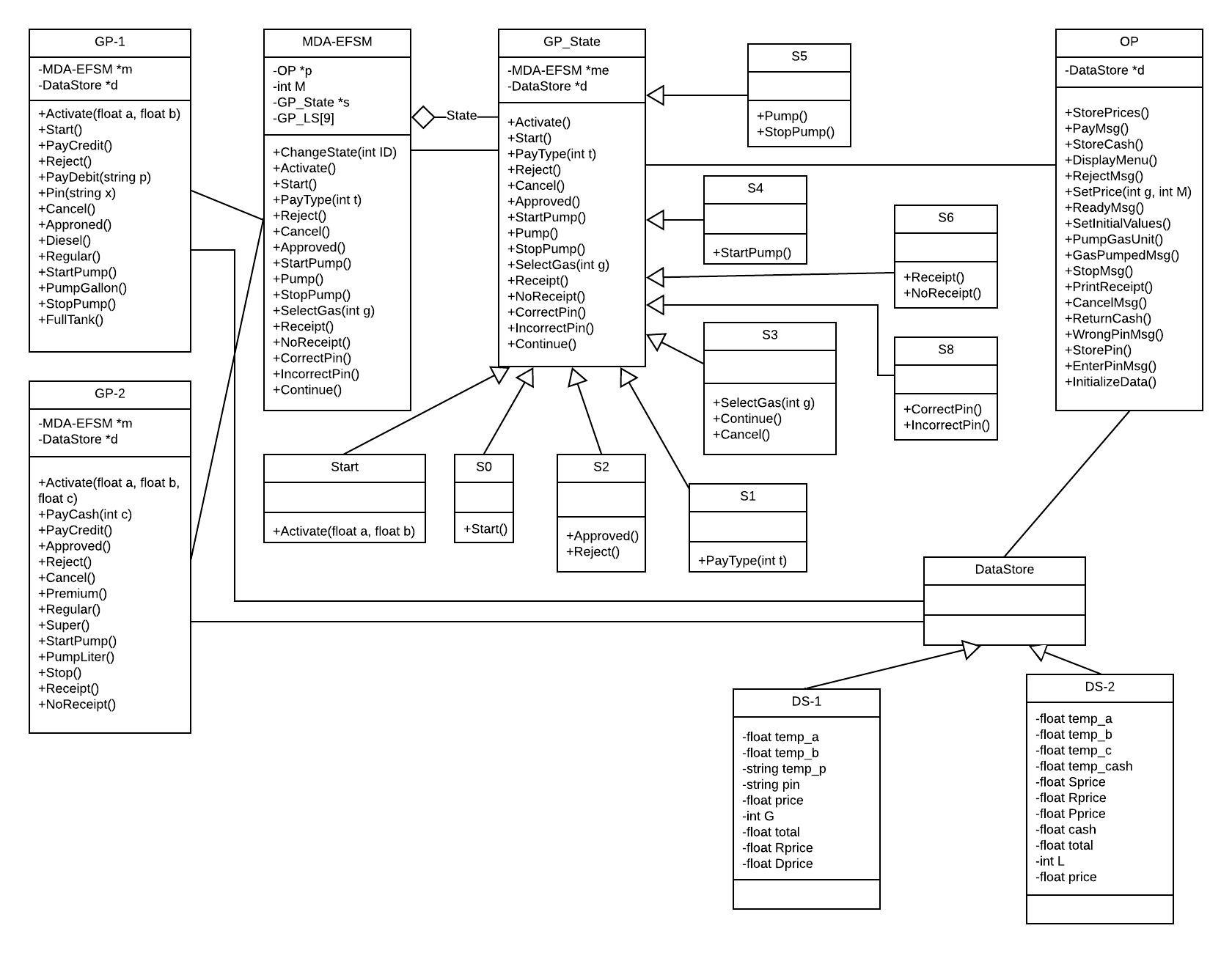
1. Class diagrams of the MDA of the GasPump components.

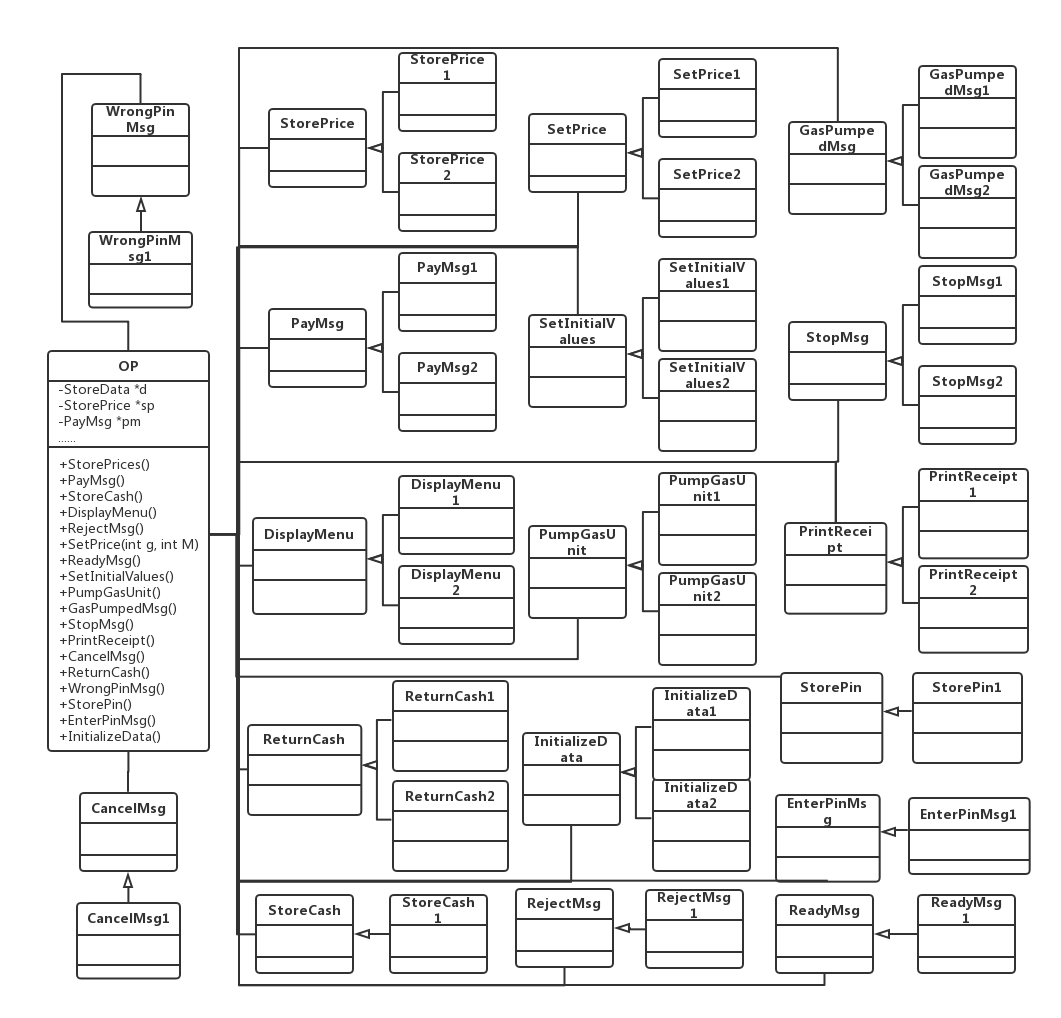
Separate the whole class diagram into three parts according to three patterns.

1. State pattern



In state pattern, the MDA-EFSM component is a state-based component. This project designed the gas pump system by using de-centralized version of state pattern. OP class will be the connection between state pattern and strategy pattern.

1. Strategy pattern



In strategy pattern, this project provides two strategies for all operations in out processor with respect to GasPump1 and GasPump2. Among these classes, EnterPinMsg, StorePin and WrongPinMsg classes are only involved in GasPump1 which provides PayDebit method. Store Cash are only involved in GasPump2 which provides PayCash method.

1. Class diagram description

a. Describe the purpose of the class, i.e., responsibilities.

b. Describe the responsibility of each operation supported by each class.

In this document, I will describe each class and its operations for different patterns.

GP-1 and GP-2 are input processers. User input is gotten by these two classes. They contain the operations related to the properties of GasPump1 and GasPump2.

The GasPump-1 component supports the following operations:

Activate (float a, float b) // the gas pump is activated where a is the price of the Regular gas

// and b is the price of Diesel gas per gallon

Start() //start the transaction

PayCredit() // pay for gas by a credit card

Reject() // credit card is rejected

PayDebit(string p) // pay for gas by a debit card, where p is a pin #

Pin(string x) // pin # is provided, where x represents the pin #

Cancel() // cancel the transaction

Approved() // credit card is approved

Diesel() // Diesel gas is selected

Regular() // Regular gas is selected

StartPump() // start pumping gas

PumpGallon() // one gallon of gas is disposed

StopPump() // stop pumping gas

FullTank() // Tank is full and pump is stopped

The GasPump-2 component supports the following operations:

Activate (float a, float b, float c) // the gas pump is activated where a is the price of Super gas, b is //the price of Regular gas and c is the price of Premium gas per liter

PayCash(int c) // pay for gas by cash, where c represents prepaid cash

PayCredit() // pay for gas by a credit card

Approved() // credit card is approved

Reject() // credit card is rejected

Cancel() // cancel the transaction

Premium() // Premium gas is selected

Regular() // Regular gas is selected

Super() // Super gas is selected

StartPump() // start pumping gas

PumpLiter() // one liter of gas is disposed

Stop() // stop pumping gas

Receipt() // Receipt is requested

NoReceipt() // No receipt

DataStore:

This is an abstract class for DS-1 and DS-2.

DS-1 :

This class contains all variables that we get from GasPump1 and the data we need to store for GasPump1.

DS-2:

This class contains all variables that we get from Gaspump2 and the data we need to store for GasPump2.

1. State pattern

This project chooses de-centralized State Pattern.

The MDA-EFSM component is a state-based component. This contains the list of classes which follows in state:

SL[0]-Start, SL[1]-S0, SL[2]-S1, SL[3]-S2, SL[4]-S3, SL[5]-S4, SL[6]-S5, SL[7]-S6, SL[8]-S8.

It provides some events:

ChangeState(int ID) // change gas pump state

Activate() //call activate() in GP\_State class

Start() //call start() in GP\_State class

PayType(int t) //credit: t=1; cash: t=2; debit: t=3 , call PayTepy(int t) in GP\_State class

Reject() //call Reject() in GP\_State class

Cancel() //call Cancel() in GP\_State class

Approved() //call Approved() in GP\_State class

StartPump() //call StartPump() in GP\_State class

Pump() //call Pump() in GP\_State class

StopPump() //call StopPump() in GP\_State class

SelectGas(int g) // Regular: g=1; Super: g=2; Premium: g=3; Diesel: g=4 , call SelectGas(int g) in GP\_State class

Receipt() //call Receipt() in GP\_State class

NoReceipt() //call NoReceipt() in GP\_State class

CorrectPin() //call CorrectPin() in GP\_State class

IncorrectPin() //call IncorrectPin() in GP\_State class

Continue() //call Continue() in GP\_State class

GP\_State:

This is an abstract class for all the states (Start, S0, S1, S2, S3, S4, S5, S6, S8)

Start:

This is the “Start” state of Gas Pump.

Activate() //activate the gas pump, change the state from “Start” to “S0”

S0:

Start() //start the transition, change the state from “S0” to “S1”

S1:

PayType(int t) //choose the pay type. t = 1, pay credit , change the state from “S1” to “S2”; t = 2, pay cash, change the state from “S1” to “S3”; t = 3 pay debit, change the state from “S1” to “S8”

S2:

Approved() //credit card is valid, change state from “S2” to “S3”

Reject() //credit card is rejected, change state from “S2” to “S0”

S3:

SelectGas(int g) //select gas type

Continue() //change state from “S3” to “S4”

Cancel() //change state from “S3” to “S0”

S4:

StartPump() //start to pump, change the state from “S4” to “S5”

S5:

Pump() //pump gas

StopPump() //stop to pump,change the state from “S5” to “S6”

S6:

Receipt() //print the receipt, change the state from “S6” to “S0”

NoReceipt() //don’t print receipt, change the state from “S6” to “S0”

S8:

CorrectPin() //pin is valid, change state from “S8” to “S3”

IncorrectPin() //pin is wrong, change state from “S8” to “S0”

1. Strategy pattern

OP:

This class gets the object from the concrete factory and performs the actions of MED-EFSM. But this is an abstract

class. For GasPump1 and GasPump2, we generate different instances to perform validation on different incoming

data.

StorePrices, PayMsg, StoreCash, DisplayMenu, SetPrice, ReadyMsg, SetInitialValues, PumpGasUnit, GasPumpedMsg, StopMsg, PrintReceipt, CancelMsg, ReturnCash, WrongPinMsg, StorePin, EnterPinMsg, and InitializeData are all abstract classes.

StorePrices1 & StorePrices2:

StorePrices() // stores price(s) for the gas from the temporary data store DS-1 and DS-2 for the GasPump1 and GasPump2 respectively

PayMsg1 & PayMsg2:

PayMsg() // displays a type of payment method for the GasPump1 and GasPump2 respectively

StoreCash1

StoreCash() // stores cash from the temporary data store DS-2 for the GasPump2

DisplayMenu1 & DisplayMenu2:

DisplayMenu() // display a menu with a list of selections for the GasPump1 and GasPump2 respectively

RejectMsg1

RejectMsg() // displays credit card not approved message for both the GasPump1 and GasPump2

SetPrice1 & SetPrice2:

SetPrice(int g, int M) // set the price for the gas identified by g identifier as in SelectGas(int g)( if M=1, the price

may be increased) for the GasPump1 and GasPump2 respectively

ReadyMsg1:

ReadyMsg() // displays the ready for pumping message for both the GasPump1 and GasPump2

SetInitialValues1 & SetInitialValues2:

SetInitialValues() // set G (or L) and total to 0 for the GasPump1 and GasPump2 respectively

PumpGasUnit1 & PumpGasUnit2:

PumpGasUnit() // disposes unit of gas and counts # of units disposed for the GasPump1 and GasPump2 respectively

GasPumpedMsg1 & GasPumpedMsg2:

GasPumpedMsg() // displays the amount of disposed gas for the GasPump1 and GasPump2 respectively

StopMsg1 & StopMsg2:

StopMsg() // stop pump message and receipt? msg (optionally) for the GasPump1 and GasPump2 respectively

PrintReceipt1 & PrintReceipt2:

PrintReceipt() // print a receipt for the GasPump1 and GasPump2 respectively

CancelMsg1:

CancelMsg() // displays a cancellation message for both the GasPump1 and GasPump2

ReturnCash1 & ReturnCash2:

ReturnCash() // returns the remaining cash for the GasPump1 and GasPump2 respectively

WrongPinMsg1:

WrongPinMsg() // displays incorrect pin message for the GasPump1

StorePin1:

StorePin() // stores the pin from the temporary data store for the GasPump1

EnterPinMsg1:

EnterPinMsg() // displays a message to enter pin for the GasPump1

InitializeData1 & InitializeData2:

InitializeData() // set the value of price and cash to 0 for the GasPump1 and GasPump2 respectively

1. Abstract Factory pattern

AbstractFactory class is an abstract class. GP\_Factory1 and GP\_Factory2 are inherited form it.

OP class will get the objects from GP\_Factory1 and GP\_Factory2 by call get method in AbstractFactory:

GetDS\_1(), GetDS\_2(), GetStorePrices(), GetPayMsg(), GetStoreCash(), GetDisplayMenu(), GetSetPrice(), GetReadyMsg(), GetSetInitialValues(), GetPumpGasUnit(), GetGasPumpedMsg(), GetStopMsg(), GetPrintReceipt(), GetCancelMsg(), GetReturnCash(), GetWrongPinMsg(), GetStorePin(), GetEnterPinMsg(), Getand InitializeData()

Return the objects to OP by these operations in two concrete factory classes are listed below:

GP\_Factory1:

GetDS\_1() //return DS\_1

GetStorePrices() //return StorePrices1

GetPayMsg() //return GetPayMsg1

GetDisplayMenu() //return GetDisplayMenu1

GetSetPrice() //return SetPrice1

GetReadyMsg() //return ReadyMsg1

GetSetInitialValues() //return SetInitializeValues1

GetPumpGasUnit() //return PumpGasUnit1

GetGasPumpedMsg() //return GasPumpedMsg1

GetStopMsg() //return StopMsg1

GetPrintReceipt() //return PrintReceipt1

GetCancelMsg() //return CancelMsg1

GetReturnCash() //return ReturnCash1

GetWrongPinMsg() //return WrongPinMsg1

GetStorePin() //return StorePin1

GetEnterPinMsg() //return EnterPinMsg1

Getand InitializeData() //return InitializeData1

GP\_Factory2:

GetDS\_2() //return DS\_2

GetStorePrices() //return StorePrices2

GetPayMsg() //return GetPayMsg2

GetStoreCash() //return GetStoreCash1

GetDisplayMenu() //return GetDisplayMenu2

GetSetPrice() //return SetPrice2

GetReadyMsg() //return ReadyMsg1

GetSetInitialValues() //return SetInitializeValues2

GetPumpGasUnit() //return PumpGasUnit2

GetGasPumpedMsg() //return GasPumpedMsg2

GetStopMsg() //return StopMsg2

GetPrintReceipt() //return PrintReceipt2

GetCancelMsg() //return CancelMsg1

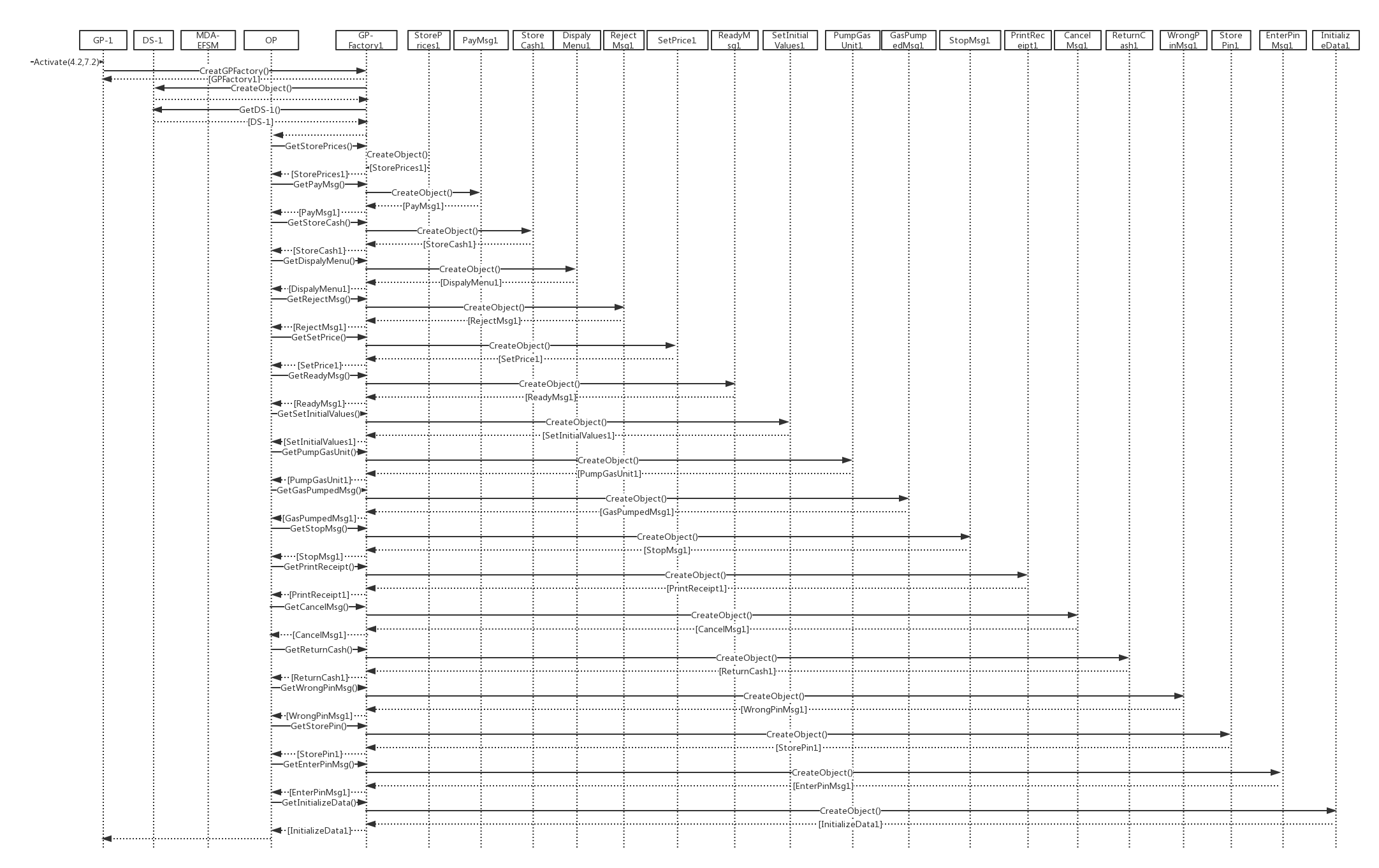
GetReturnCash() //return ReturnCash2

Getand InitializeData() //return InitializeData2

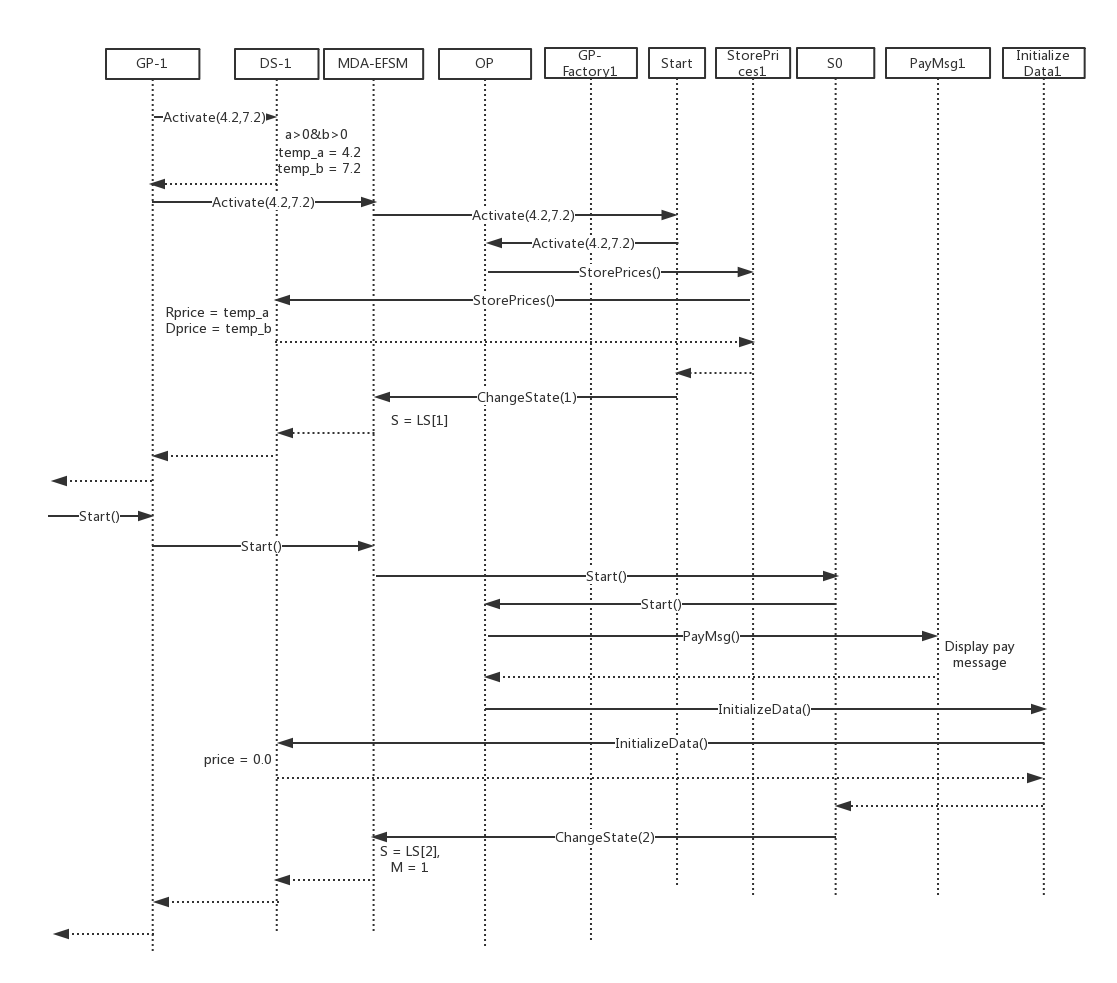
4. Dynamics. Provide two sequence diagrams for two Scenarios:

1. Scenario-1 should show how one gallon of Diesel gas is disposed in GasPump-1, i.e., the following sequence of operations is issued: Activate(4.2, 7.2), Start(), PayDebit(“abc”), Pin(“abc”), Diesel(), StartPump(), PumpGallon(), FullTank().

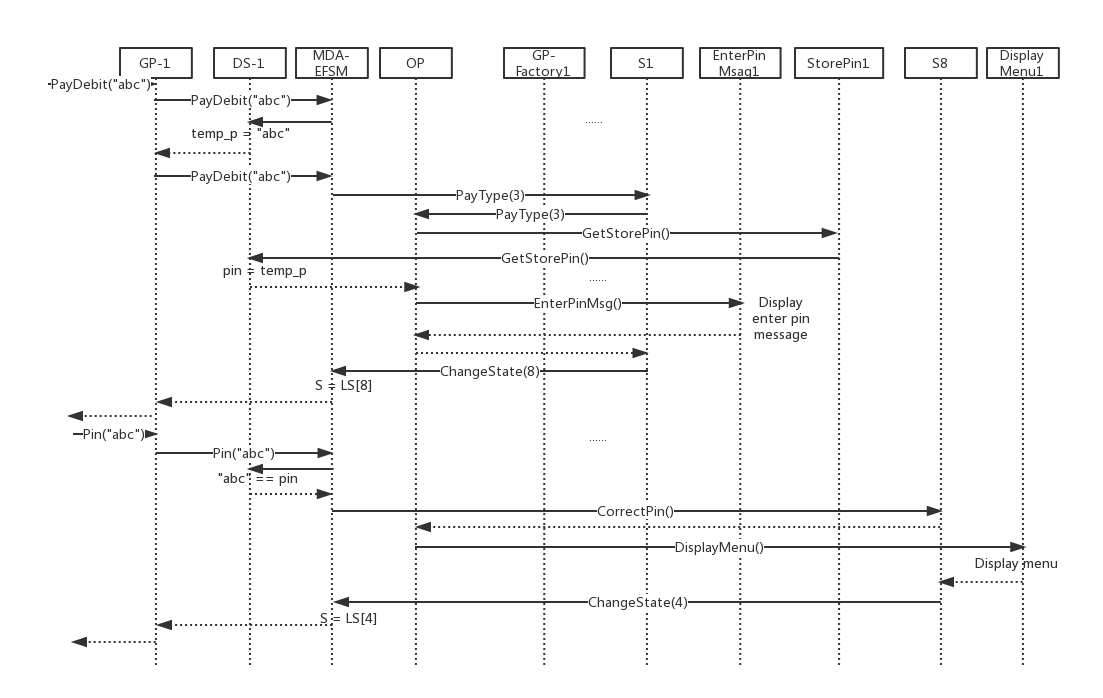
Separate sequence diagram into continuous four parts: sequence1-1, sequence1-2, sequence1-3 sequence1-4, and sequence1-5.



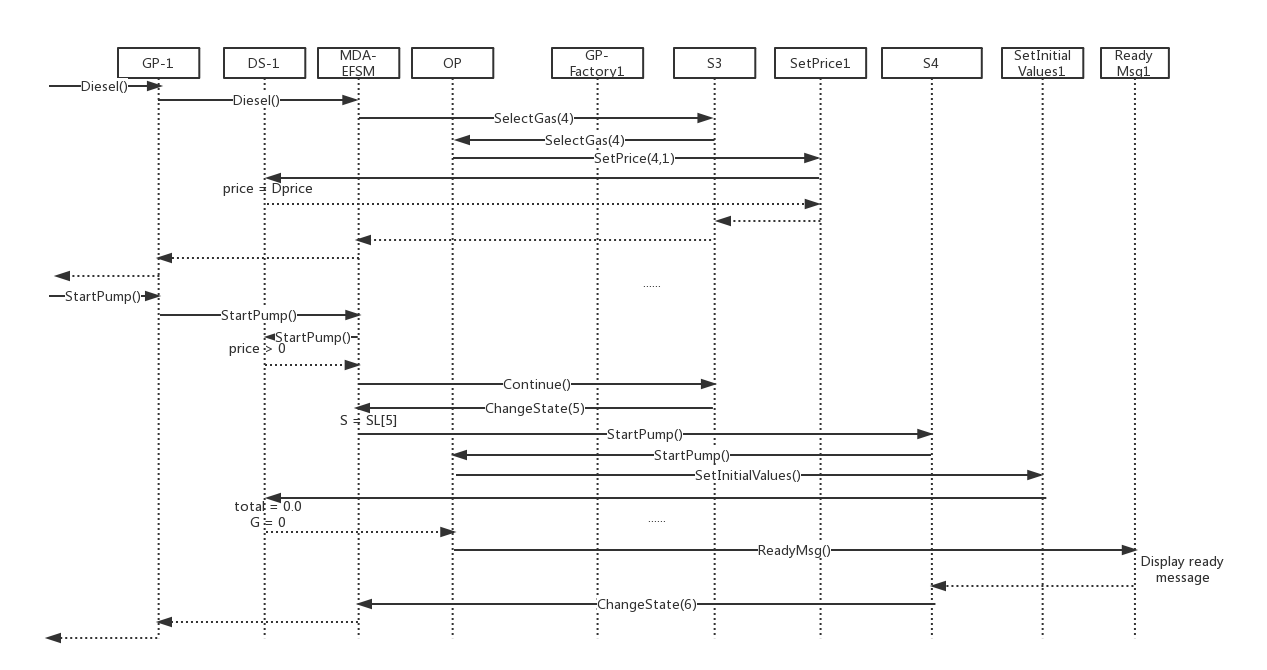
Sequence 1-1. Perform generating operations of out processor for gas pump 1 in Activate()



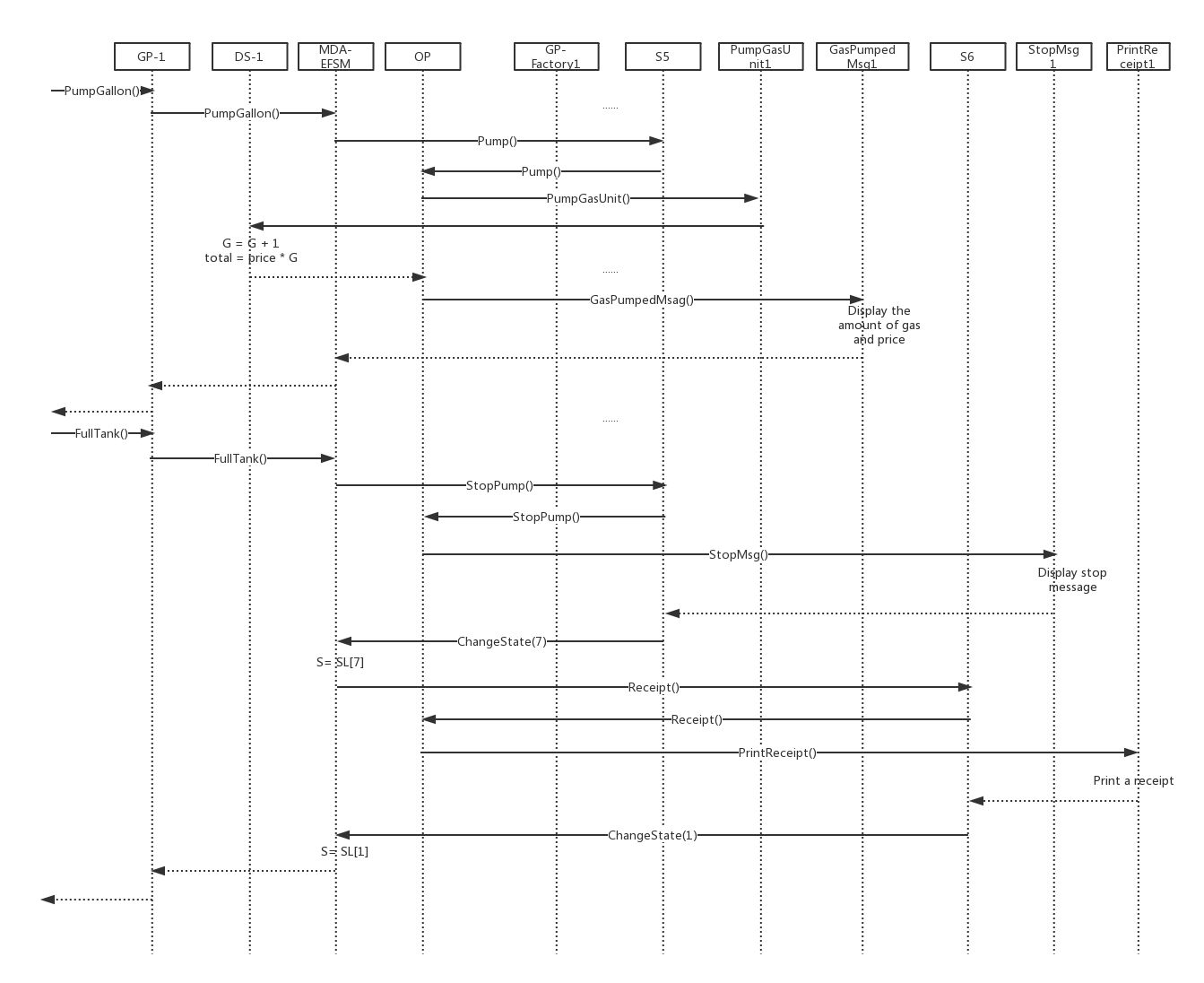
Sequence 1-2. Continue to finish Activate() and perform Pin()



Sequence 1-3. Perform PayDebit() and Pin()



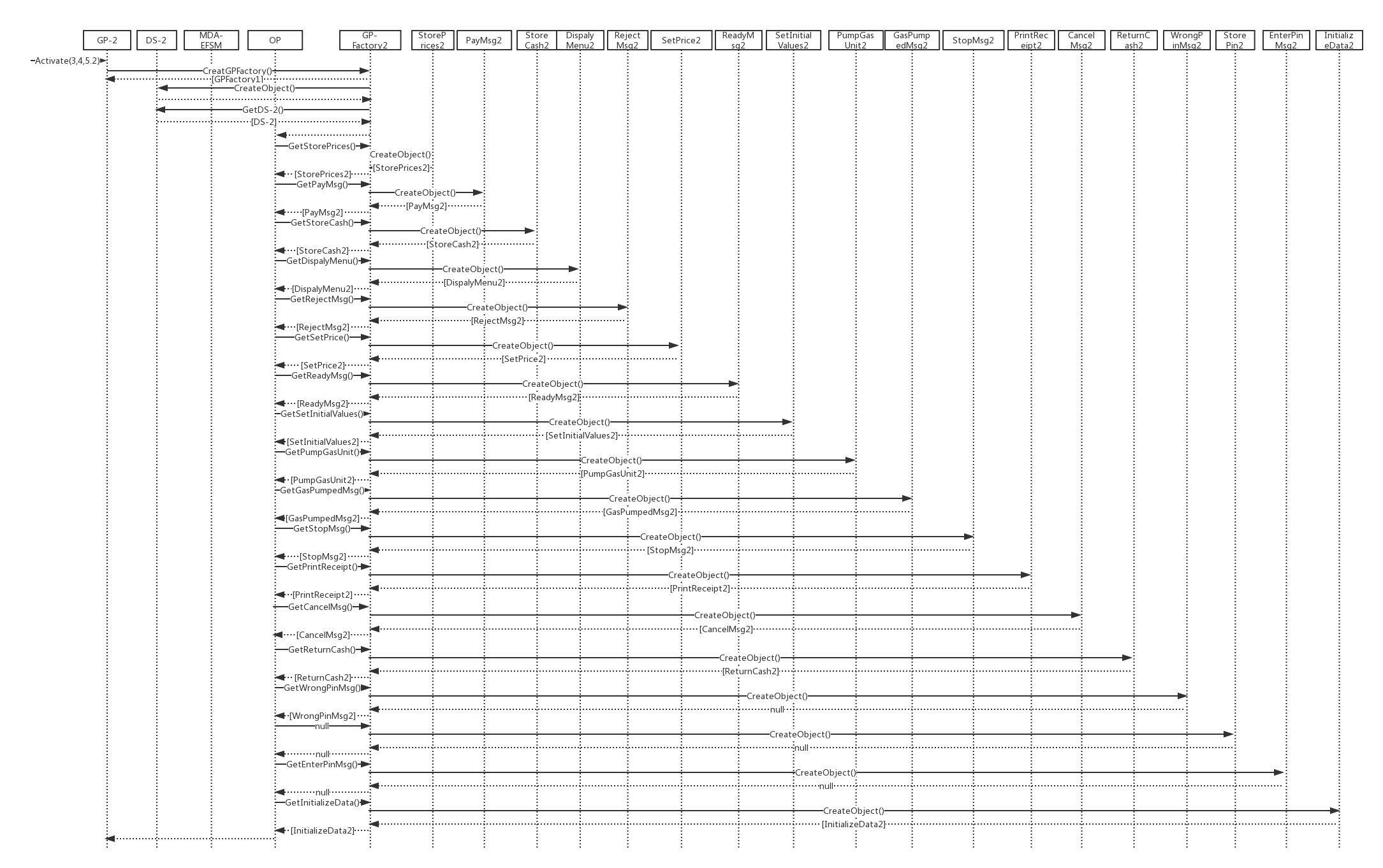
Sequence 1-4. Perform Diesel() and StartPump()



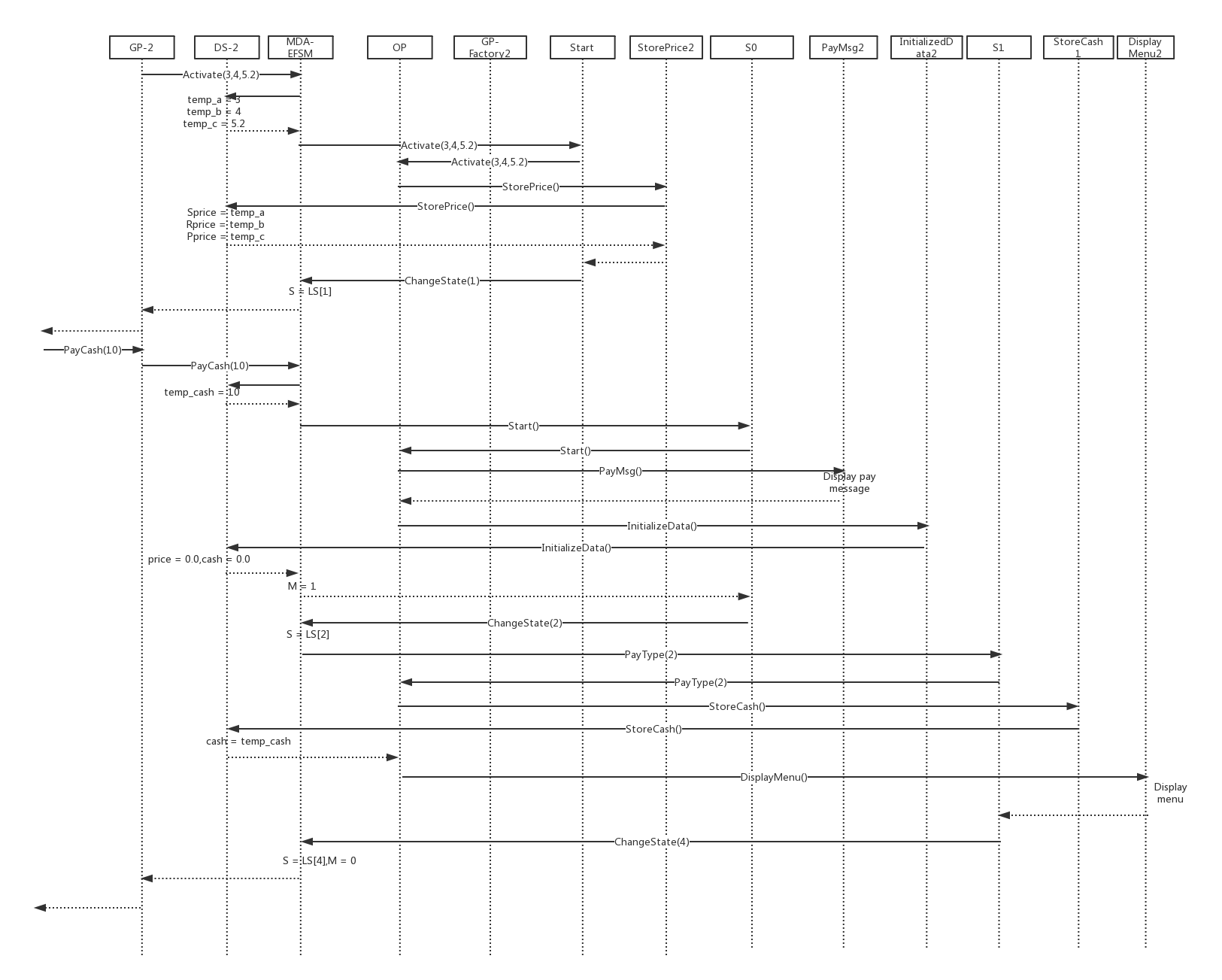
Sequence 1-5. Perform PumpGallon() and FullTank()

1. Scenario-2 should show how one liter of Premium gas is disposed in GasPump-2, i.e., the following sequence of operations is issued: Activate(3, 4, 5.2), PayCash(10), Premium(), StartPump(), PumpLiter(), PumpLiter(), NoReceipt()

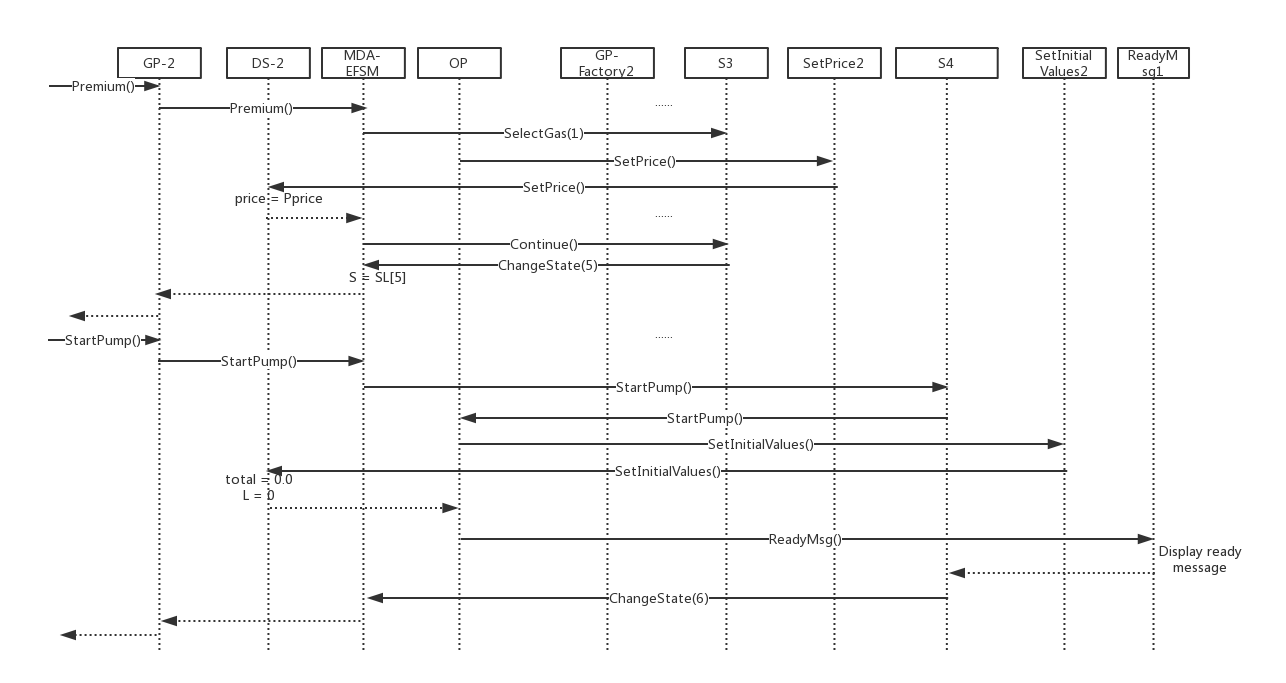
Separate sequence diagram into continuous four parts: sequence2-1, sequence2-2, sequence2-3 and sequence2-4.



Sequence 2-1. Perform generating operations of out processor for gas pump 2 in Activate()

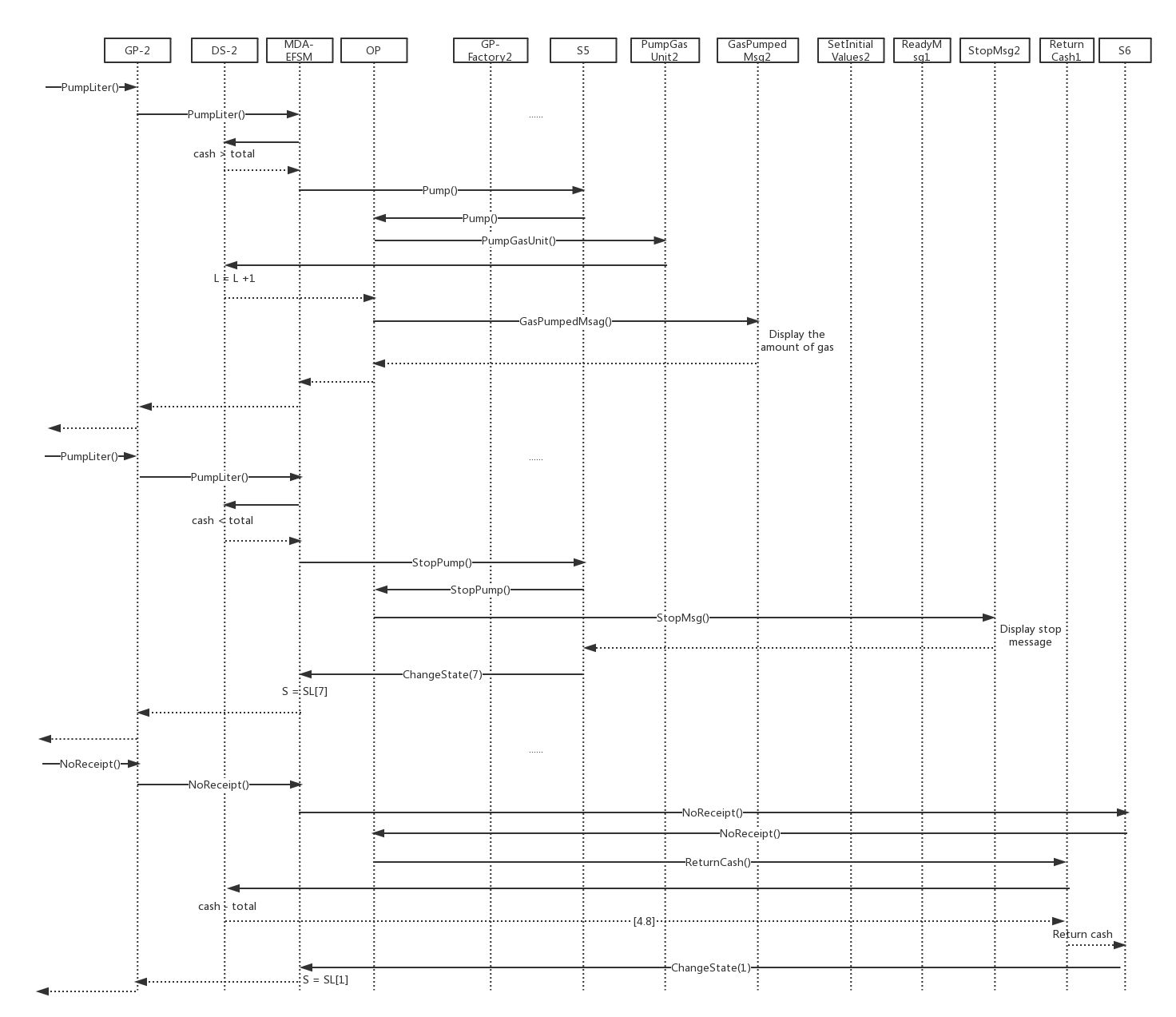


Sequence 2-2. Continue to finish Activate() and perform PayCash(10)



Sequence 2-3. Perform Premiunm and StartPump()

Sequence 2-3. Perform Premium() and StartPump()



Sequence 2-4. Perform PumpLiter(), PumpLiter and NoReceipt()

5. Source Code.

Source code will be well documented into another folder, and submitted on Blackboard.