm, how about the following sequence:



iv. I think I know what to do now! I am pumped! Yeah!