Project Part 2: Report

Christopher Wszolek

A20434973

cwszolek@hawk.iit.edu

CS586 Fall 2020

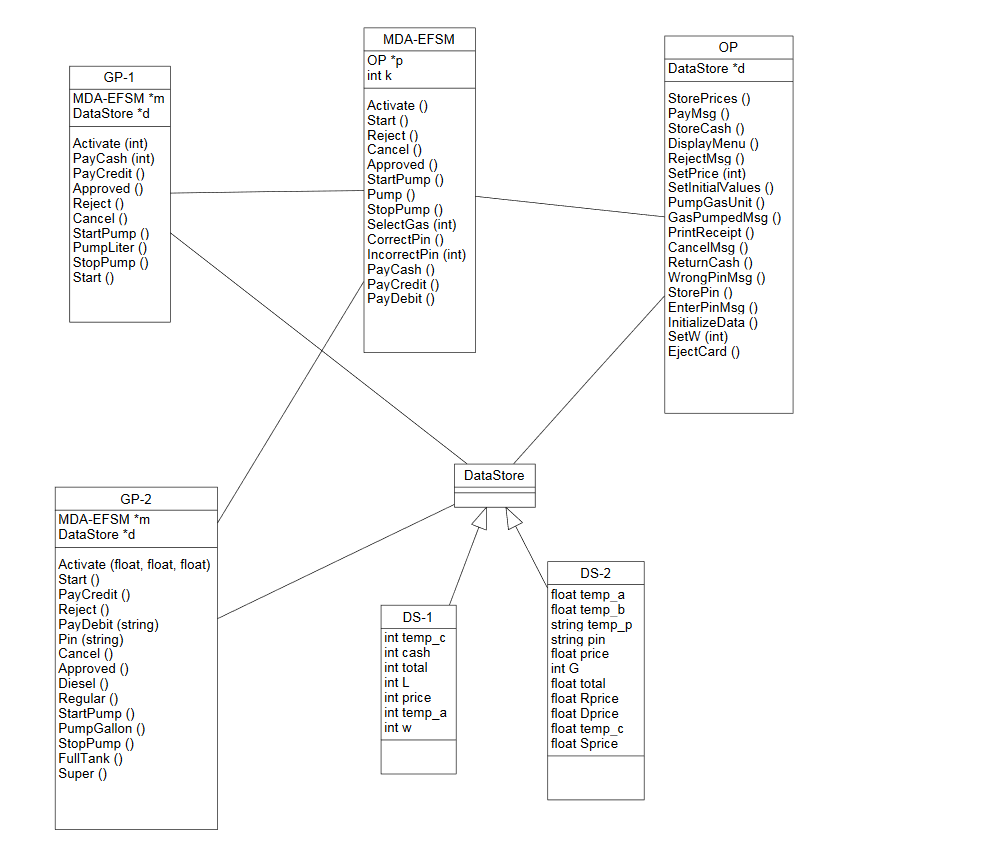
**1.**

Using provided MDA-EFSM model components, as all of the work done was based upon this.

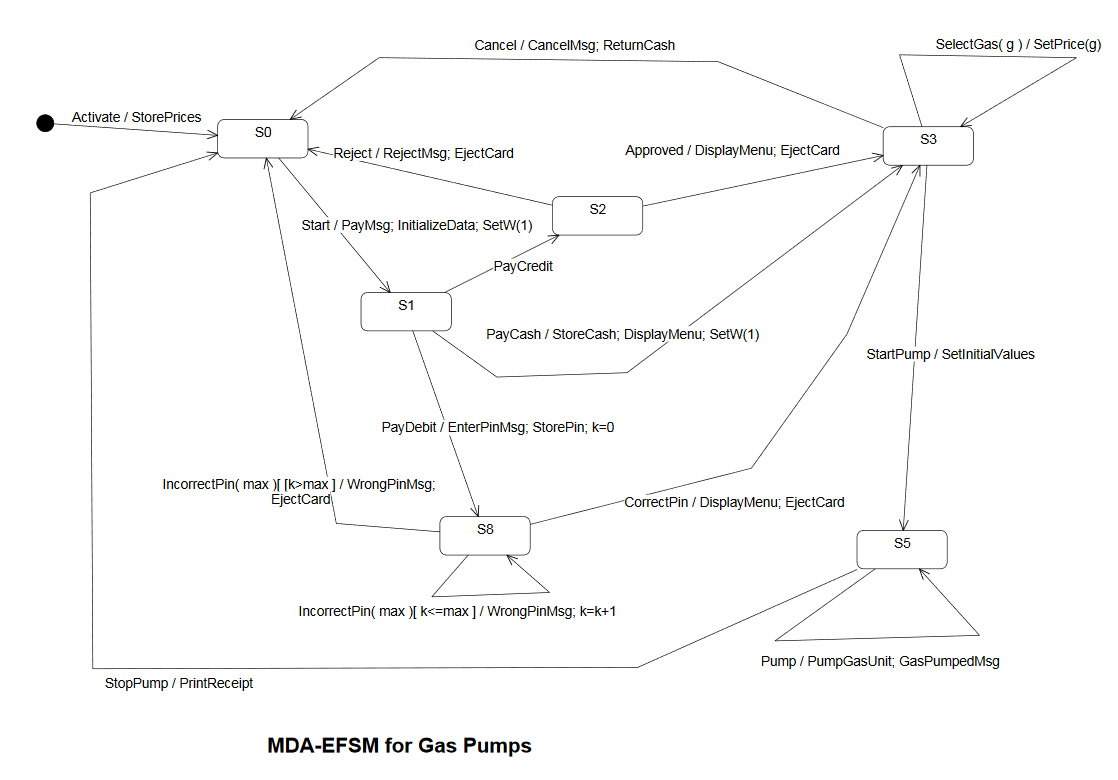
(These are nigh identical to what I would create). Thank you Professor for allowing us to use this.

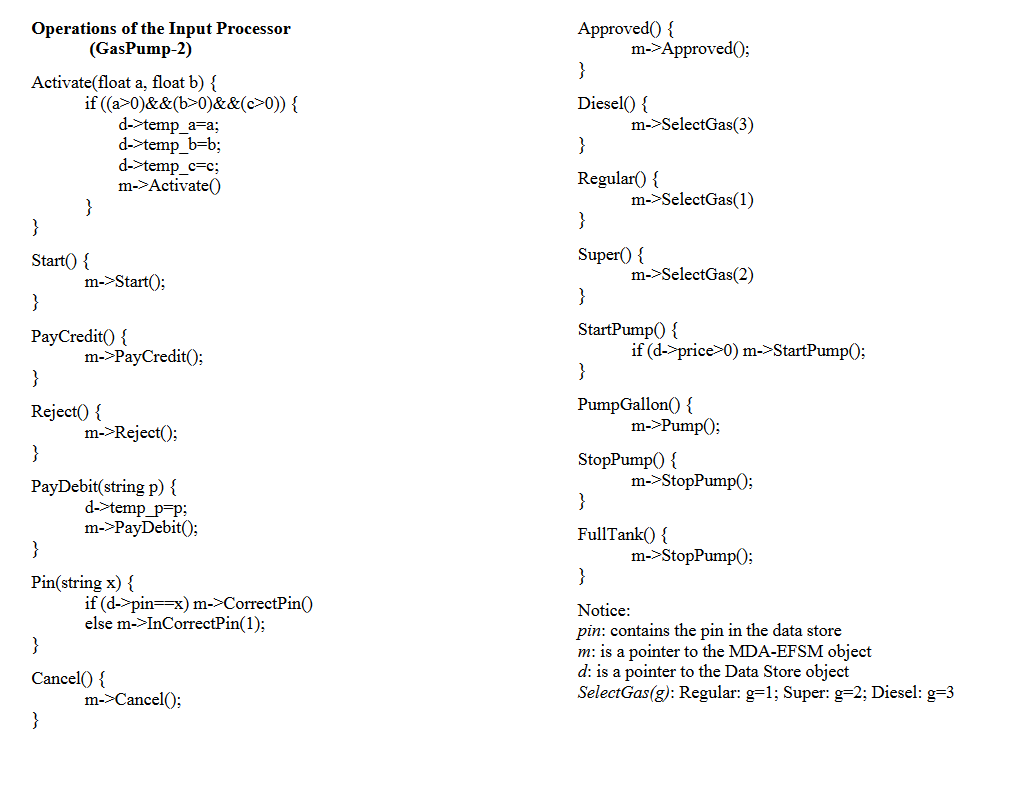
Class Diagram:

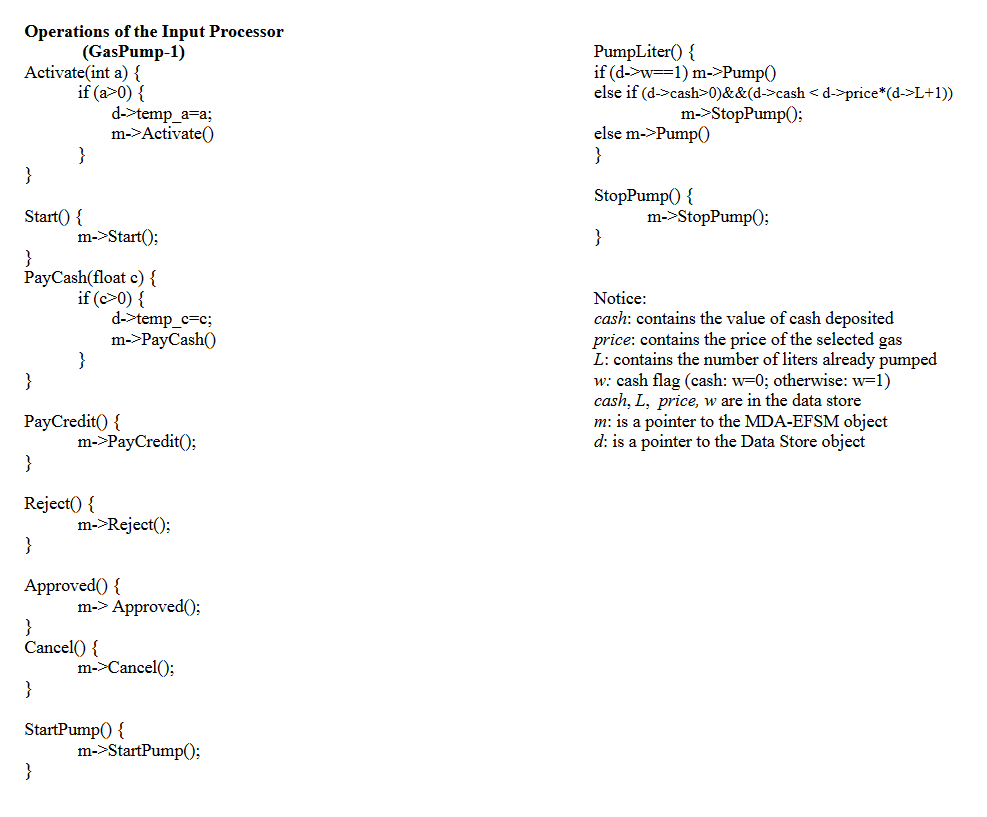




State Diagram:

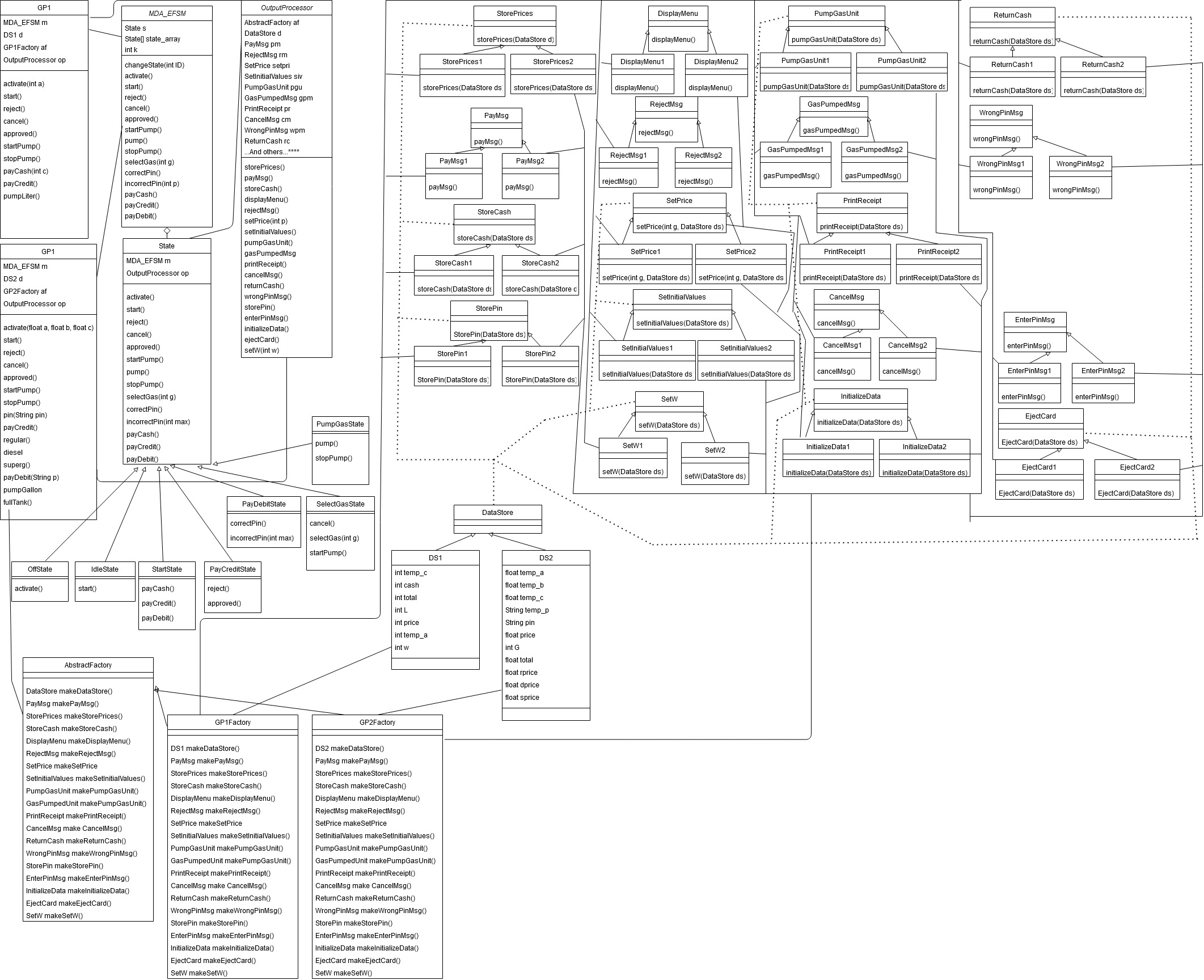




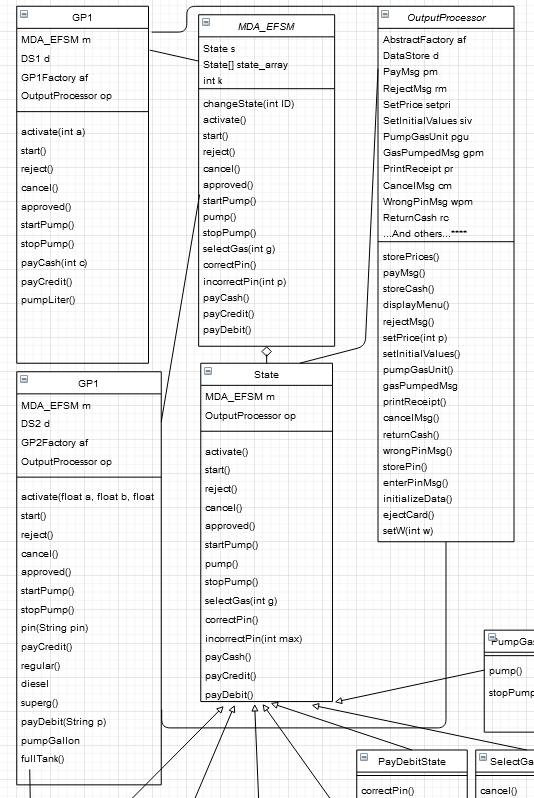


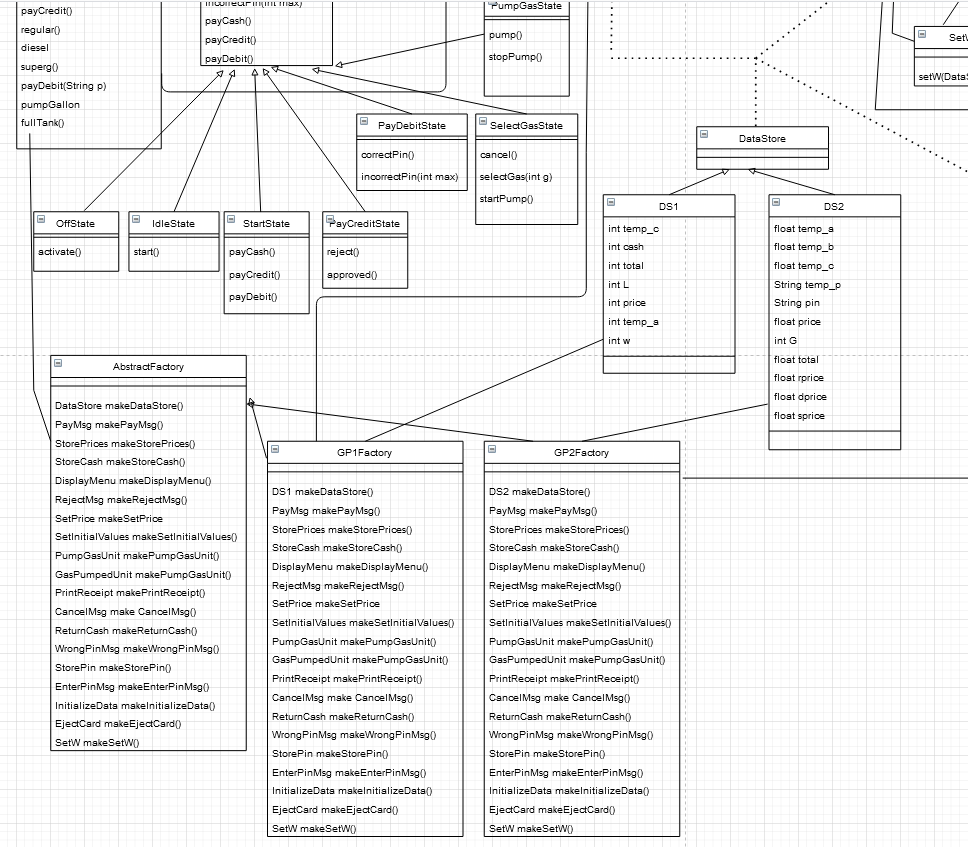
**2.**

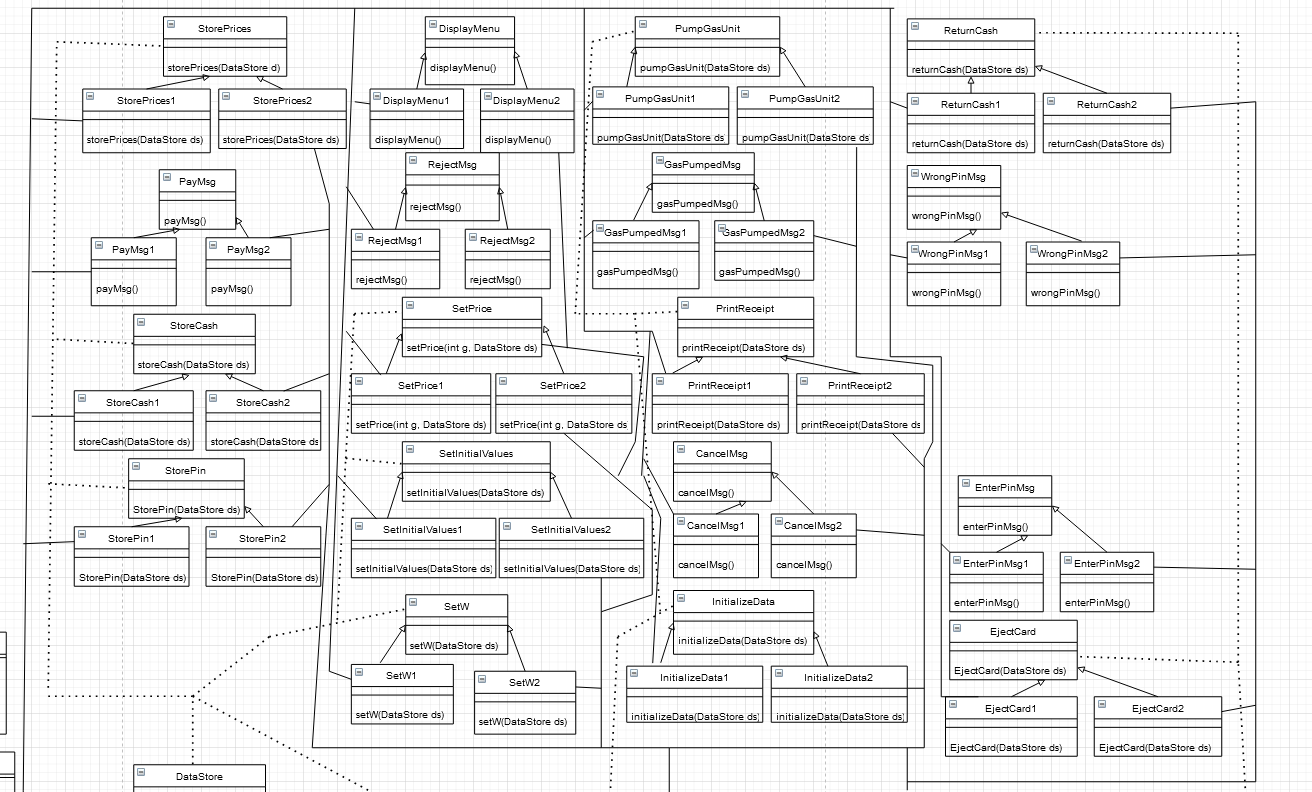
Class Diagram of entire system:



Zoomed in:







**3.**

Class Descriptions

**GP1:**

This is the primary user class of the model, and represents the Gas Pump 1. This will have calls for events that will be handled by other classes. Also creates the MDA\_EFSM, the DataStore 1, GP1Factory, and OutputProcessor. Calls for events have some checks and data storages before sent off to MDA-EFSM.

*activate(int a):* Creates a GP1Factory, a DS1, a OutputProcessor, a MDA\_EFSM, sets the value of temp\_a to the value input, and calls activate() from the MDA\_EFSM.

*start():* calls start from MDA\_EFSM

*reject():* calls reject from MDA\_EFSM

*cancel():* calls cancel from MDA\_EFSM

*approved():* calls approved from MDA\_EFSM

*startPump():* calls startPump from MDA\_EFSM

*stopPump():* calls stopPump from MDA\_EFSM

*payCredit():* calls payCredit from MDA\_EFSM

*payCash(int c):* stores the amount of cash entered by user in temp\_c, and then calls payCash from MDA\_EFSM

*pumpLiter():* checks to see if a credit or debit card is inserted, and if so calls pump() from MDA\_EFSM. If the user paid with cash, checks to ensure there is enough money inserted for another liter, and then calls pump from MDA\_EFSM the liter if so. Otherwise, calls stopPump from MDA\_EFSM.

**GP2:**

This is the primary user class of the model, and represents the Gas Pump 2. This will have calls for events that will be handled by other classes. Also creates the MDA\_EFSM, the DataStore , GP2Factory, and OutputProcessor. Calls for events have some checks and data storages before sent off to MDA-EFSM.

*activate(float a, float b, float c):*  Creates a GP2Factory, a DS2, a OutputProcessor, a MDA\_EFSM, sets the value of temp\_a to a, temp\_b to b, and temp\_c to c, and calls activate() from the MDA\_EFSM.

*start():* calls start from MDA\_EFSM

*reject():* calls reject from MDA\_EFSM

*cancel():* calls cancel from MDA\_EFSM

*approved():* calls approved from MDA\_EFSM

*startPump():* calls startPump from MDA\_EFSM

*stopPump():* calls stopPump from MDA\_EFSM

*pin(String pin):* checks to see if pin is equal to the pin stored in DS2, and if so calls correctPin from MDA\_EFSM. Otherwise, calls incorrectPin from mda\_EFSM.

*payCredit():* calls payCredit from MDA\_EFSM

*regular():* calls selectGas(1) from MDA\_EFSM

*superg():* calls selectGas(2) from MDA\_EFSM

*diesel():* calls selectGas(3) from MDA\_EFSM

*payDebit(String p):* gets pin from inserted debit card, stores it in the datastore in temp\_p, and calls payDebit from MDA\_EFSM

*pumpGallon():* calls pump() from MDA\_EFSM

*fullTank():* call stopPump() from MDA\_EFSM

**MDA\_EFSM:**

Represents the MDA-EFSM, and handles calls from different types of gas pumps and passes them to appropriate states. Part of decentralized state pattern, and responsible for changing states and holding the state and state array. Also keeps a counter for number of pin attempts.

*changeState(int ID):* changes the state that s is pointing to in the MDA\_EFSM - points to the newly desired state.

*activate(), start(), reject(), cancel(), approved(), startPump(), pump(), stopPump(), selectedGas(int g), correctPin(), incorrectPin(int p), payCash(), payCredit(), payDebit():*

All of these call the same named method from the currently pointed to state.

**State:**  
 Abstract class for state pattern, provides links to Output Processor and MDA\_EFSM. Decentralized model, and will call change state through MDA\_EFSM.

All actions are abstract.

**OffState:**

Applied state class that represents the gas pump before it has been activated. Essentially just awaits to be activated by the user.

*activate():*  calls storePrices from the OutputProcessor, and calls changeState(1) from the MDA\_EFSM

**IdleState:**

Applied State class that represents the gas pump after activation, but before a user has begun using it for gas pumping purposes.

*start():* calls payMsg(), initializeData(), setW(1) from the OutputProcessor, and calls changeState(2) in the MDA\_EFSM.

**StartState:**

Applied State class that represents the gas pump after a user has begun to use the gas pump, and is awaiting for a type of payment from the user.

*payCash():* calls storeCash(), displayMenu(), setW(0) from the OutputProcessor

*payDebit():* calls enterPinMsg(), storePin() from the OutputProcessor, sets k = 0 in the MDA\_EFSM, and calls changeState(4) in the MDA\_EFSM.

**PayCreditState:**

Applied State class that represents the gas pump that the user has chosen to pay with credit card, inserted card, and is awaiting to be told whether the card was accepted or declined.

*reject():*  calls rejectMsg() and ejectCard() from the OutputProcessor, and calls changeState(1) in the MDA\_EFSM

*approved():* calls displayMenu() and ejectCard() from the OutputProcessor, and calls changeState(5) in the MDA\_EFSM

**PayDebitState:**

Applied State class that represents the gas pump that the user has chosen to pay with a debit card, has inserted it, and has entered their pin. Will verify pin with what is the correct, or incorrect pin.

*correctPin():* Calls displayMenu() and ejectCard() from the OutputProcessor, and the calls changeState(5) from the MDA\_EFSM.

*incorrectPin():* checks to see if k <= max, and if so will call wrongPinMsg from the OutputProcessor, and then increases k by 1 in the MDA\_EFSM. If k is > max, calls wrongPinMsg() and ejectCard() from the OutputProcessor, and calls changeState(1) from the MDA\_EFSM.

**SelectGasState:**

Applied State class that represents the gas pump is awaiting for the user to choose a type of gas or to begin pumping gas, or cancel the transaction.

*selectGas(int g):* calls setPrice(g) from the OutputProcessor.

*cancel():* calls cancelMsg() and returnCash() from the OutputProcessor, and then changeState(1) from the MDA\_EFSM.

*startPump():* calls setInitialValues() from the OutputProcessor, and changeState(6) from the MDA\_EFSM.

**PumpGasState:**

Applied State class that represents the gas pump being allowed to pump gas, and the user will pump a gas of a certain amount, until stopped.

*pump():* calls pumpGasUnit() and gasPumpedMsg() from the OutputProcessor.

*stopPump():* calls printReceipt() from the OutputProcessor, and changeState(1) from the MDA\_EFSM.

**OutputProcessor:**  
 Part of the MDA-EFSM architecture, this class is responsible for getting actions from the MDA-EFSM (through the state pattern), and will send these to the appropriate kind of gas pump’s actions. Calls abstract factory to create the correct type of actions.

*OutputProcessor(AbstractFactory afToSet, DataStore ds):* stores the DataStore to be passed into actions, and creates actions that it will store in this class by using the AbstractFactory provided when called.

*storePrices(), payMsg(), storeCash(), displayMenu(), rejectMsg(), setPrice(int p), setInitialValues(), pumpGasUnit(), gasPumpedMsg(), printReceipt(), cancelMsg(), returnCash(), wrongPinMsg(), storePin(), enterPinMsg(), initializeData(), ejectCard(), setW(int w):* All of these methods call the appropriate action created by the appropriate abstract factory (IE, GP1 factory makes SetPrices1 class, and this calls setPrices() from that created class).

**Abstract Factory:**  
 Abstract class that is part of Abstract Factory Pattern that will create the appropriate type of DataStore and MDA-EFSM actions that are required for the certain kind of gas pump, according to the kind of gas pump.

All methods are Abstract.

**GP1 Factory:**

Applied class that is part of Abstract Factory Pattern that will create a DS1 DataStore and actions for the GP1 type pump.

All methods return a new class object given by the name of the method of type 1 (makeStorePrices returns StorePrices1)

**GP2 Factory:**

Applied class that is part of Abstract Factory Pattern that will create a DS2 DataStore and actions for the GP2 type pump.

All methods return a new class object given by the name of the method of type 2 (makeStorePrices returns StorePrices2)

**DataStore:**  
 Abstract class that exists to hold the type of classes for DS1 and DS2. Useful for several patterns, though used for the MDA-EFSM architecture primarily.

**DS1:**

Applied class that holds specific data necessary for GP1 processing.

**DS2:**

Applied class that holds specific data necessary for GP2 processing.

**StorePrices:**

MDA-SFM action and part of strategy pattern, this stores entered prices into the appropriate DataStorage unit.

**PayMsg:**

MDA-SFM action and part of strategy pattern, this displays a message to the user asking them to select a type of payment.

**StoreCash:**

MDA-SFM action and part of strategy pattern, this stores the amount of cash entered by user into the DataStorage unit. Does nothing for StoreCash2.

**StorePin:**

MDA-SFM action and part of strategy pattern, this stores the PIN of the debit card when inserted by user. Does nothing for StorePin1

**DisplayMenu:**

MDA-SFM action and part of strategy pattern, this displays a menu to the user about possible payment options.

**RejectMsg:**  
 MDA-SFM action and part of strategy pattern, this shows to the user when the credit card has been declined.

**SetPrice:**

MDA-SFM action and part of strategy pattern, this stores prices when activated by the user to the DataStorage unit.

**SetInitialvalues:**

MDA-SFM action and part of strategy pattern, this sets the amount of gas pumped and the total value of the transaction to 0.

**SetW:**

MDA-SFM action and part of strategy pattern, this sets w to 0 if the user pays with cash. Does nothing for SetW2

**PumpGasUnit:**

MDA-SFM action and part of strategy pattern, this pumps a unit of gas, and adds to the total by the price of the gas. Unit is Liter for PumpGasUnit1 and Gallon for PumpGasUnit2.

**GasPumpedMsg:**

MDA-SFM action and part of strategy pattern, this displays to the user the amount of gas just pumped.

**PrintReceipt:**

MDA-SFM action and part of strategy pattern, this prints a receipt that shows the amount of gas pumped and the total value of the gas pumped.

**CancelMsg:**

MDA-SFM action and part of strategy pattern, this tells the user that the current transaction has been cancelled.

**InitializeData:**

MDA-SFM action and part of strategy pattern, if there are multiple kinds of gas, sets the selected price of gas to 0. Does nothing for InitializeData1.

**ReturnCash:**

MDA-SFM action and part of strategy pattern, this returns cash to the user if transaction is cancelled. Does nothing for ReturnCash2

**WrongPinMsg:**

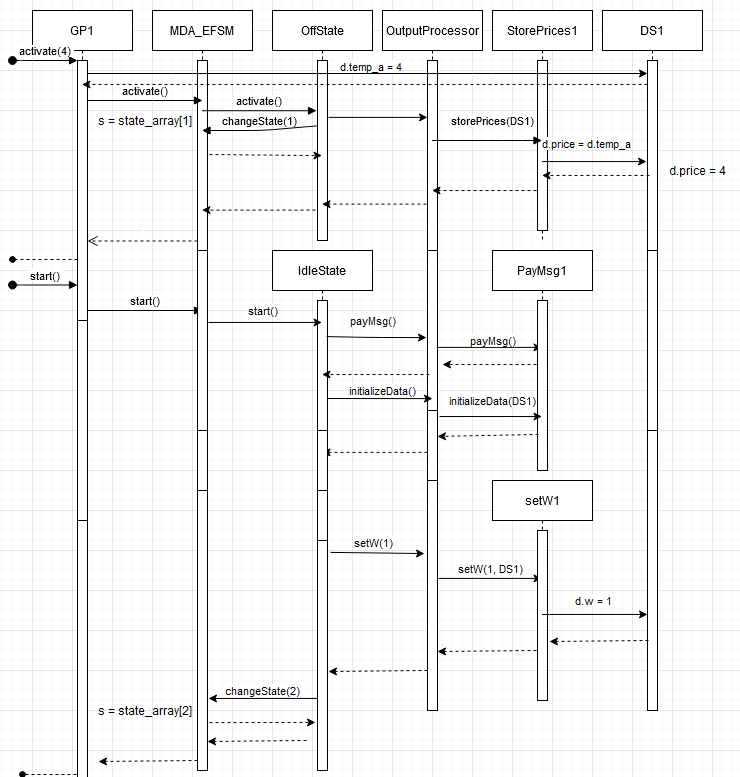
MDA-SFM action and part of strategy pattern, this lets the user know that the wrong pin was entered for a debit card transaction. Does nothing for WrongPinMsg1

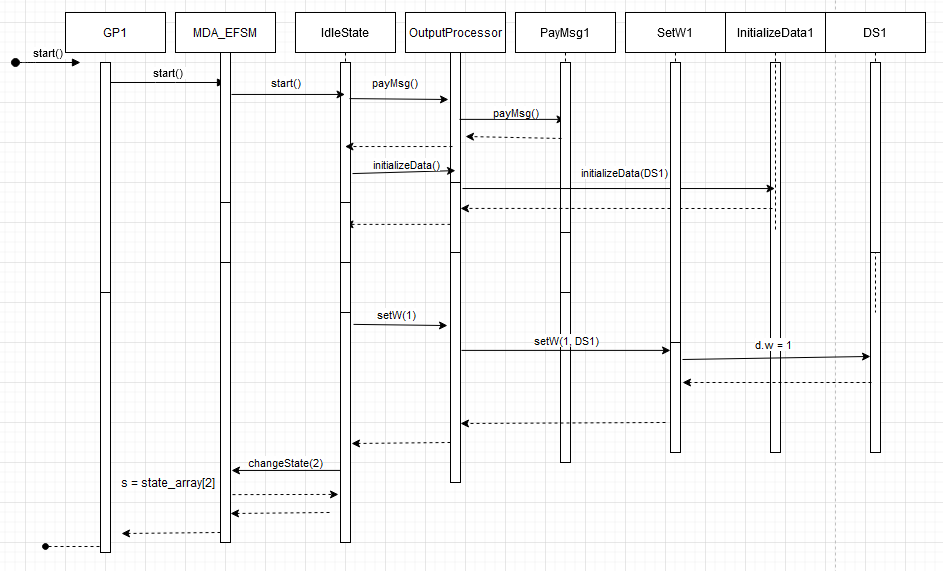
**EnterPinMsg:**  
 MDA-SFM action and part of strategy pattern, this displays a message to the user to enter a pin for a debit card transaction. Does nothing for WrongPinMsg1

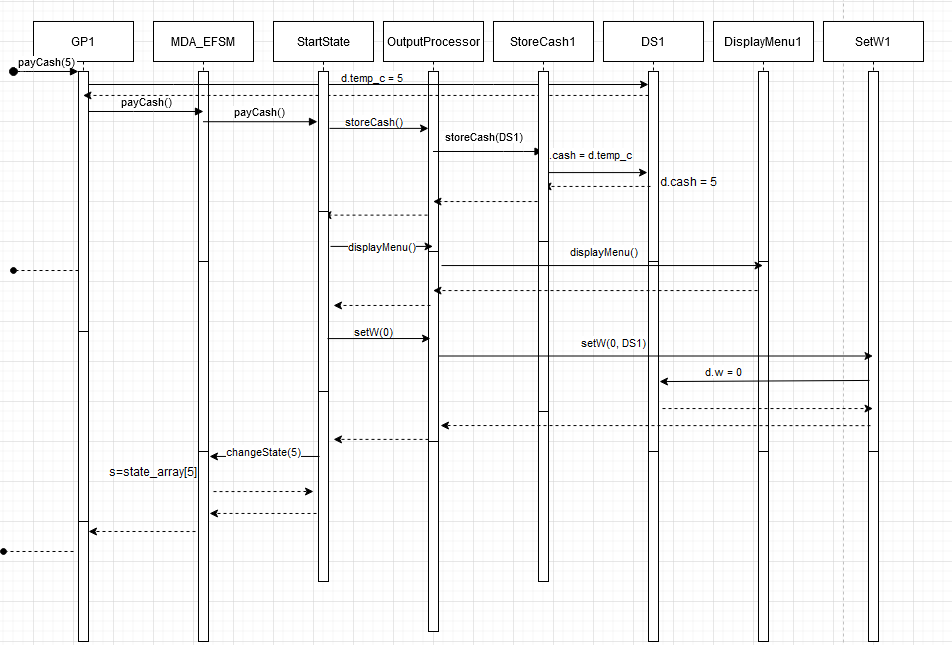
**EjectCard:**  
 MDA-SFM action and part of strategy pattern, this returns an inserted debit or credit card to the user.

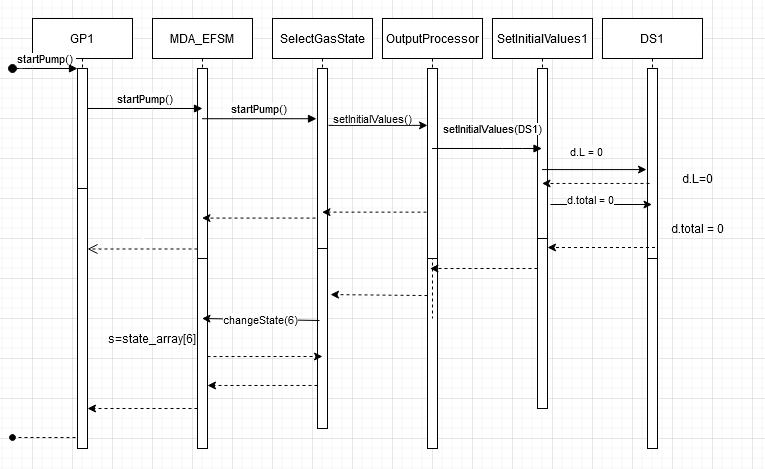
**4.**

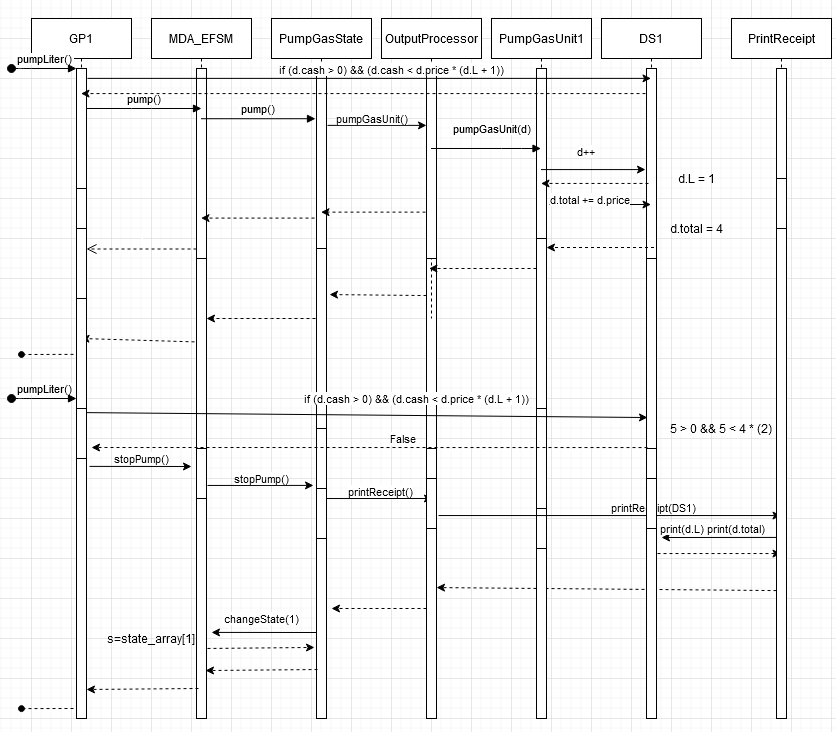
Sequence Diagram A:











Sequence Diagram B:

