```
In [3]: proxy = Pyro5.api.Proxy(uri)
In [4]: proxy.multiply(2,3)
Out[4]: 6

Do you really want to exit ([y]/n)? y
```

But I had a Pyro5 nameserver running. Let's stop it and try again. Did not make any difference.

Anyway, this does not use callbacks. Also, the command line is too slow to test multiple threads. As far as I can see, `server_multiplexed.py` creates a server which combines a working object of class `EmbeddedServer` with `Pyro5.nameserver.NameServerDaemon` and `Pyro5.nameserver.BroadcastServer` objects. I'm not sure what it demonstrates.

Just playing around to see if I have an inspiration. `nmc_client.py` issues for `LRM()` requests of diminishing durations and then enters `requestLoop()`. `nmc_server.py` returns the duration for each LRM. This is what happened on the server:

```
LRM: called for 6
LRM: called for 4
LRM: called for 2
LRM: called for 0
LRM: sleep finished
LRM: finished 0
LRM: sleep finished
LRM: finished 2
LRM: sleep finished
LRM: finished 4
LRM: sleep finished
LRM: finished 6
```

noindent This is what happened on the client:

```
finished: 0
finished: 2
finished: 4
finished: 6
```

In a program I can generate all the requests before calling the `eventLoop`.

2.109 2020 Aug 10 Monday (223)

MonitorControl

Taking Stock

I've taken some time off to consider the big picture – what am I trying to do and what is the best approach.

- The goal is to get a system, described in Fig. 2.55 (reproduced here as Figure 2.15) of the Monitor and Control work log (page 752 in which the observer uses a browser (web pages developed using Vue) to communicate with a small local Python server using Flask. The local server uses Pyro to communicate with a remote server which wraps a master client. The master client is composed of a number of clients each communicating with its hardware server using Pyro. Each time we had to wait for the Flask socket connection to time out – about 90 s.
- We encountered a problem with the ROACH server connection, in that it appeared that a Pyro transaction between the Flask client and server was inhibited by data transfer of spectra between the spectrometer server and the spectrometer client. Each time we had to wait for the Flask socket connection to time out – about 90 s.
- Some special features of this interface which may be factors are:

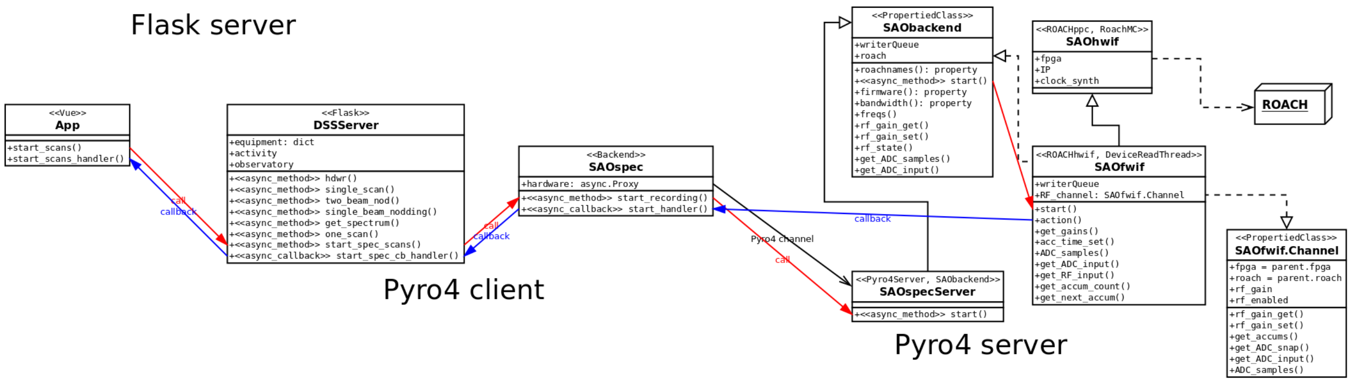

Figure 2.15: Simplified class diagram for spectrometer monitor and control.

  - The `start()` command is sent to the spectrometer server which passes it on to each individual ROACH server, which also passes the callback to the ROACH server, which the ROACH server uses to send the response back directly to the spectrometer client.
  - Each ROACH client (the ROACH itself being a server) runs in its own thread.

- The scheme uses a Pyro support module called `async` (an unfortunate choice of name in retrospect) which Dean invented. I haven't fully figured out how that works but suspect that it underlies the problem we encountered.
- I'm looking at two alternatives (oxymoron?):
  - Python 3 has an module `asyncio`. (Most of the M&C software was write in Python 2.7.) It works at the higher level with **coroutine**s and **future**s. A *coroutine* is a function or method that invokes a **Task** and then waits until a *future* (result) is returned. Class `Task` has some low-level methods called `add_done_callback()` and `remove_done_callback()`.
  - `Pyro` has a callback mechanism in which a method of a proxy is decorated `@oneway` which means that no response is returned, but a (callable) *callback* is passed as an argument. The client is also configured as a server and has an `eventLoop` which processes invocations of the callback by the server. The main problem I have with this is the event loop, which makes it impossible to instantiate the server from a Python command line and then issue commands to it to test the software. Also, I suspect that Dean's scheme was somehow based on this, since it should work just as well in Pyro4.
  - Pyro has schemes for switching between event loops, or for merging event loops (where the switching is hidden from the user). However, trying to do that with a Python or IPython main thread would be complicated, and not amenable to quick command-line testing.

So I think the answer is to go back to `asyncio`.

The issue of adapting Pyro to work with `asyncio`[158] was raised on July 26, 2018 and tagged by Irmen as an enhancement on Oct 29, 2018. This implies that it isn't straight-forward.

[158]`https://github.com/irmen/Pyro5/issues/4`