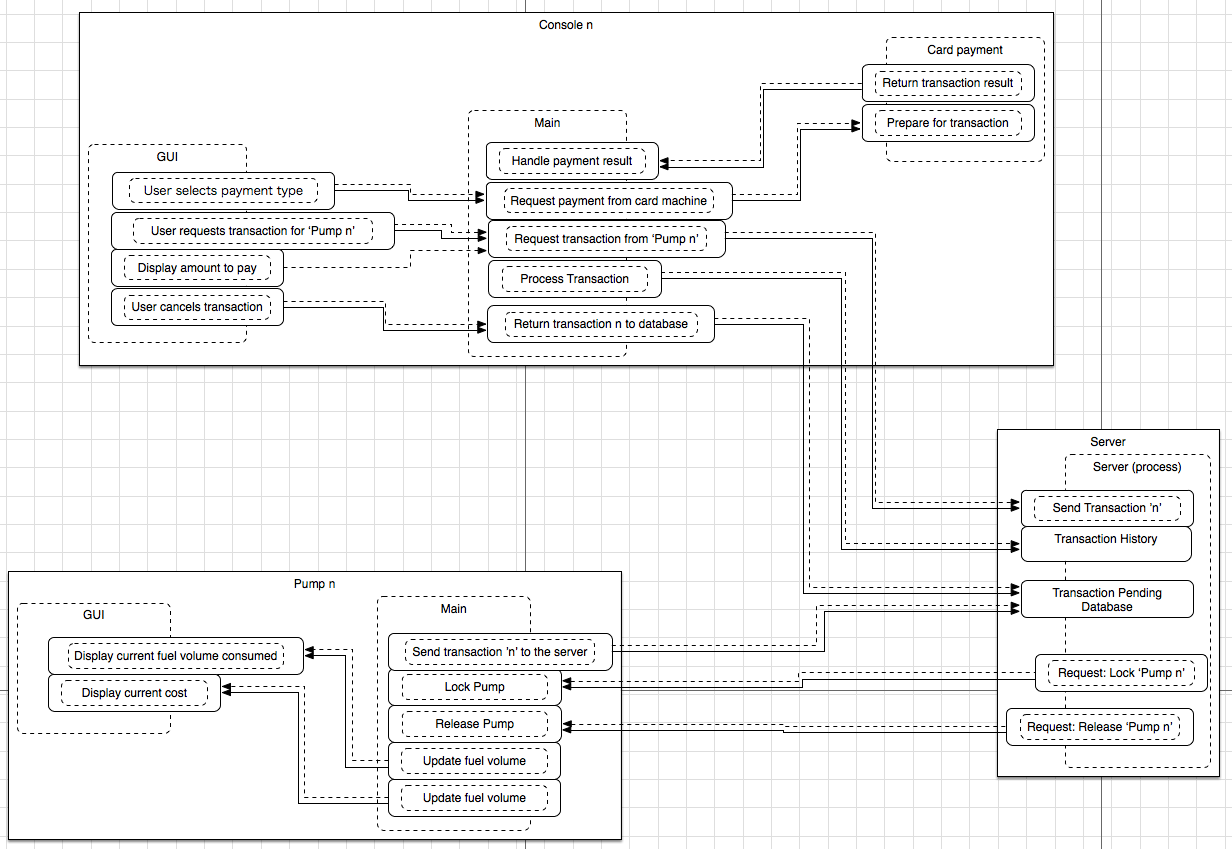
**Object Oriented Architectural Style**



**Rationale:**

I have chosen to use the ‘Object Oriented’ design style. This will consist of a main class with which all other classes, in this case the pump and console classes, interact.

The nature of computation of the desired system and the nature of computation of the object oriented architecture are compatible for the following reasons:

* Due to its data abstraction based methodology, the object oriented approach to software design lends itself to the implementation of object oriented programming in the sense that the design will be closely related to the inherent structure of an object oriented programming language.
* It is relatively easy to design a modular system when utilising object orientation.
  + Modular system design is essential for **//TODO** as it allows for individual components to be tested, debugged, upgraded or replaced with minimal risk of negatively impacting or even damaging the rest of the system.
* The various classes do not simply represent one class, but multiple instances of that class, allowing for a further development and hardware diversity in the future.

**Component and connector description:**

1. **Server**
   1. Release Pump
      1. Once the server has started, the first action would be to release all of the pumps for use. It would achieve this by sending a command to ‘2.1 Release Pump’.
   2. Lock Pump
      1. Adversely, should a pump require locking, i.e. as the server shuts down/crashes, the server would send a lock request to ‘2.2 Lock Pump’.
   3. Send Transaction
      1. Upon request from ‘3.1 Request Transaction’, the server will send transaction data from a specific pump (the data to which will be stored in ‘1.4 Fuel Taken / Cost of Fuel’) to a specific console.
   4. Fuel Taken / Cost of Fuel
      1. The data store for pump specific, unclaimed transactions. This component would receive data from ‘2.3 Send Data’. Once the transaction is complete (i.e. the Transaction ID matches that of a Transaction ID in ‘1.5 Transaction History’, it will remove the previously ‘unclaimed transaction’ from ‘1.4 Fuel Taken / Cost of Fuel’.
   5. Transaction History
      1. The data store for completed transactions. This component would receive data from ‘3.2 Process Transactions’.
2. **Pump**
   1. Release Pump
      1. Once this component receives the command from ‘1.1 Release Pump’, it will interface with the hardware with the hardware to release / unlock the pump.
   2. Lock Pump
      1. Once this component receives the command from ‘1.2 Lock Pump’, it will interface with the hardware with the hardware to lock the pump.
   3. Send Data
      1. Once the hardware signals that it’s transaction is ready, it will send the transaction data to ‘1.4 Fuel Taken / Cost of Fuel’.
3. **Console**
   1. Request Transaction
      1. Upon user interaction the console will request a transaction from a specific pump via ‘1.3.
   2. Process Transaction
      1. Once the transaction has been completed, the transaction data would be sent to ‘1.5 Transaction History’.