**OSPRE Basic Overview:**

* OSPRE Servers will not follow the classic Server to Client relationship. OSPRE Servers will act as both a Server and Client.
  + All Processes will be listening on a dedicated socket for connection requests (Typical Server). Specific processes will connect to others processes because they need information from said process (Typical Client).
* An abstract OSPRE server class will be written in C++ from which all process classes will be derived from.
  + The server class will contain a virtual run method, which each individual process class will implement.
  + The Server class will implement generic methods that all derived classes need such as init(), openServerSocket(), and connectToServer()
  + The Server class will be built by composition and have instances of other classes to deal with intra-process communication (details below)

**OSPRE Server Details:**

* All sockets are non-blocking TCP/IP stream sockets
* Servers will be using a call to select and registering for read and write events when interested in them
* The select call will be using timeouts to allow for the handling of error cases
* SIGPIPES are ignored so that the Server doesn’t quit when the client disconnects
* In the event that two processes lose a connection, the process acting as the client will attempt to reconnect to the process acting as the Server
* Watchdog will be able to handle multiple clients of the same type
  + All other servers will be handling only 1 type of client at a time

**OSPRE Server Helper Classes:**

ByteBuffer:

* A linear buffer class of a fixed size that will have functionality required for builders and parsers (A copy of JAVA’s built in ByteBuffer class)

Selector:

* Will contain select logic and select error handling logic (A copy of JAVA’s built in Selector class)

Builder:

* This class will take a message object and put it into a ByteBuffer so that it can be written to a Socket

Parser:

* This class will parse the ByteBuffer looking for message complete messages and creating message objects when it finds a complete message

TimedEventHandler:

* This class will contain the logic to deal with Select timeouts

**Notes:**

1. As a result of this architecture, it is assumed that a process maybe busy for a period of time and may not respond until whatever task it was working on is complete. Example - image process will process one image at a time, so while it is processing an image, it will NOT be responding to any other request and will not be sending a status update to WatchDog. The assumption is that no individual task being done by any process will be so time consuming that it will be longer than WatchDogs time out or cause excessive amounts of queuing.

2. We will have to determine how long it takes to transfer images, process images and run one iteration of the GNC algorithm. If any task is too time consuming, then that task might need to be broken up into multiple steps so that the select() loop can get serviced often enough for the processes to all be responsive. Alternatively, we may need to reevaluate either the structure of the system or the single threaded design assumption.