Xiao Li
A20370639

PROJECT DESIGN and REPORT

CS 586; Spring 2017

Final Project Deadline: **April 26, 2017**
Late submissions: 50% off
After **May 1** the final project will not be accepted.

This is an **individual** project not a team project.

The hardcopy of the project must be submitted. Electronic submissions are not acceptable. Notice that the Blackboard project submissions are only considered as a proof of submission on time (before the deadline). If the hardcopy of the project is different than the electronic version submitted on the Blackboard, then **50%** penalty will be applied. If the project assignment is submitted on the Blackboard on time, we must receive the hardcopy of the project by noon on **Friday, April 28**. If the hardcopy is received after this deadline, **20%** penalty will be applied.

DESIGN and IMPLEMENTATION

The goal of the second part of the project is to design two *GasPump* components using the Model-Driven Architecture (MDA) and then implement these *GasPump* components based on this design using the OO programming language. This OO-oriented design should be based on the MDA-EFSM for both *GasPump* components that was identified in the first part of the project. You may use your own MDA-EFSM (assuming that it was correct) or you can use the posted sample MDA-EFSM. In your design, you **MUST** use the following OO design patterns:

- state pattern
- strategy pattern
- abstract factory pattern

In the design, you need to provide the class diagram, in which the coupling between components should be minimized and the cohesion of components should be maximized (components with high cohesion and low coupling between components). In addition, two sequence diagrams should be provided as described on the next page (Chapter 4 of the report).

After the design is completed, you need to implement the *GasPump* components based on your design using the OO programming language. In addition, the driver for the project to execute and test the correctness of the design and its implementation for the *GasPump* components must be implemented.

Outline of the Report & Deliverables

1. MDA-EFSM model for the *GasPump* components
   i. A list of meta events for the MDA-EFSM
   ii. A list of meta actions for the MDA-EFSM with their descriptions
   iii. A state diagram of the MDA-EFSM
   iv. Pseudo-code of all operations of Input Processors of *GasPump-1* and *GasPump-2*

2. Class diagram(s) of the MDA of the *GasPump* components. In your design, you **MUST** use the following OO design patterns:
   i. State pattern
   ii. Strategy pattern
   iii. Abstract factory pattern

3. For each class in the class diagram(s) you should:
   a. Describe the purpose of the class, i.e., responsibilities.
   b. Describe the responsibility of each operation supported by each class.

4. Dynamics. Provide two sequence diagrams for two Scenarios:
   a. Scenario-I should show how one gallon of Regular gas is disposed in *GasPump-1*, i.e., the following sequence of operations is issued: *Activate(3.1, 4.3), Start(), PayCredit(), Approved(), Regular(), StartPump(), PumpGallon(), StopPump()*

   b. Scenario-II 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), Start(), PayCash(6), Premium(), StartPump(), PumpLiter(), PumpLiter(), NoReceipt()*

5. Source-code and patterns
   In this part of the report you should clearly indicate/highlight which parts of the source code are responsible for the implementation of the three required design patterns **(if this is not clearly indicated in the source code, 20 points will be deducted)**:
   - state pattern
   - strategy pattern
   - abstract factory pattern.

6. Well documented (commented) source code. ~~**Printed hardcopy**~~ **of the source code is required and should be a part of the report.** Otherwise, **15 POINTS** will be automatically deducted from the project.

IMPORTANT: The project executable(s) of the *GasPump* components with detailed instructions explaining the execution of the program must be prepared by students and made available for grading. The best way is to provide the project executable on a flash drive (or CD). However, you may also submit the project executable on the Blackboard. If the executable is not provided (or not easily available), **20 POINTS** will be automatically deducted from the project grade. The flash drive (or CD) should also contain the source code of your implementation. Note that the source code may be compiled during the grading and then executed.

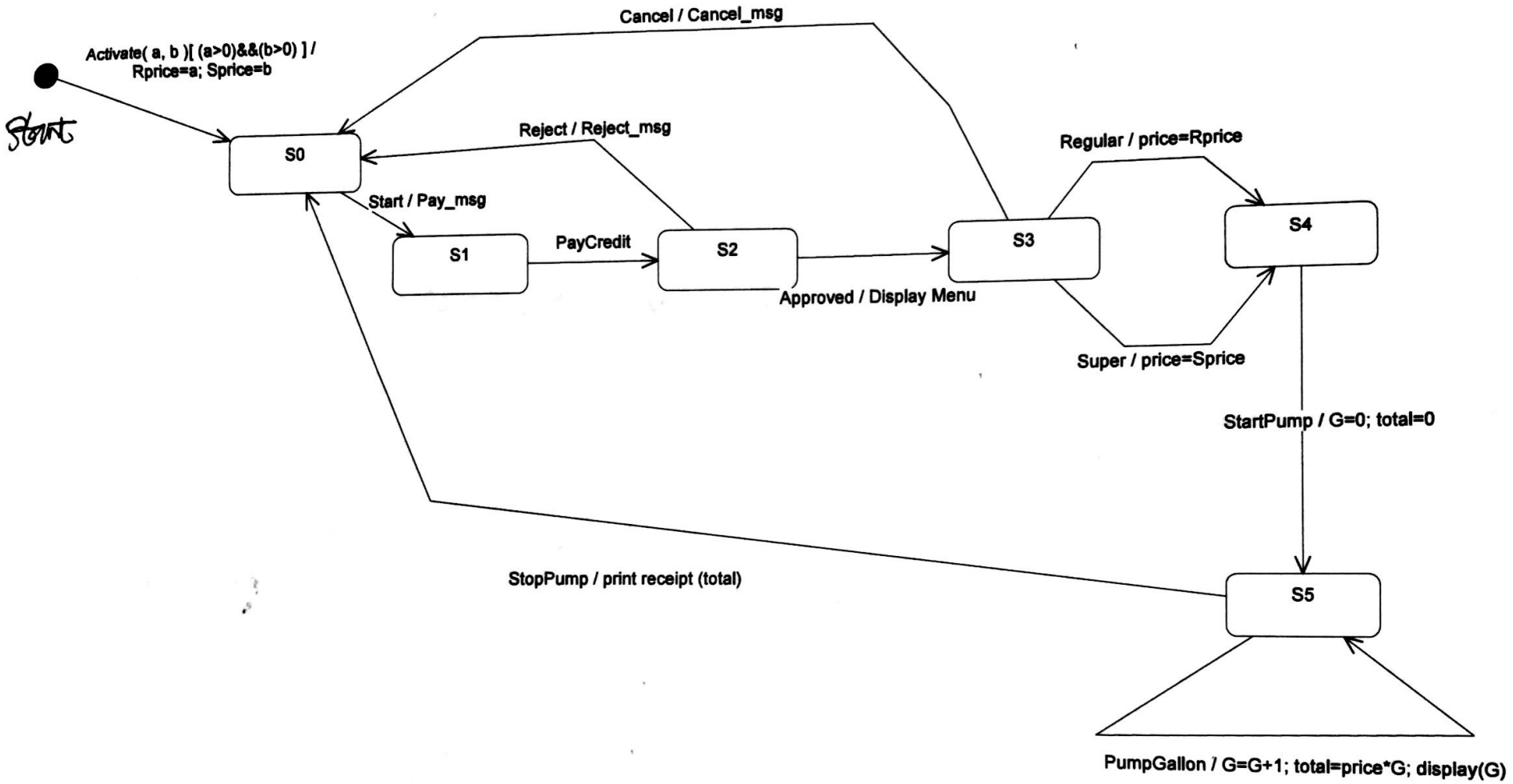


Figure 1: EFSM for GasPump-1

stateBehavior use EFSM

Difference

1. Operation Names / Signatures
2. types of payments
3. types of gas disposed
4. display menu
5. No receipts.

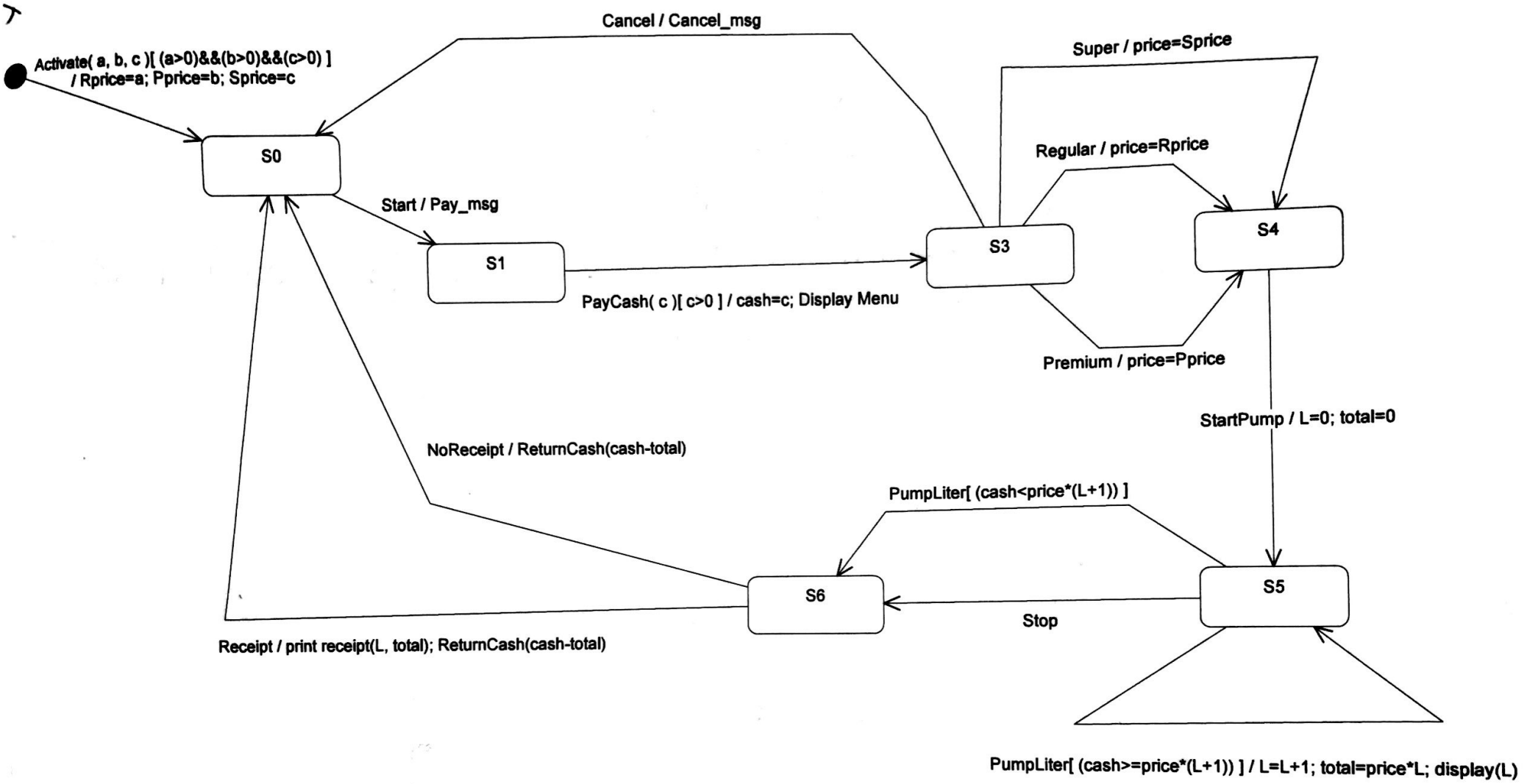


Figure 2: EFSM for GasPump-2

1. MDA-EFSM model for the GasPump components.

(1) A list of meta events for MDA-EFSM

Activate()
Start()
PayType(int t)
Reject()
Cancel()
Approved()

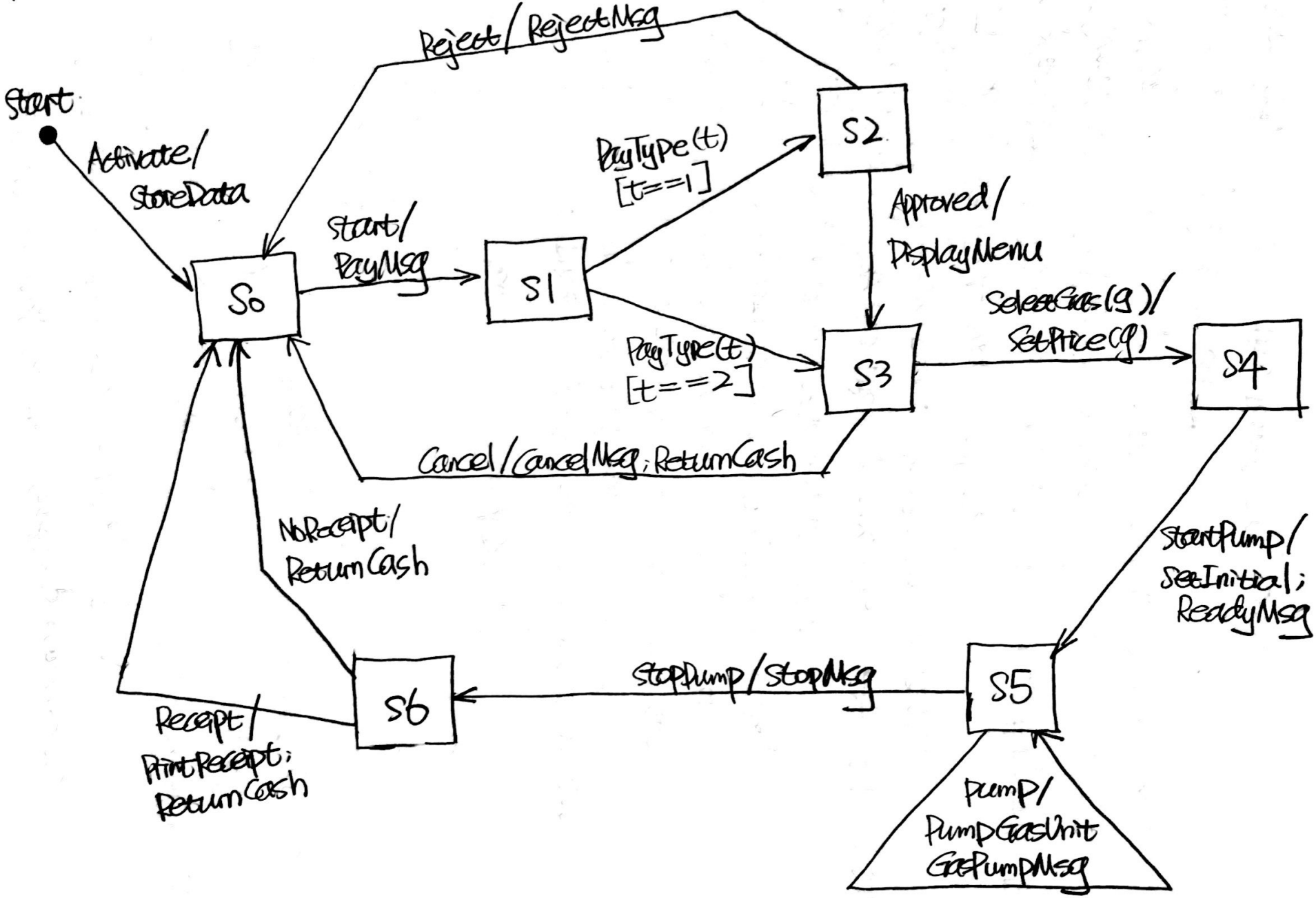
StartPump()
Pump()
StopPump()

SelectGas(int g)
Receipt()
NoReceipt()

(2) A list of meta actions for MDA-EFSM

StoreData // store price for gas from temporary data store
PayMsg // Display a type of payment method
StoreCash // Store cash from the temporary data store
DisplayMenu // Display a menu with a list of selections
RejectMsg // Display credit card not approved message
CancelMsg // Display a cancellation message
SetPrice(int g) // set the price for the gas identified by g identifier
ReadyMsg // Display the ready for pumping message
SetInitial // set G/L and total to 0
PumpGasUnit // Dispose unit of gas and counts # of units disposed
GasPumpMsg // Display the amount of disposed gas
StopMsg // Stop pump message and receipt message
PrintReceipt // Print a receipt
ReturnCash // Return the remainning cash.

(3) State Diagram of the NDA-EFSM



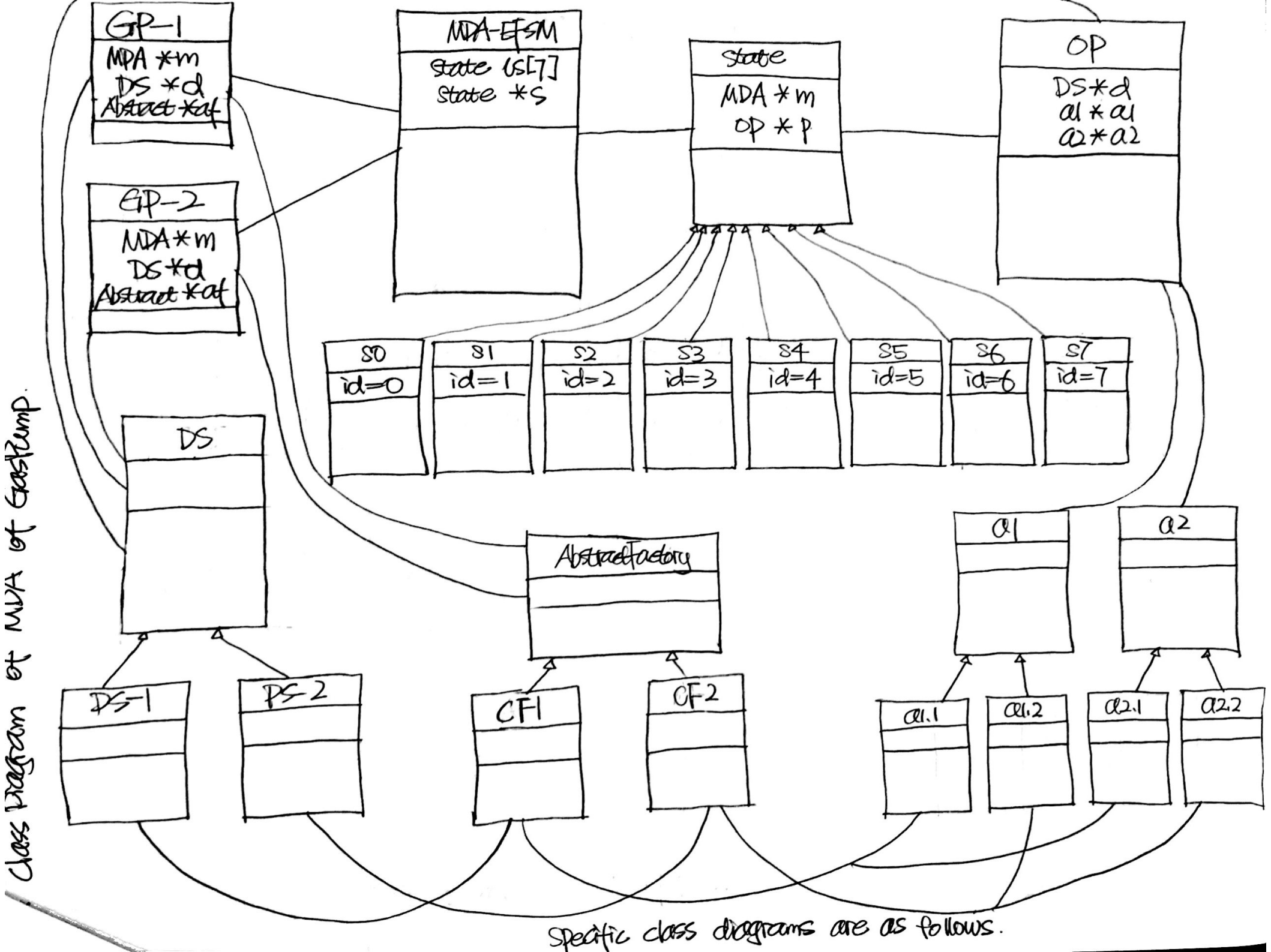
4) pseudo code of all operation of Input

GasPump-1

```
Activate(float a, float b)
{   if(a>0)&&(b>0)
        d->temp_a=a
        d->temp_b=b
        m->Activate()
}

Start(){
    m->start(); }

PayCredit(){
    m->PayType(1);}

Reject(){
    m->Reject(); }

Cancel(){
    m->cancel(); }

Approved(){
    m->Approved(); }

Super(){
    m->SelectGas(2); }

Regular(){
    m->SelectGas(1); }

StartPump(){
    m->StartPump(); }

PumpGallon(){
    m->Pmup(); }

StopPump(){
    m->StopPump();
    m->Receipt(); }
```

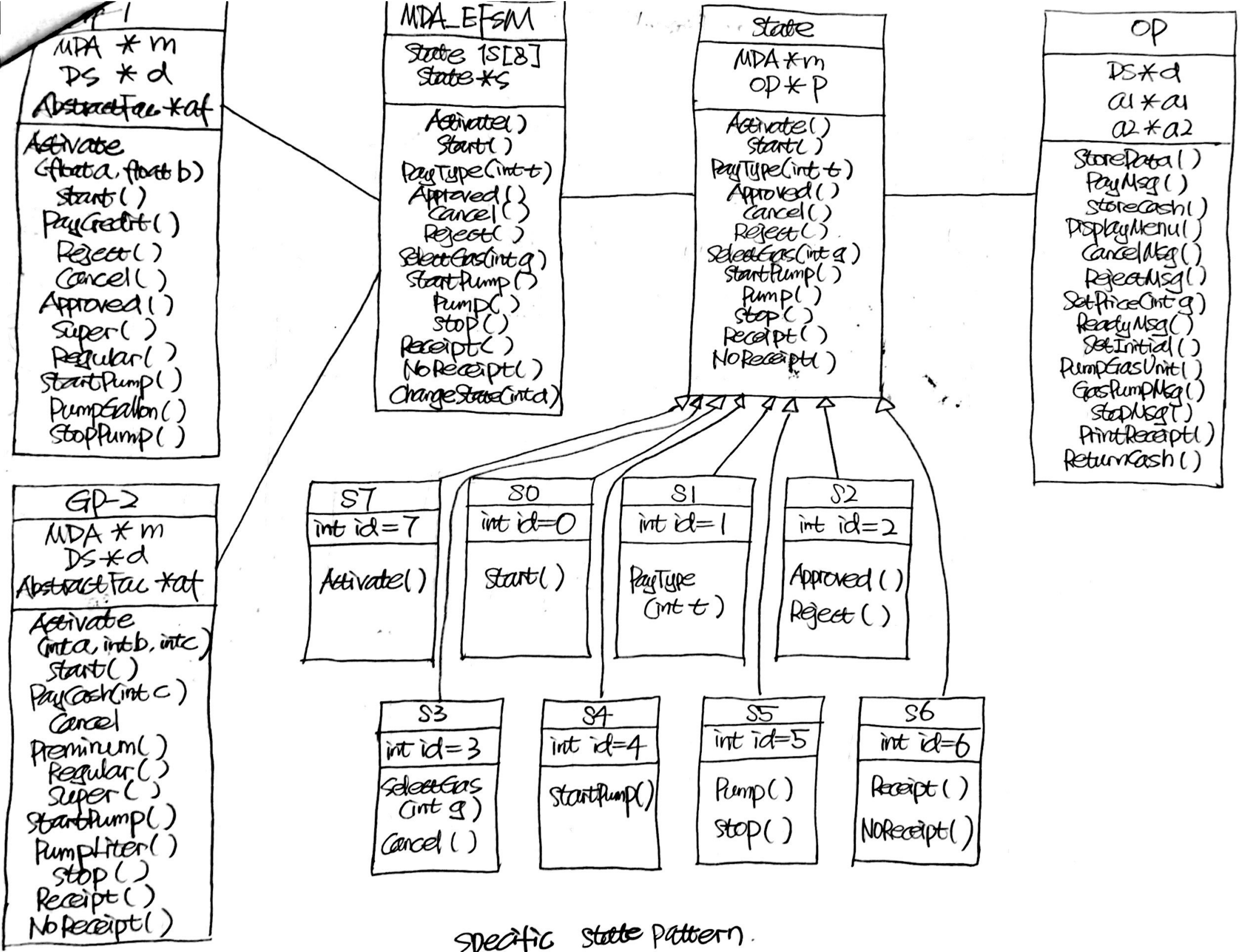
GasPump-2

```
Activate(int a, int b, int c)
{ if((a>0)&&(b>0)&&(c>0)
      d->temp_a=a
      d->temp_b=b
      d->temp_c=c
      m->Activate1()
}

Start(){
   m->start(); }

PayCash(float c){
    if(c>0)
       d->temp_cash=c
       m->PayType(2)
}

Cancel(){
   m->cancel(); }

Super(){
   m->SelectGas(2); }

Premium(){
   m->SelectGas(3); }

Regular(){
   m->SelectGas(1); }

StartPump(){
   m->StartPump(); }

PumpLiter(){
   if(d->cash<(d->L+1)*d->price
       m->StopPump();
   else m->Pump();
}

Stop(){
   m->StopPump();

Recept(){
   m->Recept(); }

NoRecept(){
   m->NoRecept(); }
```

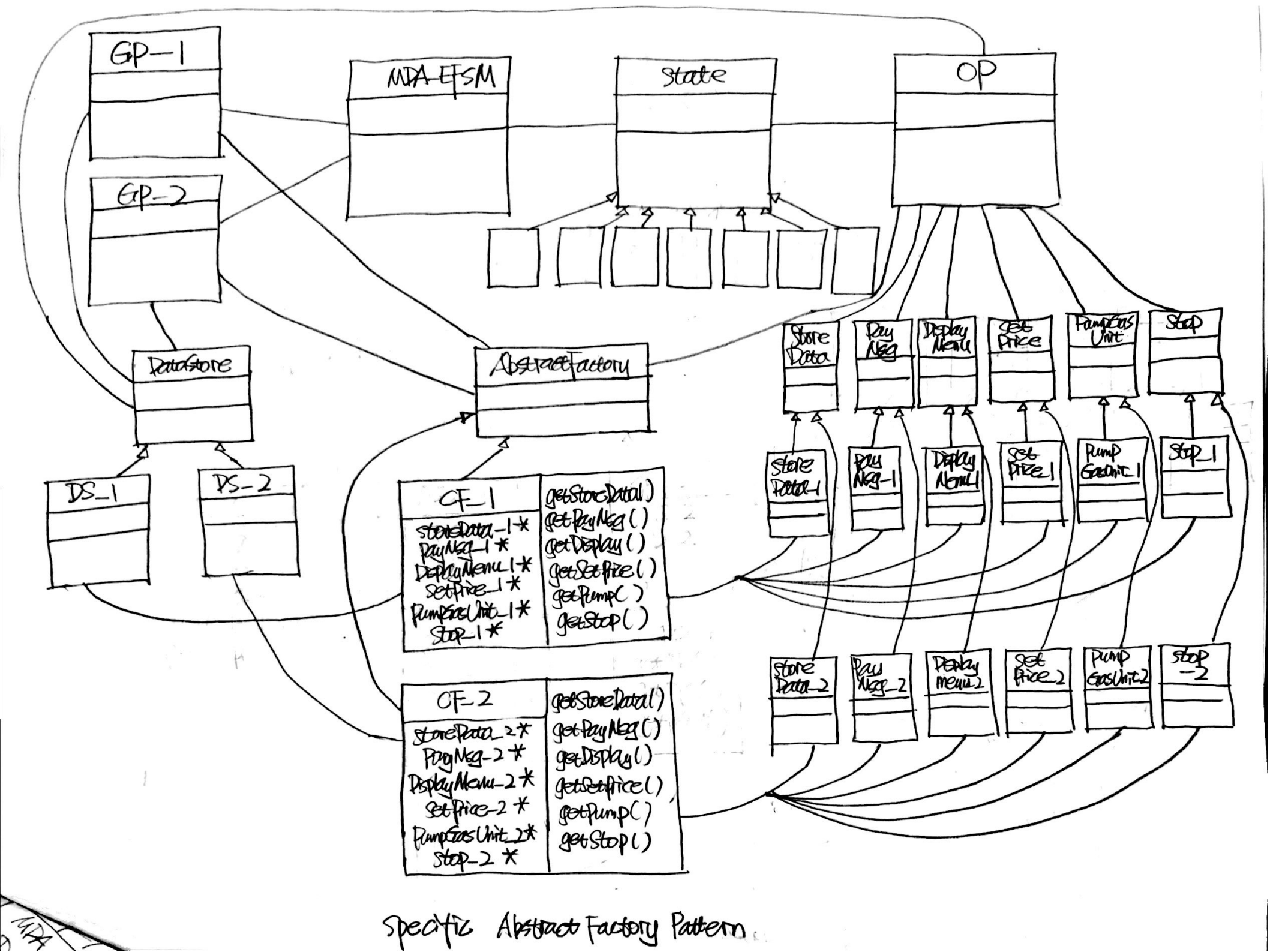


Class Diagram of MDA of Gastrump.

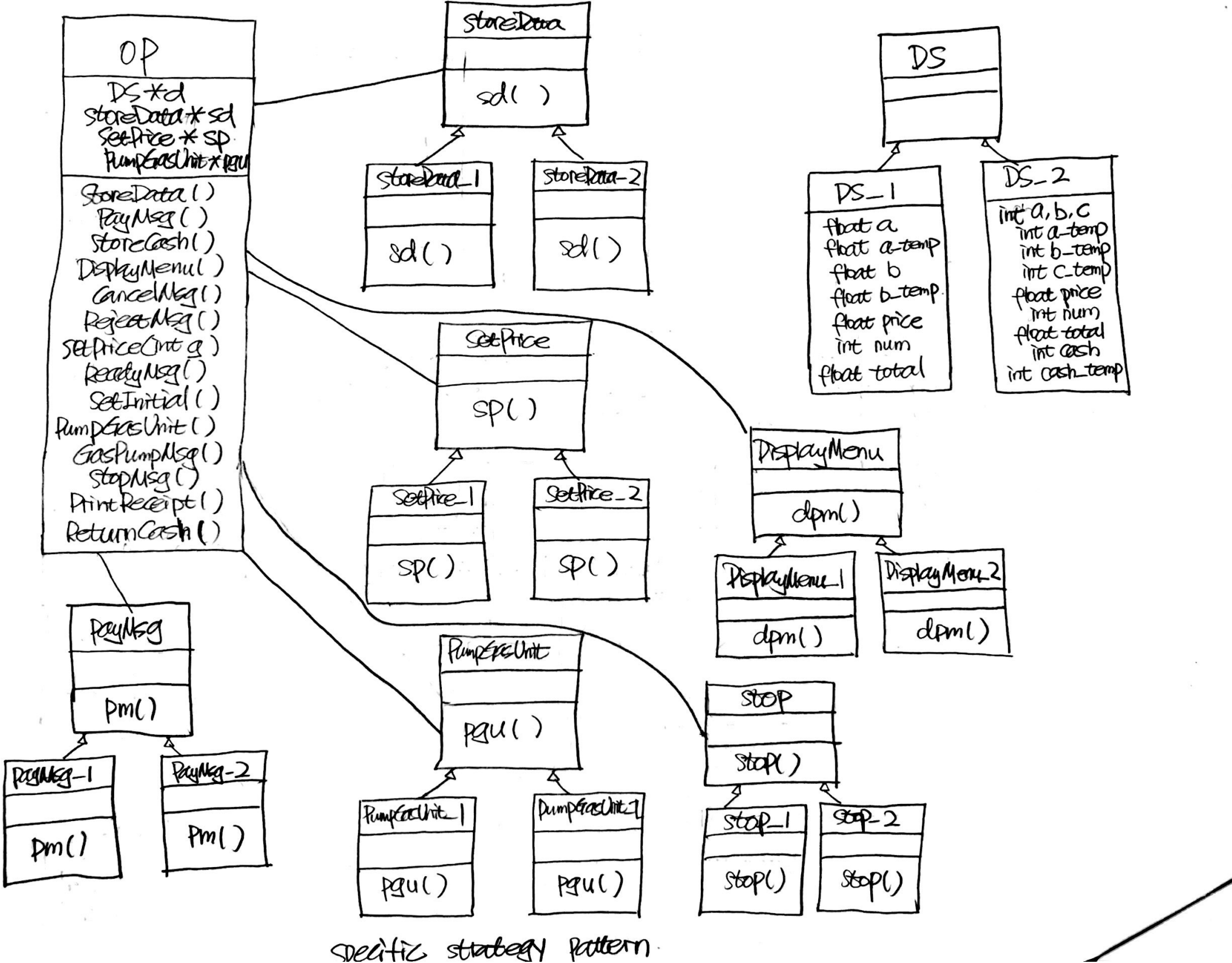
Specific class diagrams are as follows.



specific state pattern.



Specific Abstract Factory Pattern



specific strategy pattern

3. For each class and operations, Describe the responsibility

GP_1 # GasPump_1 contains all the functional operations of the gas pump system, creating the class will create corresponding classes it will use.

Activate(float a, float b) //provide the method to activate the gas pump system where the Regular gas price is a and Super gas price is b and store them temporary in the DataStore

Start() //provide the method to start the transaction

PayCredit() //provide the method to pay for gas by credit

Reject() //provide the method to reject the credit card

Cancel() //provide the method to cancel the transaction

Aprpoved() //provide the method to approve the credit card

Super() //provide the method to select the super gas

Regular() //provide the method to select the regular gas

StartPump() //provide the method to start pumping gas

PumpGallon() // provide the method to pump one gallon gas

StopPump() // provide the method to stop pumping gas

GP_2 # GasPump_2 contains all the functional operations of the gas pump system, creating the class will create corresponding classes it will use.

Activate(int a, int b, int c) //provide the method to activate the gas pump system where the Regular gas price is a, Premium gas price is b and Super gas price is c and store them temporary in Data Store

Start() // provide the method to start the transaction

PayCash(int c) // provide the method to pay for gas by cash and store it temporary in DS

Cancel() //provide the method to cancel the transaction

Premium() //provide the method to select the premium gas

Super() //provide the method to select the super gas

Regular() //provide the method to select the regular gas

StartPump() //provide the method to start pumping gas

PumpLiter() // provide the method to pump one Liter gas

Stop () //provide the method to stop pumping gas

Receipt() //provide the method to request the receipt

NoReceipt() //provide the method to request no receipt

MDA_EFSM #contains all the meta events for the MDA-EFSM model

Activate() //meta event to activate the gas pump system

Start() //meta event to start the transaction

PayType(int t) //meta event to pay for gas by credit if 1, and cash 2

Cancel() //meta event to cancel the transaction

Reject() //meta event to reject the credit card

SelectGas(int g) //meta event to select gas, regular 1, super 2, premium 3

StartPump() //meta event to start pumping gas

Pump() //meta event to pump gas
Stop () //meta event to stop pumping gas
Receipt() //meta event to request the receipt
NoReceipt() //meta event to request no receipt

State #contains all states in the state pattern, do the actions and change state.
S7 #the start state. Activate(), to store data and change state to S0
S0 #the S0 state. Start(), to start and change state to S1
S1 #the S1 state. PayType(int t), to change state to S2 if t=1, S3 if t=2
S2 #the S2 state. Approved(), to display menu and change state to S3
S3 #the S3 state. Selectgas(int g), to store gas price and change state to S4; Cancel(), to display cancel msg and change state to S0
S4 #the S4 state. StartPump(), to set initial data and change state to S5
S5 #the S5 state. Pump(), to pump unit gas; Stop(), to stop pumping and change state to S6
S6 #the S6 state. Receipt(), to print receipt, return cash and change state to S0

OP #contains all meta actions for the MDA-EFSM model
StoreData() //store price for gas from temporary data store
PayMsg() //display a type a payment method
StoreCash() //store cash from the temporary data store
DisplayMenu() //display a menu with a list of selections
RejectMsg() //display a credit card not approved message
CancelMsg() //display a cancelation message
SetPrice(int g) //select the price for the gas for the identified by a identifier
ReadyMsg() //display a ready message for pumping
SetPrice(int g) //select the price for the gas for the identified by a identifier
ReadyMsg() //display a ready message for pumping
SetInitial() //set initial volumn and total
PumpGasUnit() //dispose unit gas and counts
GasPumpMsg() //display the amount of disposed gas
StopMsg() //stop pump message and receipt message
Receipt() //print the receipt
ReturnCash() //return the remaining cash

StoreData #abstract class for the storedata method
StoreData_1 // contains the method for gas pump system 1
StoreData_2 // contains the method for gas pump system 2

PayMsg #abstract class for the display payment message method
PayMsg_1 // contains the method for gas pump system 1
PayMsg_2 // contains the method for gas pump system 2

SetPrice #abstract class for the SetPrice method
    SetPrice_1 // contains the method for gas pump system 1
    SetPrice_2 // contains the method for gas pump system 2

DisplayMenu #abstract class for the display menu method
    DisplayMenu_1 // contains the method for gas pump system 1
    DisplayMenu_2 // contains the method for gas pump system 2

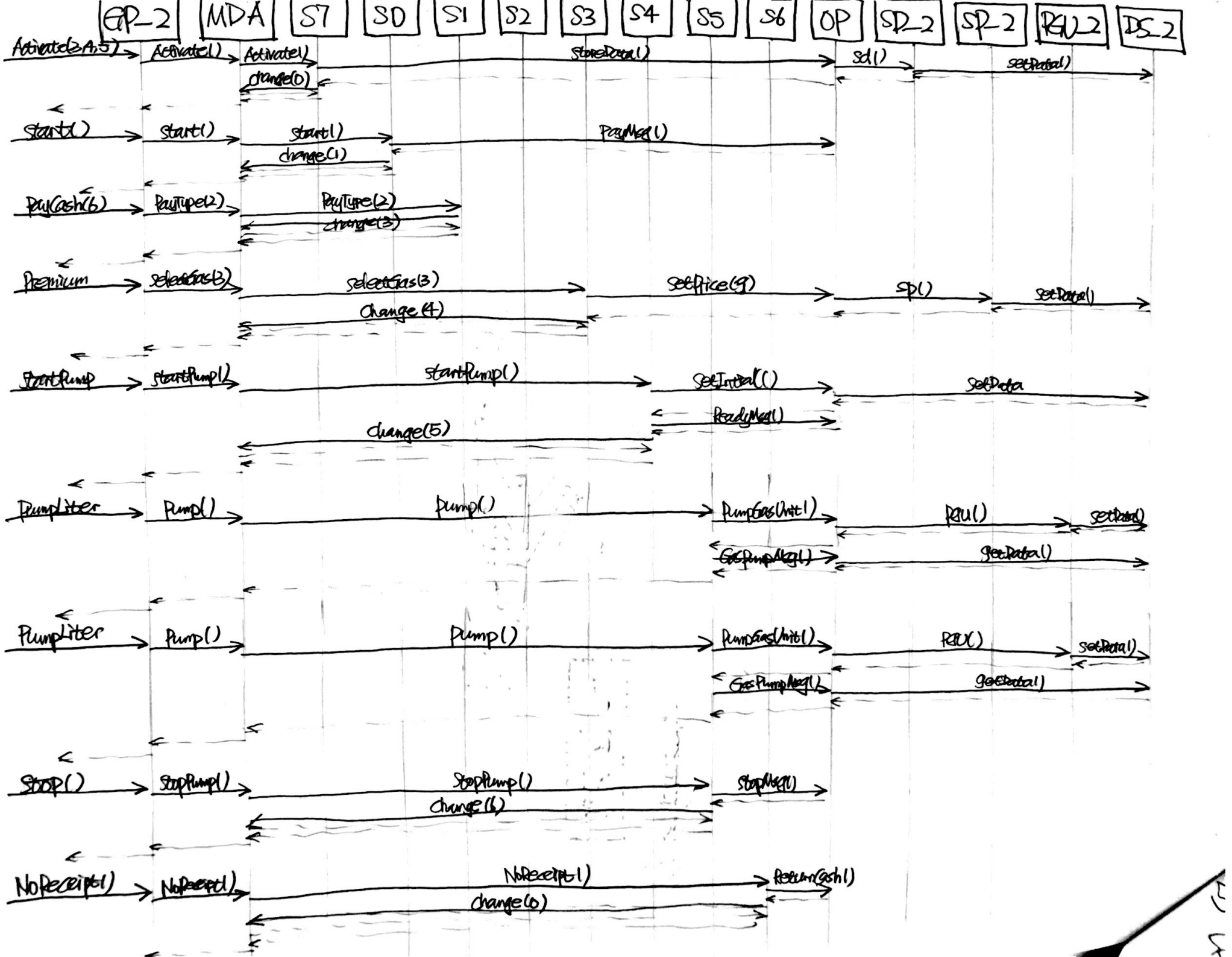
PumpGasUnit #abstract class for the PumpGasUnit method
    PumpGasUnit_1 // contains the method for gas pump system 1
    PumpGasUnit_2 // contains the method for gas pump system 2

Stop #abstract class for the stop method
    Stop_1 // contains the method for gas pump system 1
    Stop_2 // contains the method for gas pump system 2

DS #abstract class for the Data Store
    DS_1 // contains the data type for gas pump system 1
    DS_2 // contains the data type for gas pump system 2

AbstractFactory #abstract class for the abstract factory
    CF_1 // create all the corresponding classes for gas pump system 1
    CF_2 // create all the corresponding classes for gas pump system 2

Dynamics, Sequence Diagram

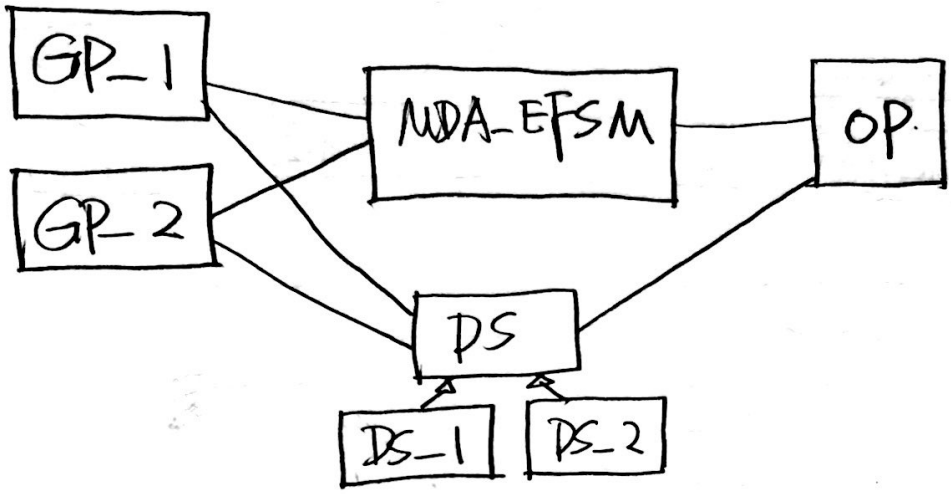


5. Source Code Patterns

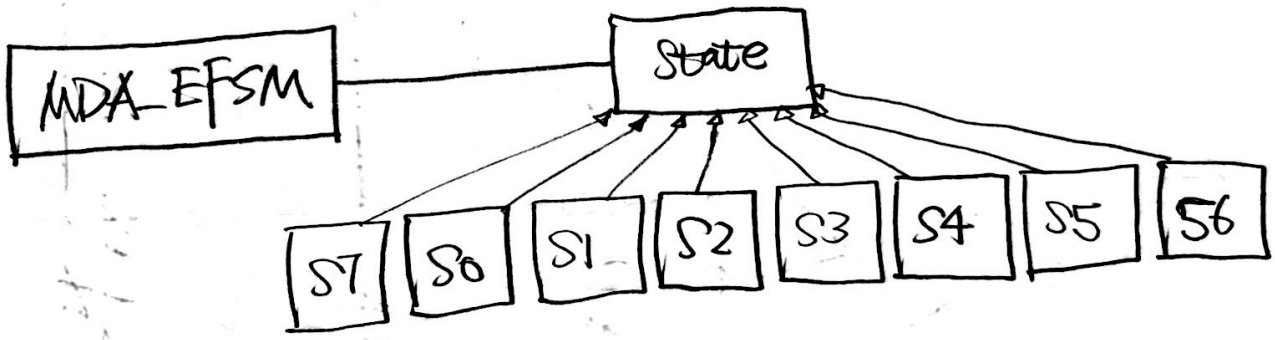
Source Code classes are as below and Indicate which part are responsible for implementation of three patterns.

Driver

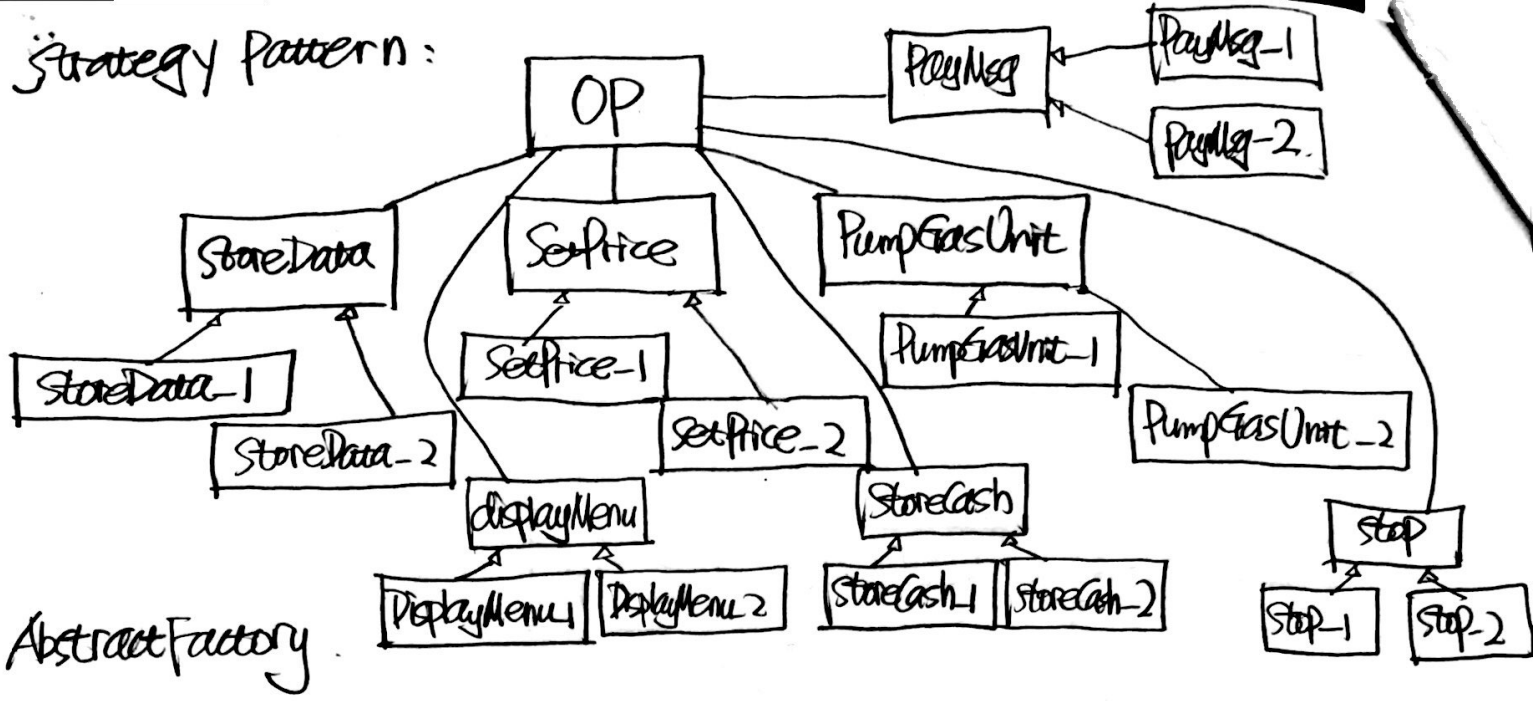
MDA_EFSM:



State Pattern:



∴ Strategy Pattern :



AbstractFactory