Homework
Report of software component development

Dragos Mihaita Iftimie

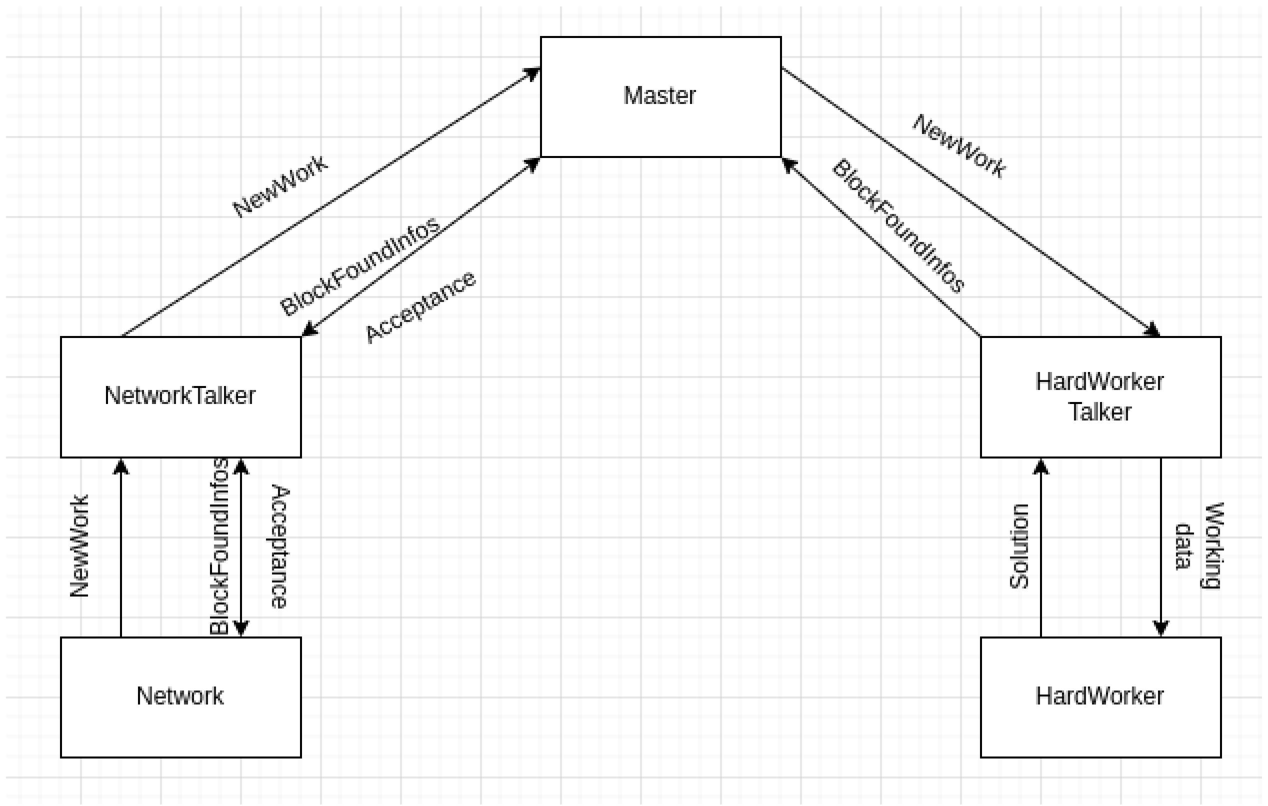
usage examples – how to use the component:

for the course:
Component based software engeneering
prof. Šarūnas Packevičius

A needed introduction:

My project is related to the blockchain world, mining to be precise.
In an extremely short way, we could say that a miner retrieves the needed informations for the next block from the blockchain network, manipulate those info and tries to find a solution to the problem (see proof of work blockchains for example).
If a solution is found, the next block that has been solved has to be propagated to the network.
An important aspect to be noticed is that in case someone else solved the block the miner was working on, it should stop working and should start looking for the solution of the "next next block".

Generic design of a miner:



The following design (pretty simple) has been made by having in mind to design an "as much as possible" generic miner.
It means that:
- the hard worker may be anything that can do computations (cpu,gpu,dedicated hardware…) and communicate with an HardWorkerTalker;
- the hard worker migth be located anywhere (not on the same machine of the master);
- there may be multiple hard workers, those workers may even be able to work on different problems (e.g. different cryptocurrencies);

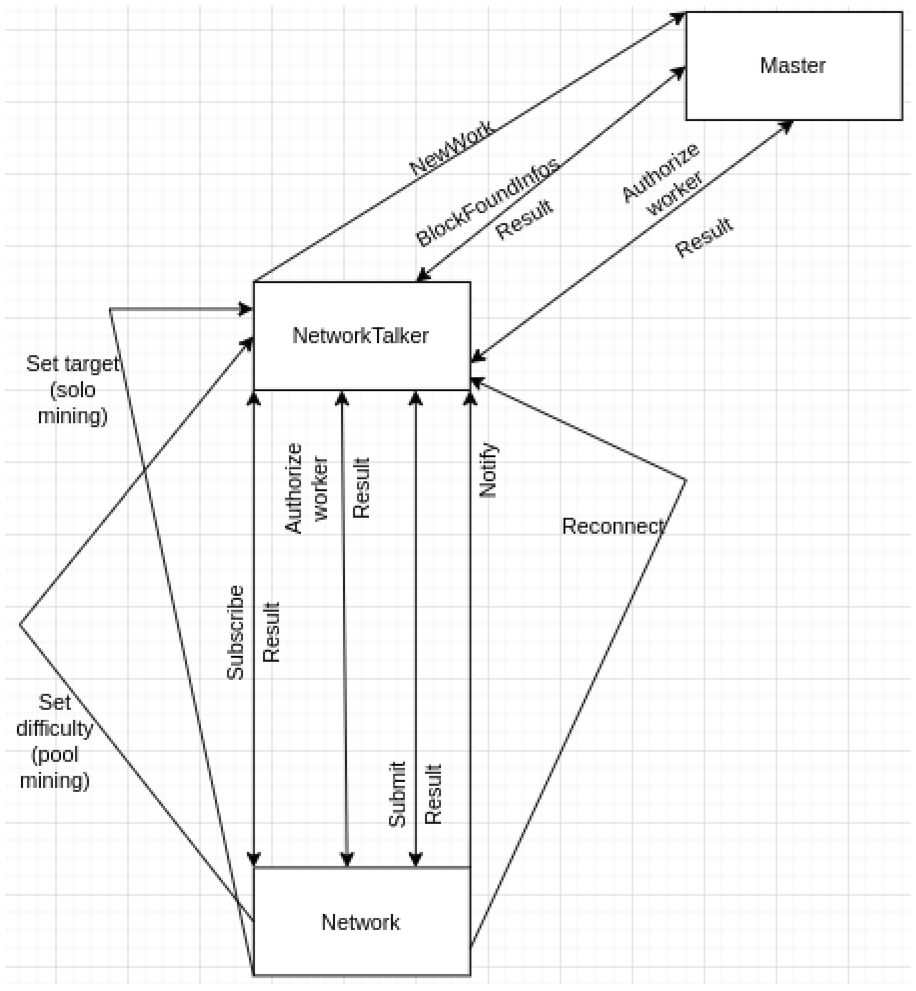
- the network from which we get the infos may be anything (a running blockchain node, a solo mining server, a pool mining server…) and may talk with us in any protocol (stratum v1, stratum v2, getblocktemplate, getwork…);
- the talker may be located anywhere (if you need that it means you are doing dirty stuff -_-, of course you may use a proxy, but in that design the NetworkTalker is the proxy itself, this is more efficient);
- the master just coordinates the other actors and keeps in a centralized way all the informations/statistics/logs about the process.

The component that I have developed is a NetworkTalker, more precisely it allows the master to interact with a Stratum v1 pool mining server.

Let's take a closer look:

Here some documentation about stratum v1
https://braiins.com/stratum-v1/docs
https://github.com/aeternity/protocol/blob/master/STRATUM.md

Here we can see all the possible interactions between the actors.



Basically the talker is a client (towards the network) and a server (towards the master).
Ok, let’s start now.

Component description:

a) provided functions (interface):

The stratum v1 talker provides the following functions:

- alert the master that a new work from the network arrived
- submit a solution received from the master
- ask the server to add a new worker
- alert the master about the results of his interactions with the network

The implementation is in python and the talker has to be launched as a new process from the master.
As communication channel between the master and the talker I’ve chosen to use multiprocessing queues. 4 multiprocessing queues has to be shared between the master and the talker:
1) solution queue, write only (from master perspective), put a solution here and the talker will retrieve it and send to the server;
2) add worker queue, write only, put a worker here and the talker will ask the server to add it;
3) new work queue, read only, periodically read from this queue to check if a new work has arrived;
4) results queue, read only, periodically read from this queue in order to check how previous requests to the server ended.

b) purpose:
I think I’ve already explained it in the “A needed introduction” part.

c) business problem that the component solves:
same.

d) intended use:
The component can be used if a stratum v1 talker that runs on the same system of the master is needed.

e) restrictions for component usage:
The component right now has not fully implemented stratum v1, just the basic methods to be able to start mining.

That’s why it cannot still be used for solo mining, the set target metod is not implemented yet.
Another important aspect is that the component uses sockets and the select() function; this means that the component interacts with the operating system in a deep way; I’ve only tested it on an UNIX like operating system, I don’t assure it will work on Windows.
Another thing to say is that the log management sucks.

f) important informations:

This is just the first version of the component, in the next updates you should expect:
- full implementation of stratum v1 (set target and reconnect are missing now);
- for efficiency, the data format of the communicatoins between the talker and the master may change;
- the communication channel may change, right now queues are used but… who knows, it may be done with a pipe or a socket.
- the reconnect method asks the talker to reconnect to the server, this may also mean to a different uri (the server will tell which uri to use), for security reasons (we don’t want our precious mining rig to be redirected to a malicius server :^)) the master should tell the talker (somehow) if he’s allowed to reconnect to a server with a different uri or not.

Component architecture:

a) design of the component:

Ok so, basically the component has to be a client and a server in the same time.
For this reason, we have to use the communication channels in a non blocking way.
For the communication with the server a socket is used while for communication with the master queues are used (It was a surprise to find out that they can be used in an asynch way in python).
To be able to handle those communication channels in an asynch way and not wait actively (the process is in sleeping state, thank you O.S.) I’ve used the select() function.

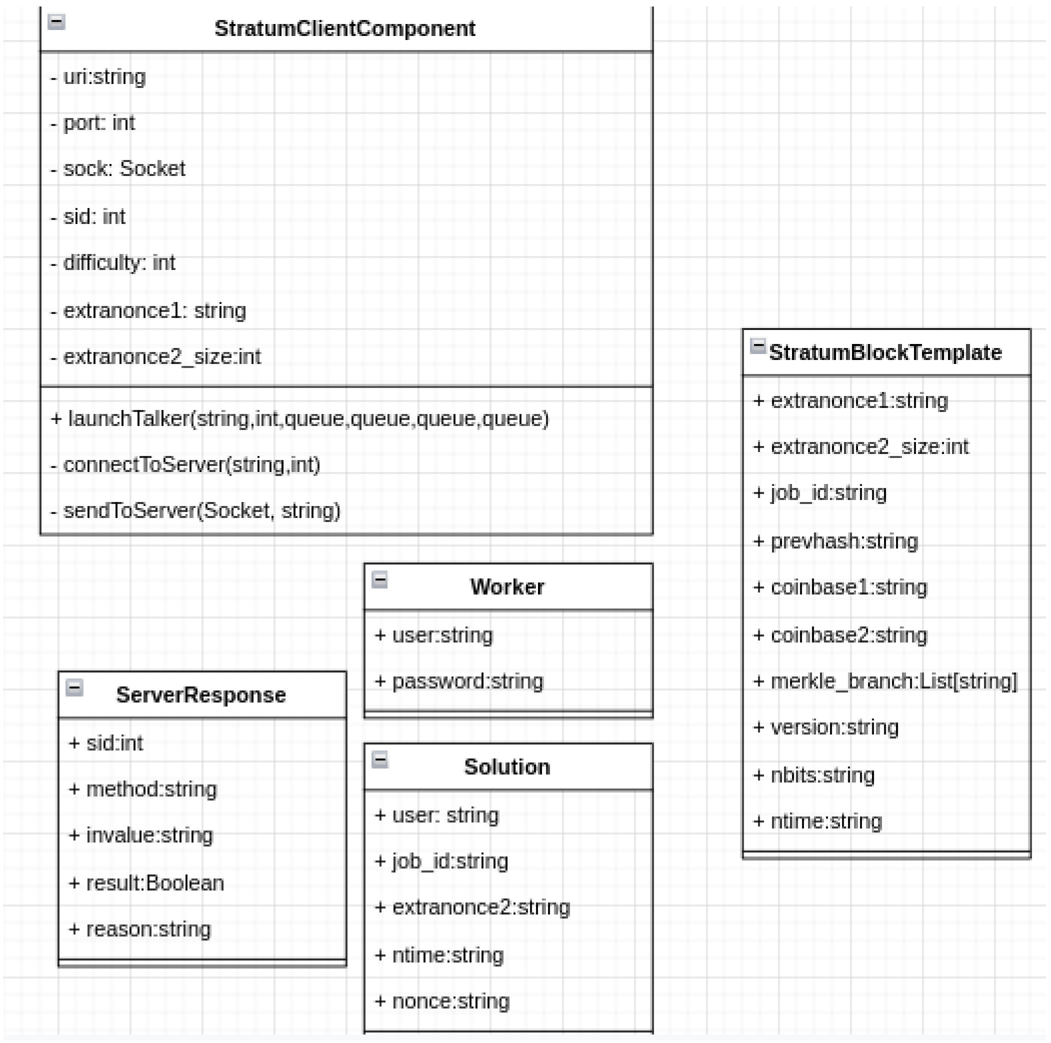
The component exposes only one method (LaunchTalker) and it requires the uri/port to connect to the server and 4 queues for communicating with the master.
The method has to runned by a different process, not the master one.

The queues are used as unidirectional communication channel, that’s why a queue for the results is needed.
The talker keeps track of the methods requested from the master and when the

response arrives, it gives to the master the id of the method, what he requested, with which parameters and what the server replied.

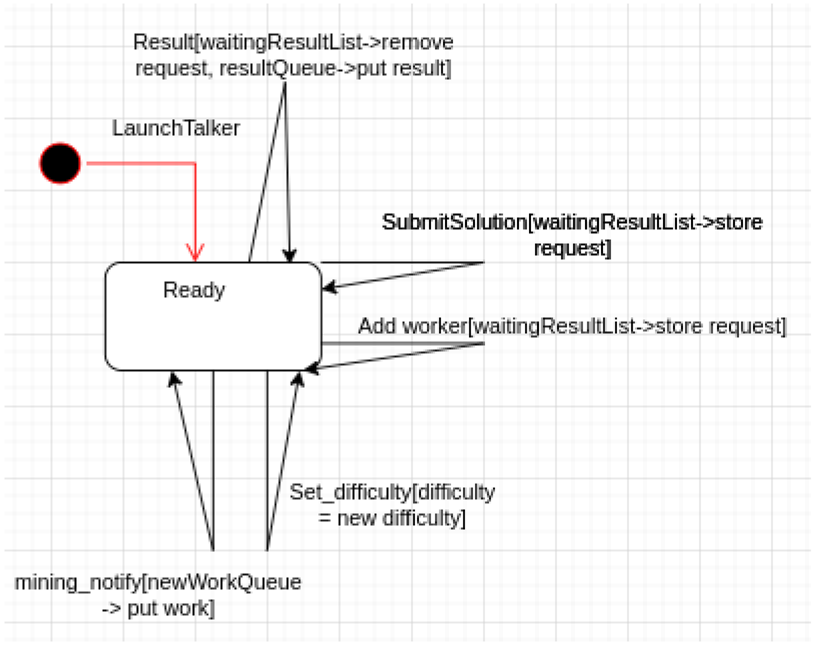
Once the method is executed, the talker initialize the stratum protocol immediatly and is ready to serve.

b) class diagram:



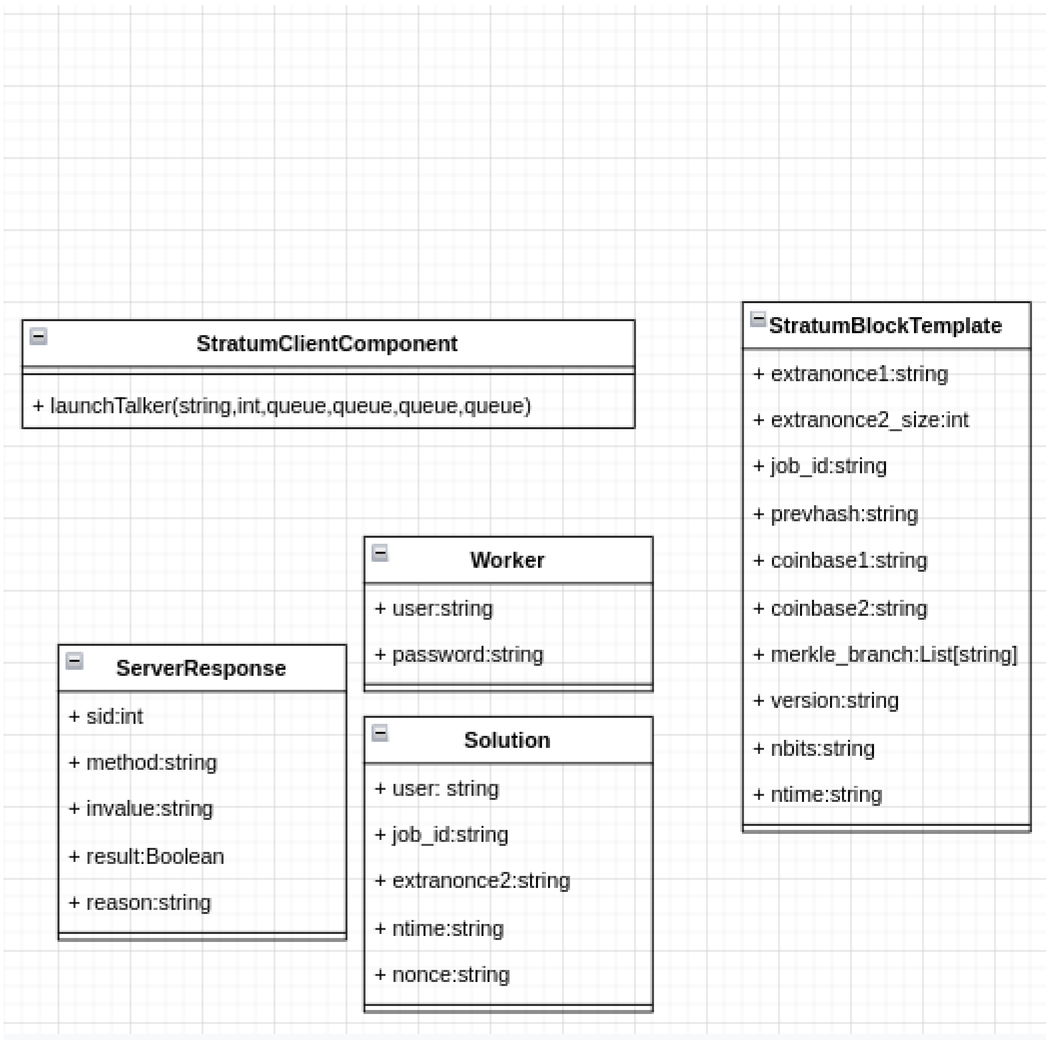
c) usage scenarios:
Already talked about that

d) state diagrams:


Result[waitingResultList->remove request, resultQueue->put result]
LaunchTalker
SubmitSolution[waitingResultList->store request]
Ready
Add worker[waitingResultList->store request]
Set_difficulty[difficulty = new difficulty]
mining_notify[newWorkQueue -> put work]

Component description:

a) class diagrams:



b) detailed class, methods and packages description:

The class StratumClientComponent exposes only one method, this method has to be called by a different process.
It accepts the uri/port of the server to connect to and 4 queues in order to interact with it.
The first one is the solution queue, put a solution here and the talker will send it to the server.
The third one is the add worker queue, put a worker here and the talker will ask the server to authorize it.
The second one is the new work queue, read from it periodically in order to get a StratumBlockTemplate.
The forth one is the result queue, read from it periodically in order to get the results of the previosly requested methods.

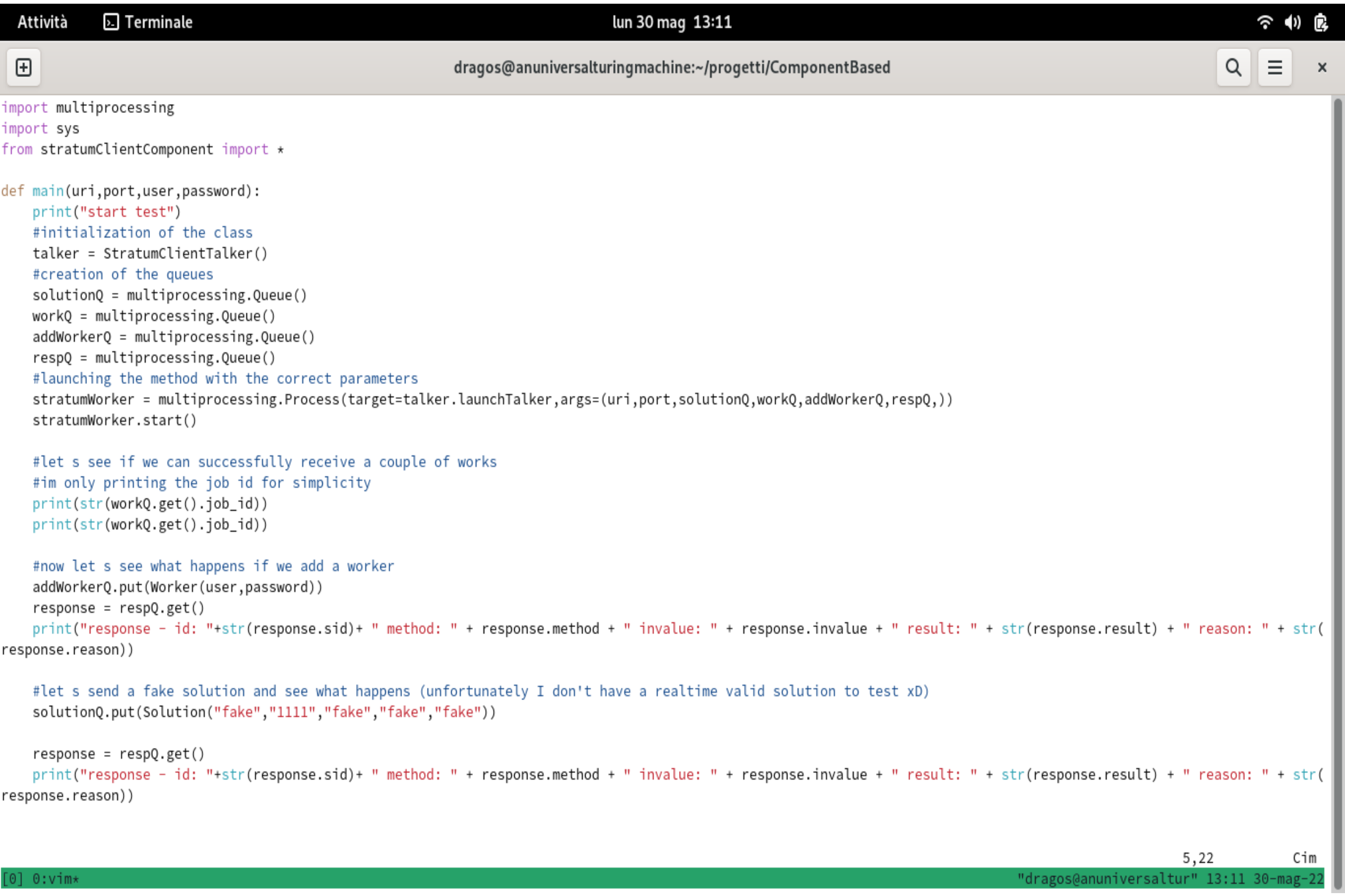
The stratumBlockTemplate is a class that encapsulates all the needed informations to start working (more infos here https://braiins.com/stratum-v1/docs).

The Solution class encapsulates all the informations needed to send the solution of the actual block to the server (more infos here https://braiins.com/stratum-v1/docs).
The worker class contains the user and the password of the worker, those infos are needed by the server.
The serverResponse class has the id of the requested (by the master) method, a string that represents the method, invalue that represent the parameters sent by the master, the result (success/failure) and the reason which has a value only in case of failure and explains the reason.

c) api documentation:

launchTalker(uri,port,solutionSubmitQueue,newWorkQueue,addWorkerQueue,responseQueue):
- uri: a string that provides the uri of the server the talker has to connect to
- port: an integer that provides the port on which the server is listening to
- solutionSubmitQueue: a multiprocessing queue, the master will put the solutions here
- addWorkerQueue: a multiprocessing queue, the master will put the new workers here
- newWorkQueue: a multiprocessing queue, when a new work arrives, it will be avaliable in this queue
- responseQueue: a multiprocessing queue, when a result of a precedent requested method arrives, it will be avaliable in this queue

d) usage examples:



```
import multiprocessing
import sys
from stratumClientComponent import *

def main(uri,port,user,password):
    print("start test")
    #initialization of the class
    talker = StratumClientTalker()
    #creation of the queues
    solutionQ = multiprocessing.Queue()
    workQ = multiprocessing.Queue()
    addWorkerQ = multiprocessing.Queue()
    respQ = multiprocessing.Queue()
    #launching the method with the correct parameters
    stratumWorker = multiprocessing.Process(target=talker.launchTalker,args=(uri,port,solutionQ,workQ,addWorkerQ,respQ,))
    stratumWorker.start()

    #let s see if we can successfully receive a couple of works
    #im only printing the job id for simplicity
    print(str(workQ.get().job_id))
    print(str(workQ.get().job_id))

    #now let s see what happens if we add a worker
    addWorkerQ.put(Worker(user,password))
    response = respQ.get()
    print("response - id: "+str(response.sid)+ " method: " + response.method + " invalue: " + response.invalue + " result: " + str(response.result) + " reason: " + str(
response.reason))

    #let s send a fake solution and see what happens (unfortunately I don't have a realtime valid solution to test xD)
    solutionQ.put(Solution("fake","1111","fake","fake","fake"))

    response = respQ.get()
    print("response - id: "+str(response.sid)+ " method: " + response.method + " invalue: " + response.invalue + " result: " + str(response.result) + " reason: " + str(
response.reason))


                                                                                                                      5,22          Cim
[0] 0:vim*                                                                                          "dragos@anuniversaltur" 13:11 30-mag-22
```

f) installation istructions and dependencies:

to use the component the following python3 libraries are needed:
- multiprocessing
- socket
- json
- sys
- errno
- select
- the provided file stratumStructs

g) requirements:
as far as I know, on Windows it may not work.
In any UNIX like OS python3 is enough

i) license:

```
       DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE
```

4)perform analysis of similar, competing components:

As far as I know (I did a quick research) there are not similar components, not commercial nor open source.