

**Illinois Institute of Technology**College of Science  
Department of Computer Science

**FINAL REPORT**

CS 586

Submitted by:   
Jason Yeoh (A20457826)

Submitted to:  
Prof. Bogdan Korel

Department of Computer Science

# TABLE OF CONTENTS

1. MDA-EFSM
   1. Meta-events for the MDA-EFSM
   2. Meta-actions for the MDA-EFSM
   3. State diagram of the MDA-EFSM
   4. Pseudo-code of all operations of Input Processors of Gas Pumps: GP-1 and GP-2
2. Class diagrams
   1. State pattern
   2. Strategy pattern
   3. Abstract factory pattern
3. Class specifications
4. Dynamics
   1. Scenario I
   2. Scenario II

# MDA-EFSM

## Meta-events

* Activate()
* Start()
* PayCredit()
* PayCash()
* PayDebit()
* Reject()
* Cancel()
* Approved()
* StartPump()
* Pump()
* StopPump()
* SelectGas(int g)
* CorrectPin()
* IncorrectPin(int max)

## Meta-actions

|  |  |
| --- | --- |
| **Action** | **Description** |
| StorePrices | // stores price(s) for the gas from the temporary data store |
| PayMsg | // displays a type of payment method |
| StoreCash | // stores cash from the temporary data store |
| DisplayMenu | // display a menu with a list of selections |
| RejectMsg | // displays credit card not approved message |
| SetPrice(int g) | // set the price for the gas identified by g identifier as in SelectGas(int g) |
| SetInitialValues | // set G (or L) and total to 0; |
| PumpGasUnit | // disposes unit of gas and counts # of units disposed |
| GasPumpedMsg | // displays the amount of disposed gas |
| PrintReceipt | // print a receipt |
| CancelMsg | // displays a cancellation message |
| ReturnCash | // returns the remaining cash |
| WrongPinMsg | // displays incorrect pin message |
| StorePin | // stores the pin from the temporary data store |
| EnterPinMsg | // displays a message to enter pin |
| InitializeData | // set the value of price to 0 for GP-2; do nothing for GP-1 |
| EjectCard() | // card is ejected |
| SetW(int w) | // set value for cash flag |

## State diagram

# 

## Pseudocode

**Operations of the Input Processor (GP-2)**

Activate(float a, float b, float ) {

if((a>0) && (b>0) && (c>0)) {

d->temp\_a = a;  
 d->temp\_b = b;  
 d->temp\_c = c;  
 m->Activate()  
 }

}

Start() {

m->Start();

}

PayCredit() {

m->PayCredit();

}

Reject() {

m->Reject();

}

PayDebit(string p) {

d->temp\_p=p;

m->PayDebit();

}

Pin(string x) {

if (d->pin==x) m->CorrectPin()

else m->InCorrectPin(1);

}

Cancel() {

m->Cancel();

}  
   
Approved() {

m->Approved();

}

Diesel() {

m->SelectGas(3)

}

Regular() {

m->SelectGas(1)

}

Super() {

m->SelectGas(2)

}

StartPump() {

if (d->price>0) m->StartPump();

}

PumpGallon() {

m->Pump();

}

StopPump() {

m->StopPump();

}

FullTank() {

m->StopPump();

}

Notice:  
*pin*: contains the pin in the data store  
*m*: is a pointer to the MDA-EFSM object  
*d*: is a pointer to the Data Store object  
*SelectGas(g):* Regular: g=1; Super: g=2; Diesel: g=3

**Operations of the Input Processor (GP-1)**

Activate(int a) {

if(a>0) {

d->temp\_a = a;

m->Activate();

}

}

Start {

m->Start();

}

PayCash(int c) {

if(c>0) {

d->temp\_c = c;

m->PayCash();

}

}

PayCredit() {

m->PayCredit();

}

Reject() {

m->Reject();

}

Approved() {

m->Approved();

}

Cancel() {

m->Cancel();

}

StartPump() {

m->StartPump();

}

PumpLiter() {

if (d->w == 1)

m ->Pump()

else if (d->cash>0) && (d->cash < d->price\*(d->L+1))

m->StopPump();

else

m->Pump()

}

StopPump() {

m->StopPump()

}

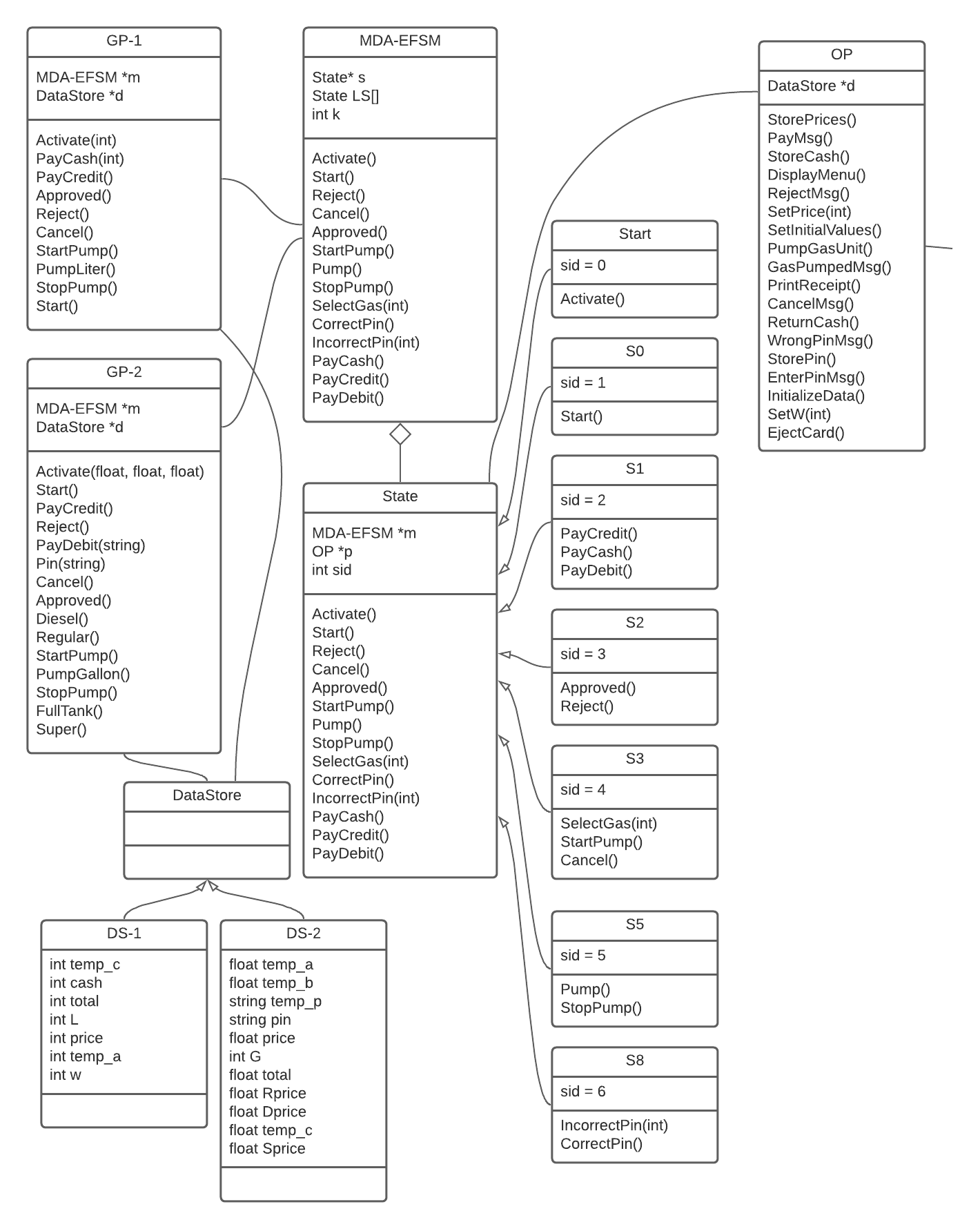
Notice:

*cash*: contains the value of cash deposited

*price*: contains the price of the selected gas  
*L*: contains the number of liters already pumped w: cash flag (cash: w=0; otherwise: w=1)  
*cash, L, price, w* are in the data store  
*m*: is a pointer to the MDA-EFSM object  
d: is a pointer to the Data Store object

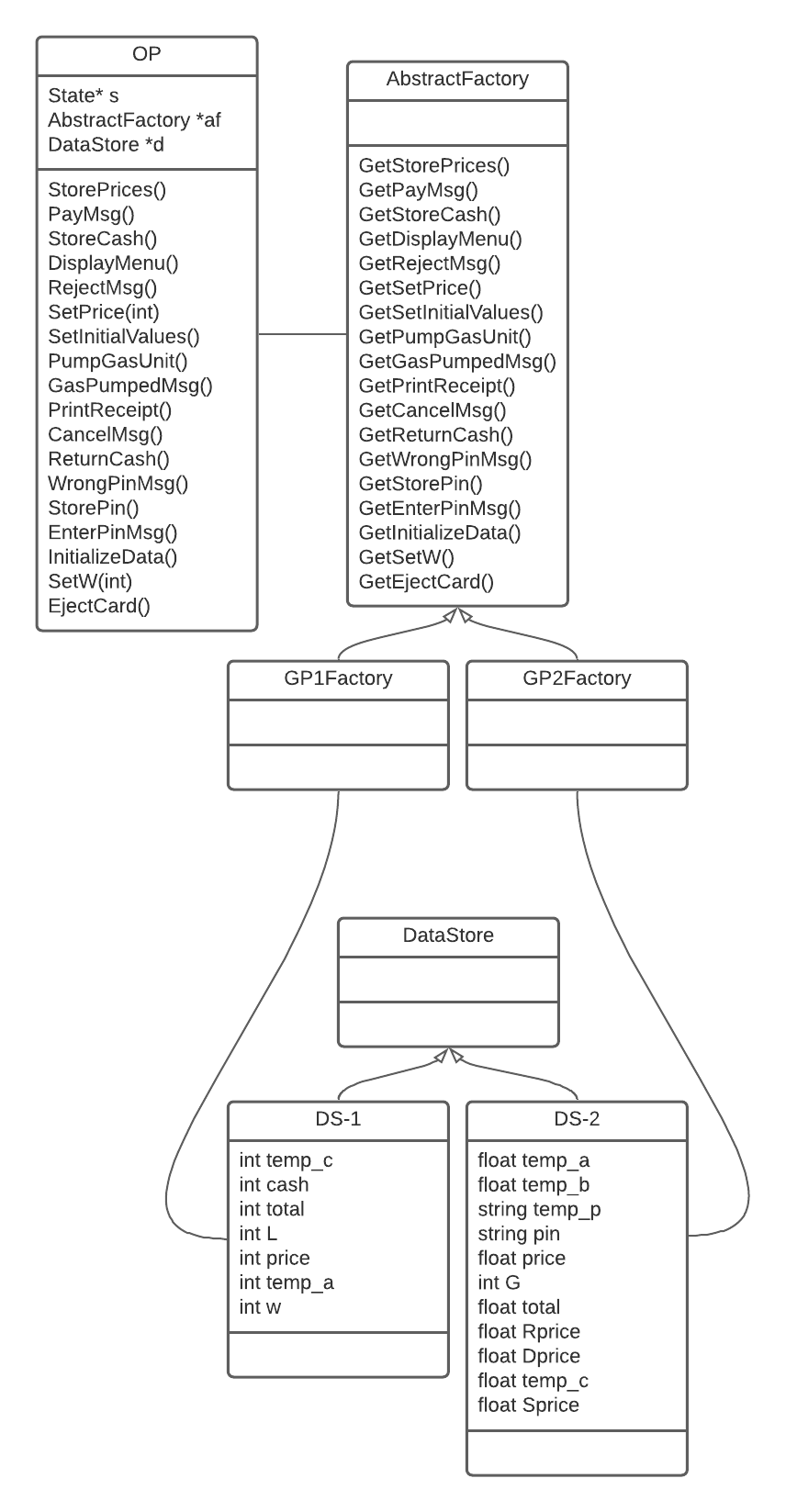
# Class Diagrams

## State pattern

It follows a centralized version.  


## Strategy pattern (continued)

## Abstract factory pattern

  
(connections to strategy methods are shown on the next page)



# Class Specifications

## Class Responsibilities

|  |  |
| --- | --- |
| **Class** | **Responsibility** |
| GP-1 | Input processor (Gas Pump 1) |
| GP-2 | Input processor (Gas Pump 2) |
| MDA-EFSM | It accepts meta-events (to be implemented by different states). It serves as the context class. |
| State | It has all the abstract methods for meta-events. |
| Start | Initial state (LS[0]) |
| S0 | Classes (or states) that implement all the abstract methods from *State* class.  (S0 🡪 LS[1], S1 🡪 LS[2], S2 🡪 LS[3], S3 🡪 LS[4], S5 🡪 LS[5],   S8 🡪 LS[6]) |
| S1 |
| S2 |
| S3 |
| S5 |
| S8 |
| OP | Output processor. It executes actions invoked by MDA-EFSM. It gets the object from concrete factories (GP1Factory and GP2Factory) and executes the invoked method. |
| StorePrices | It is the “interface” or abstract strategy class for method StorePrices. |
| StorePrices1 | It is the strategy class for GP1 (Gas Pump 1). |
| StorePrices2 | It is the strategy class for GP2 (Gas Pump 2). |
| PayMsg | It is the “interface” or abstract strategy class for method PayMsg. |
| PayMsg1 | It is the strategy class for GP1 (Gas Pump 1). |
| PayMsg2 | It is the strategy class for GP2 (Gas Pump 2). |
| StoreCash | It is the “interface” or abstract strategy class for method StoreCash. |
| StoreCash1 | It is the strategy class for GP1 (Gas Pump 1). |
| StoreCash2 | It is the strategy class for GP2 (Gas Pump 2). |
| DisplayMenu | It is the “interface” or abstract strategy class for method DisplayMenu. |
| DisplayMenu1 | It is the strategy class for GP1 (Gas Pump 1). |
| DisplayMenu2 | It is the strategy class for GP2 (Gas Pump 2). |
| SetPrice | It is the “interface” or abstract strategy class for method SetPrice. |
| SetPrice1 | It is the strategy class for GP1 (Gas Pump 1). |
| SetPrice2 | It is the strategy class for GP2 (Gas Pump 2). |
| RejectMsg | It is the “interface” or abstract strategy class for method RejectMsg. |
| RejectMsg1 | It is the strategy class for GP1 (Gas Pump 1). |
| RejectMsg2 | It is the strategy class for GP2 (Gas Pump 2). |
| SetInitialValues | It is the “interface” or abstract strategy class for method SetInitialValues. |
| SetInitialValues1 | It is the strategy class for GP1 (Gas Pump 1). |
| SetInitialValues2 | It is the strategy class for GP2 (Gas Pump 2). |
| PumpGasUnit | It is the “interface” or abstract strategy class for method PumpGasUnit. |
| PumpGasUnit1 | It is the strategy class for GP1 (Gas Pump 1). |
| PumpGasUnit2 | It is the strategy class for GP2 (Gas Pump 2). |
| GasPumpedMsg | It is the “interface” or abstract strategy class for method GasPumpedMsg. |
| GasPumpedMsg1 | It is the strategy class for GP1 (Gas Pump 1). |
| GasPumpedMsg2 | It is the strategy class for GP2 (Gas Pump 2). |
| PrintReceipt | It is the “interface” or abstract strategy class for method PrintReceipt. |
| PrintReceipt1 | It is the strategy class for GP1 (Gas Pump 1). |
| PrintReceipt2 | It is the strategy class for GP2 (Gas Pump 2). |
| CancelMsg | It is the “interface” or abstract strategy class for method CancelMsg. |
| CancelMsg1 | It is the strategy class for GP1 (Gas Pump 1). |
| CancelMsg2 | It is the strategy class for GP2 (Gas Pump 2). |
| ReturnCash | It is the “interface” or abstract strategy class for method ReturnCash. |
| ReturnCash1 | It is the strategy class for GP1 (Gas Pump 1). |
| ReturnCash2 | It is the strategy class for GP2 (Gas Pump 2). |
| WrongPinMsg | It is the “interface” or abstract strategy class for method WrongPinMsg. |
| WrongPinMsg1 | It is the strategy class for GP1 (Gas Pump 1). |
| WrongPinMsg2 | It is the strategy class for GP2 (Gas Pump 2). |
| StorePin | It is the “interface” or abstract strategy class for method StorePin. |
| StorePin1 | It is the strategy class for GP1 (Gas Pump 1). |
| StorePin2 | It is the strategy class for GP2 (Gas Pump 2). |
| EnterPinMsg | It is the “interface” or abstract strategy class for method EnterPinMsg. |
| EnterPinMsg1 | It is the strategy class for GP1 (Gas Pump 1). |
| EnterPinMsg2 | It is the strategy class for GP2 (Gas Pump 2). |
| InitializeData | It is the “interface” or abstract strategy class for method InitializeData. |
| InitializeData1 | It is the strategy class for GP1 (Gas Pump 1). |
| InitializeData2 | It is the strategy class for GP2 (Gas Pump 2). |
| SetW | It is the “interface” or abstract strategy class for method SetW. |
| SetW1 | It is the strategy class for GP1 (Gas Pump 1). |
| SetW2 | It is the strategy class for GP2 (Gas Pump 2). |
| EjectCard | It is the “interface” or abstract strategy class for method EjectCard. |
| EjectCard1 | It is the strategy class for GP1 (Gas Pump 1). |
| EjectCard2 | It is the strategy class for GP2 (Gas Pump 2). |
| AbstractFactory | It is the abstract factory class for OP. |
| GP1Factory | It is the concrete factory class for GP1 (Gas Pump 1). |
| GP2Factory | It is the concrete factory class for GP2 (Gas Pump 2). |
| DataStore | It holds the abstract DataStore class. |
| DS-1 | It holds the concrete data for GP1 (Gas Pump 1). |
| DS-2 | It holds the concrete data for GP2 (Gas Pump 2). |

## Class Operation Responsibilities

**Class GP1:**This class has the input processor operations.

Activate(int a) {

if(a>0) {

d->temp\_a = a;

m->Activate();

}

}

Start {

m->Start();

}

PayCash(int c) {

if(c>0) {

d->temp\_c = c;

m->PayCash();

}

}

PayCredit() {

m->PayCredit();

}

Reject() {

m->Reject();

}

Approved() {

m->Approved();

}

Cancel() {

m->Cancel();

}

StartPump() {

m->StartPump();

}

PumpLiter() {

if (d->w == 1) m ->Pump()

else if (d->cash>0) && (d->cash < d->price\*(d->L+1))

m->StopPump();

else

m->Pump();

}

StopPump() {

m->StopPump();

}  
  
**Class GP2:**It has all the methods for input processor GP2.

Activate(float a, float b, float ) {

if((a>0) && (b>0) && (c>0)) {

d->temp\_a = a;  
 d->temp\_b = b;  
 d->temp\_c = c;  
 m->Activate()  
 }

}

Start() {

m->Start();

}

PayCredit() {

m->PayCredit();

}

Reject() {

m->Reject();

}

PayDebit(string p) {

d->temp\_p=p;

m->PayDebit();

}

Pin(string x) {

if (d->pin==x) m->CorrectPin()

else m->InCorrectPin(1);

}

Cancel() {

m->Cancel();

}  
   
Approved() {

m->Approved();

}

Diesel() {

m->SelectGas(3)

}

Regular() {

m->SelectGas(1)

}

Super() {

m->SelectGas(2)

}

StartPump() {

if (d->price>0) m->StartPump();

}

PumpGallon() {

m->Pump();

}

StopPump() {

m->StopPump();

}

FullTank() {

m->StopPump();

}

**Class MDA-EFSM:** 🡪 Context class

public MDA\_EFSM(OP p) {

// initlalize all the states

LS = new State[7];

LS[0] = new Start(this, p);

LS[1] = new S0(this, p);

LS[2] = new S1(this, p);

LS[3] = new S2(this, p);

LS[4] = new S3(this, p);

LS[5] = new S5(this, p);

LS[6] = new S8(this, p);

s = LS[0]; // starting point

k = 0; // pin attempt count

}

// meta-events to be implement by states

public void Activate() { s.Activate(); }

public void Start() { s.Start(); }

public void Reject() { s.Reject(); }

public void Cancel() { s.Cancel(); }

public void Approved() { s.Approved(); }

public void StartPump() { s.StartPump(); }

public void Pump() { s.Pump(); }

public void StopPump() { s.StopPump(); }

public void SelectGas(int g) { s.SelectGas(g); }

public void CorrectPin() { s.CorrectPin(); }

public void IncorrectPin(int max) { s.IncorrectPin(max); }

public void PayCash() { s.PayCash(); }

public void PayCredit() { s.PayCredit(); }

public void PayDebit() { s.PayDebit(); }  
}

**Class State:**All abstract methods to be implemented by concrete states.  
  
**Class Start:**Activate() {

if (m.s == m.LS[0]) {

p.StorePrices();

m.s = m.LS[1];

}

}

**Class S0:**

Start() {

if (m.s == m.LS[1]) {

p.PayMsg();

p.InitializeData();

p.SetW(1);

m.s = m.LS[2];

System.***out***.println("Current State: S1");

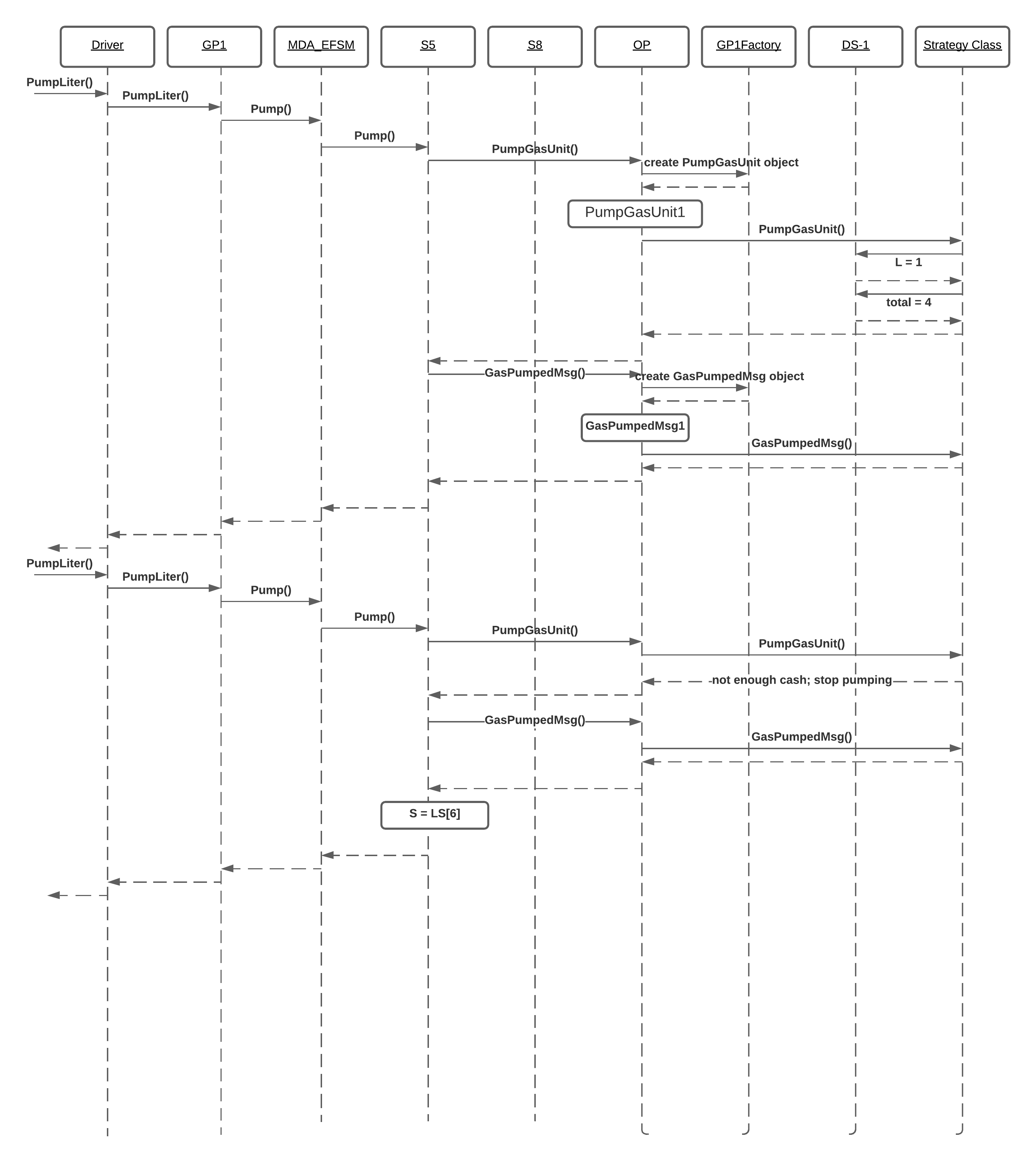
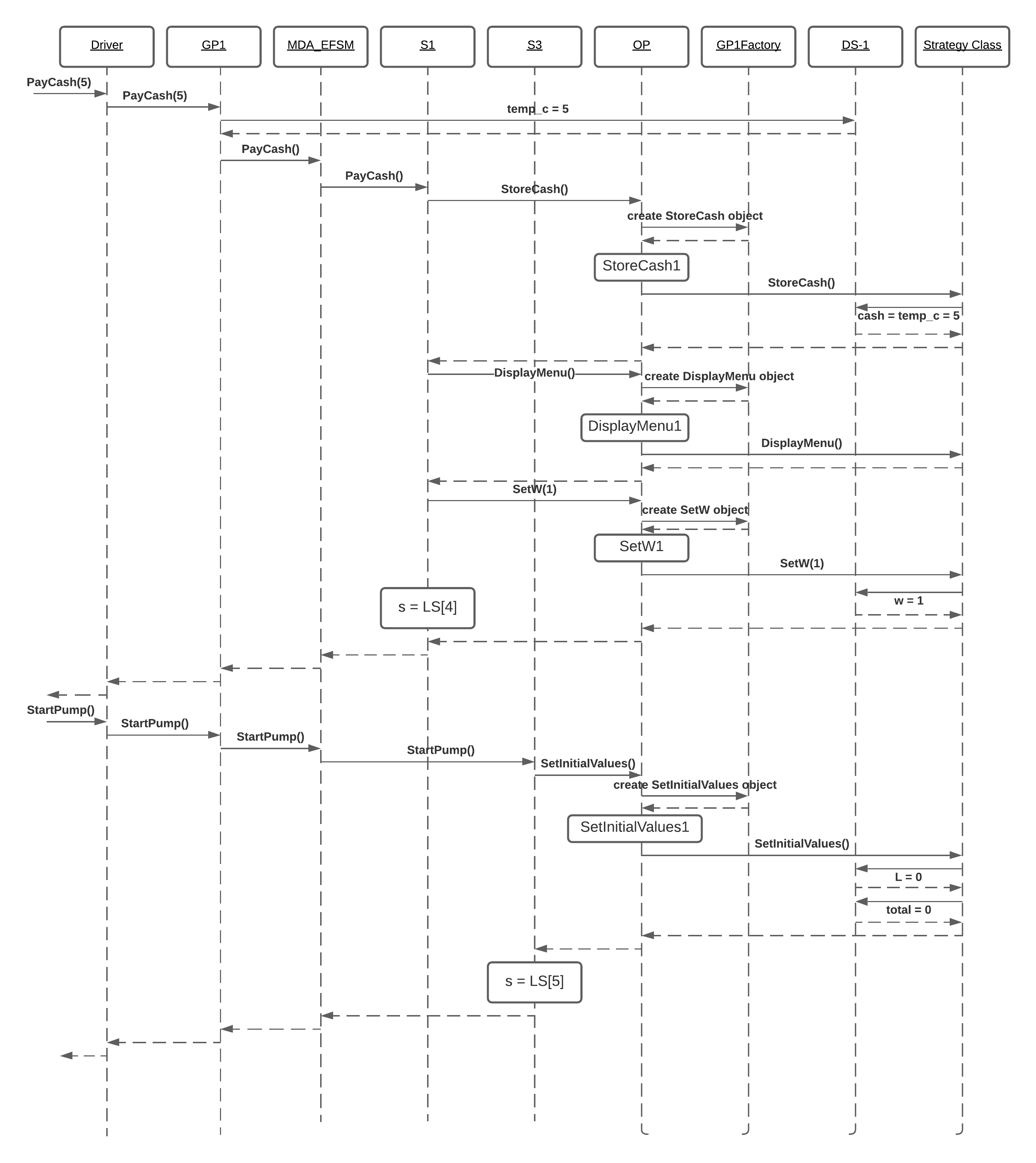
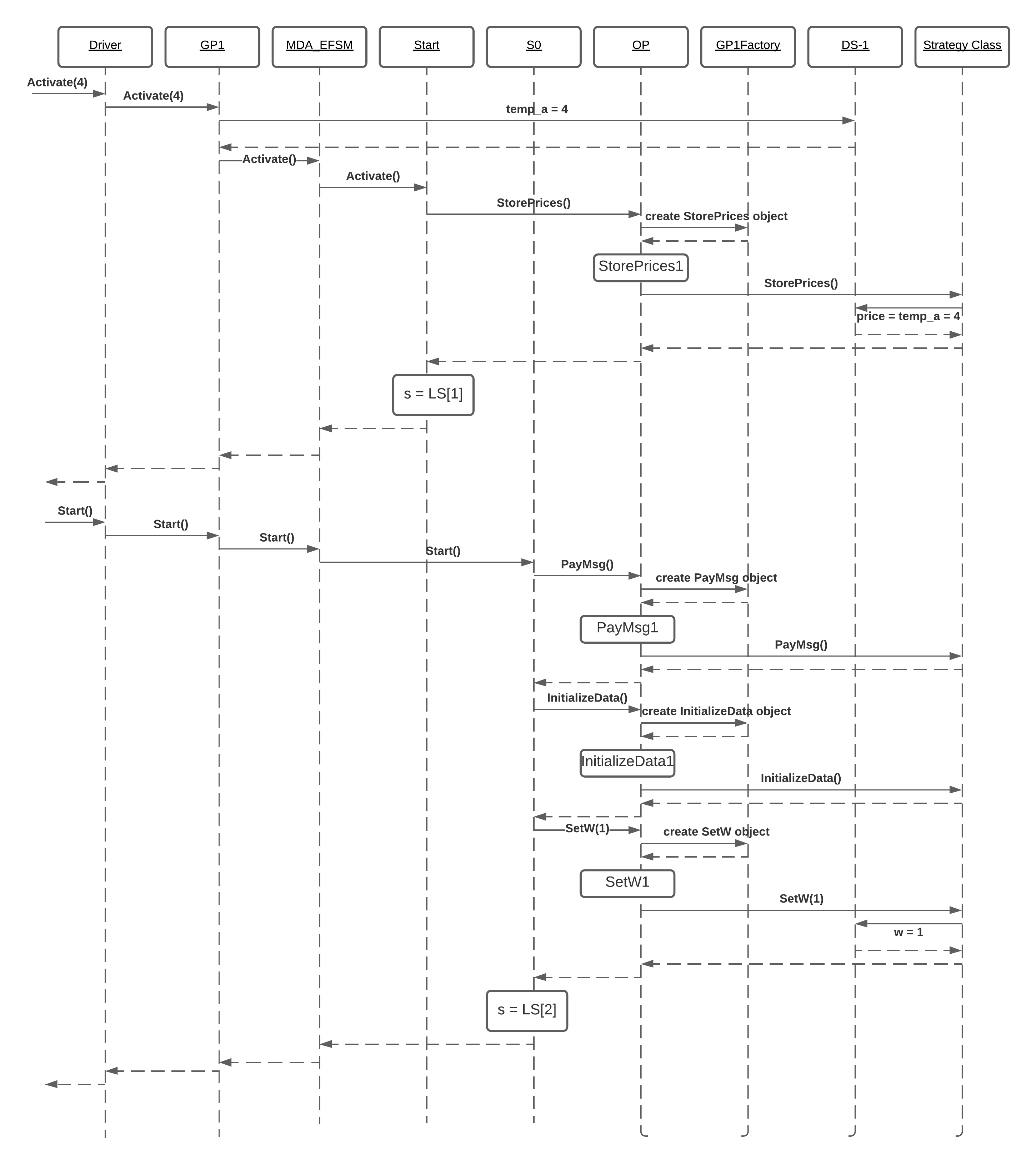
}

}

}

# Dynamics

## Scenario I (GP1)

Activate(4), Start(), PayCash(5), StartPump(), PumpLiter(), PumpLiter() 

## Scenario II

Activate(4.2, 7.2, 5.3), Start(), PayDebit(“abc”), Pin(“cba”), Pin(“abc”), Super(), StartPump(), PumpGallon(), FullTank() 