Gas Pump

Report

Sai Krishna Akula - A20396253

Table of Contents

Gas Pump Report

Sai Krishna Akula | A20396253 | Section 02

1. MDA-EFSM Model:

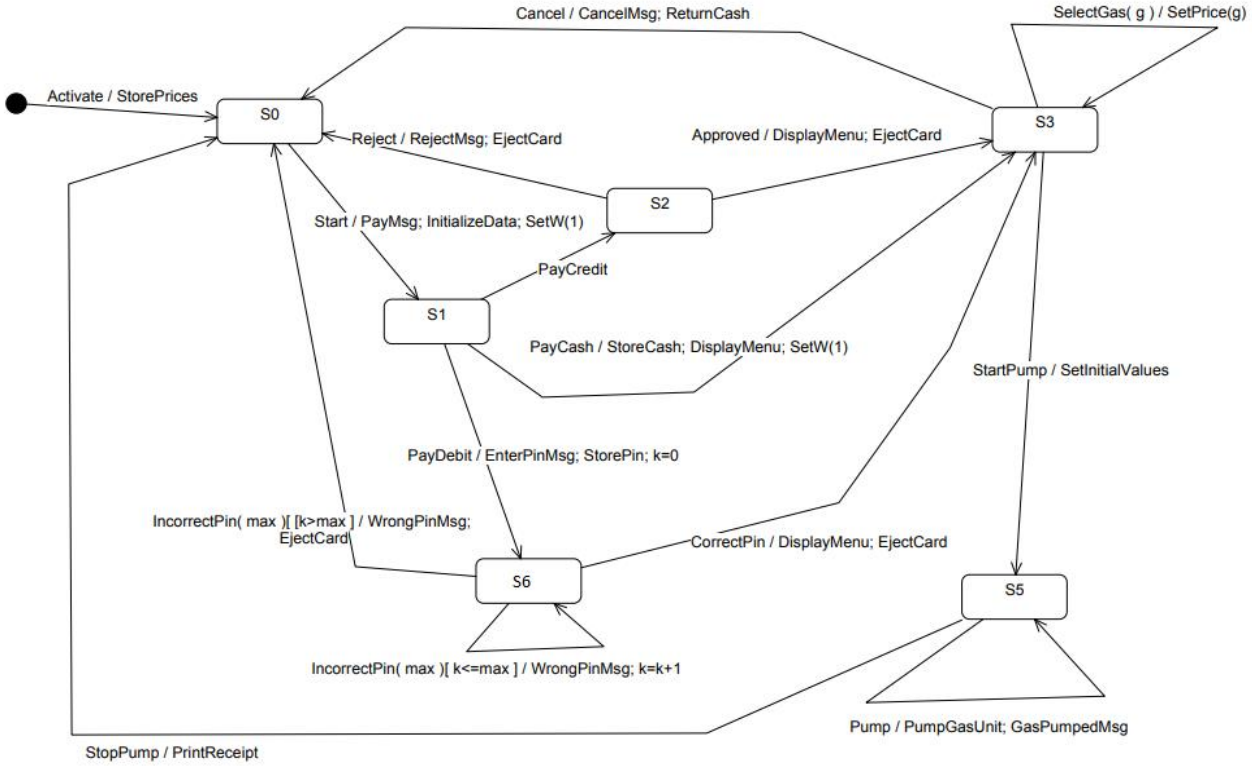
1.1. Meta Events:

- Activate()
- Start()
- PayCredit()
- PayCash()
- PayDebit()
- Reject()
- Cancel()
- Approved()
- StartPump()
- Pump()
- StopPump()
- SelectGas(int g) // Regular: g=1; Super: g=2; Diesel: g=3
- CorrectPin()
- IncorrectPin(int max)

1.2. Meta Actions:

- 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

1.3. State Diagram:



MDA-EFSM for Gas Pumps

1.4. Pseudo Code:

Operations of the Input Processor (GasPump-1)

```
Activate(int a) {
 if (a>0) {
 d->temp_a=a;
 m->Activate()
 }
}
Start() {
 m->Start();
}
PayCash(float 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 litters 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

Operations of the Input Processor (GasPump-2)

```
Activate(float a, float b) {
 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

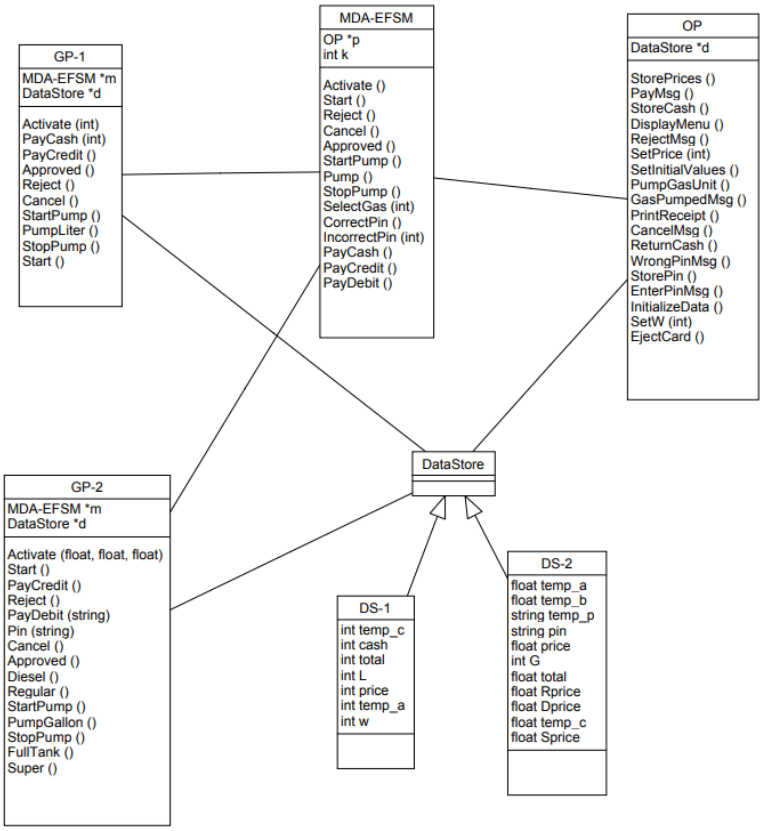
m: is a pointer to the MDA-EFSM object

d: is a pointer to the Data Store object

SelectGas(g): Regular: g=1; Super: g=2; Diesel: g=3

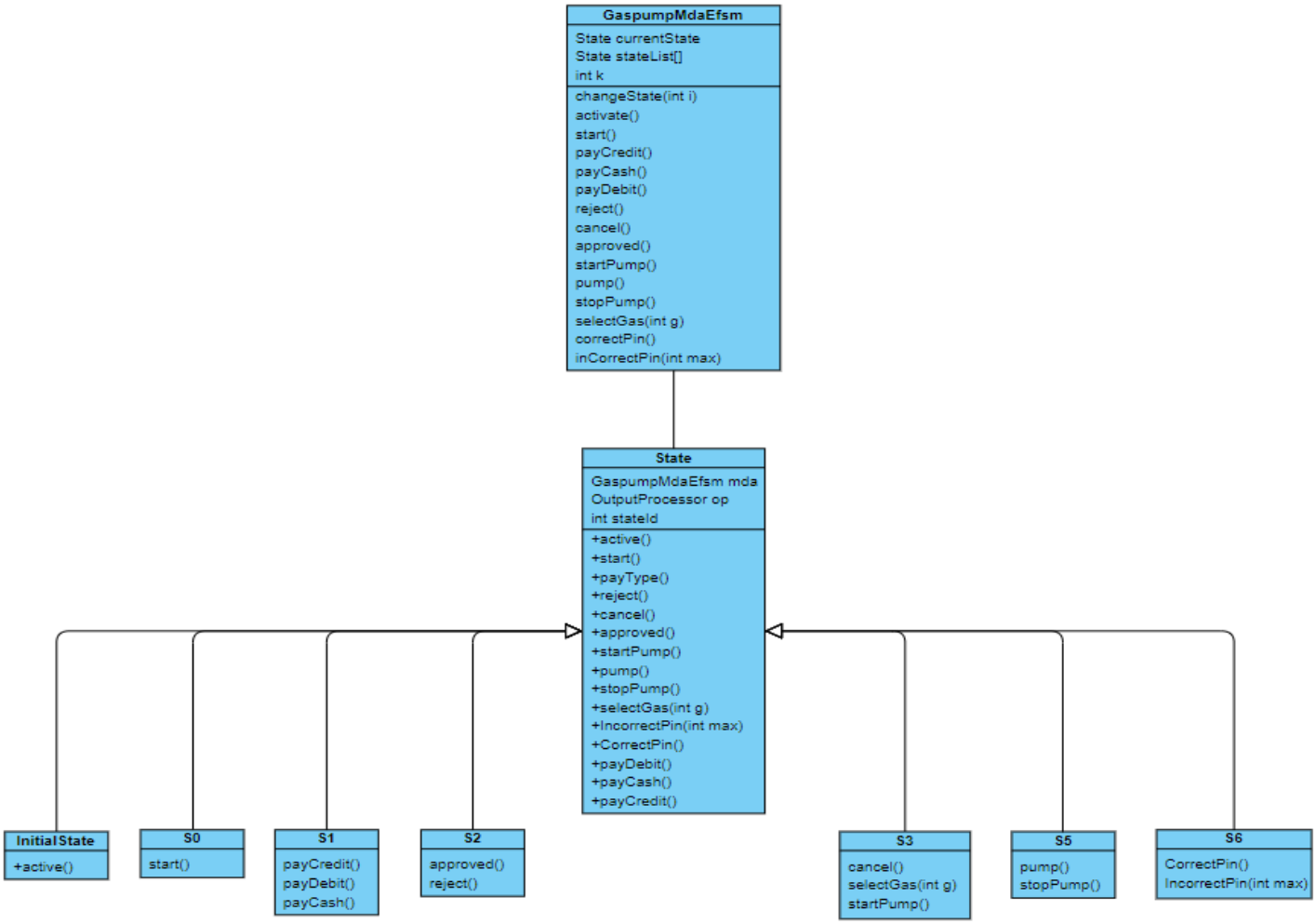
2. Class Diagrams:

2.1. Overall MDA Class Diagram:



All the three patterns will be shown in the following sections, as class diagram with all patterns will be hard to read, I have divided them as below.

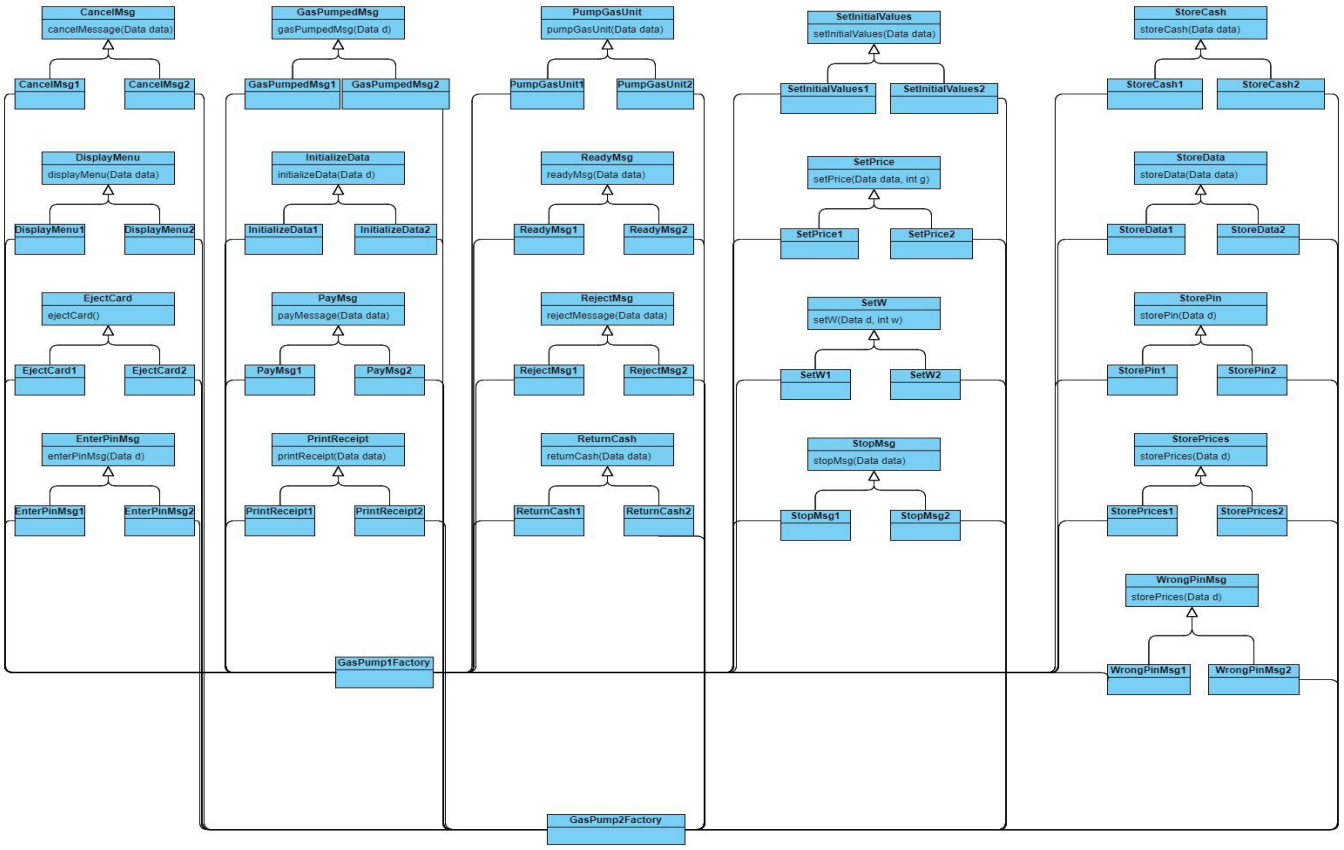
2.2. State Pattern:



- All the state changes are done in the class “GasPumpMdaEfsm” by the function changeState centrally.
- State is a abstract class for all the states available in the system.
- Initial State implements the active function which is used to set the gas price and it has the stated as 0.
- S0 is the state has start which is used to call the initialize data function for the gas pump like the L value and payment types message is displayed to the user. State id is 1.
- S1 is state calls the appropriate implementation for the payments related to gas pump. State id is 2.
- S2 is special state for credit card payment, where either credit card will be approved or rejected. State id is 3.
- S3 provides options for the user after payment options are selected, here user will be prompted for selecting the type of gas they want to pump depending upon the gas pump selected. State id is 4.
- S5 to control the gas pump and S6 is used for the debit card payment

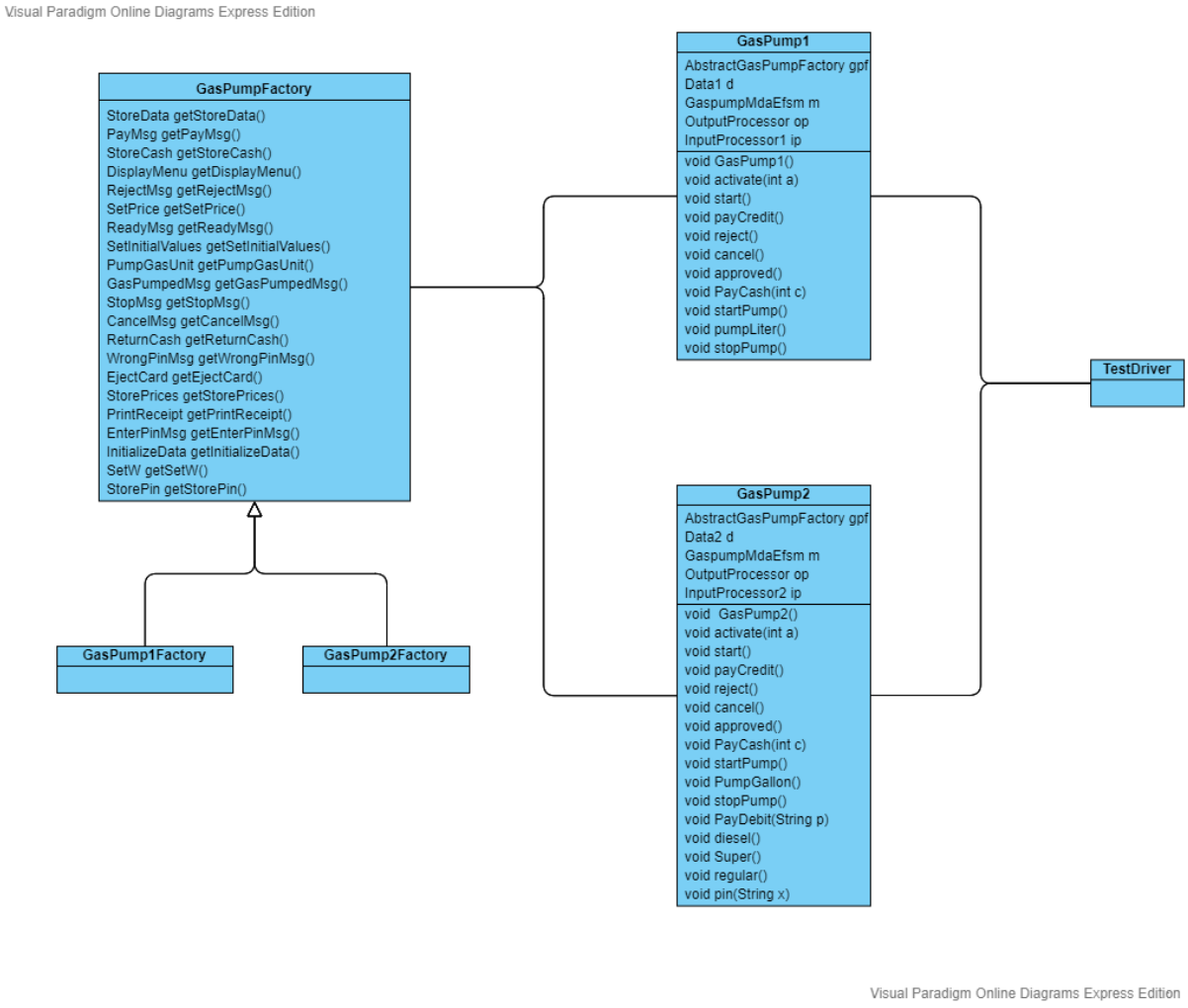
2.3. Strategy Pattern:

All the actions are performed in the strategy classes, every gas pump have its own strategy class and every action will have an interface which will be extended by the class implementing the function. I have followed a pattern where every class ending with a number signifies gas pump for which it is implemented. GasPump1Factory and GasPump2Factory are responsible for selecting the appropriate strategy class.



2.4. Abstract Factory Pattern:

Initialization of the factory is done in the construction of the gaspump1 and gaspump2 class. From previous diagram we can see that gasPump1Factory and gasPump2Factory will be initializing the strategies.



3. Classes in the system:

3.1. TestDriver:

This class is used to as user interface with the gas pump and in this class we initialize the appropriate the concrete factory class (gaspump1factory or gaspump2factory) based on user input. There are two functions testGasPump1 and testGasPump2 which will have the initialize the correct concrete class along with the options for selected gas pump.

3.2. AbstractGasPumpFactory:

This is the interface for the gas pump, it has the signature of all the meta actions that gas pumps can perform.

3.3. GasPump1Factory & GasPump2Factory:

These classes implement the AbstractGasPumpFactory and provides the right strategy to be used for performing the meta events.

3.4. Data:

This is the interface which will be providing the temp storage for meta events throughout the application.

3.5. Data1 and Data2:

Data1 and Data2 are used by gaspump1 and gaspump2 respectively for temporary storage in the meta events.

3.6. GasPump1 and GasPump2:

These classes are used to bind the actual Data, AbstractGasPumpFactory classes based gas pump to the input processor, output processor and GaspumpMdaEfsm. These classes will also call the appropriate function in the input processor for an meta action selected by the user.

3.7. InputProcessor1 and InputProcessor2:

In the input processor we store the function parameters in the temporary storage i.e, in the Data classes and call the MDAEFSM class to handle the meta action. All the functions are defined in the pseudo code in the section 1.4.

3.8. GaspumpMdaEfsm:

This class uses the state pattern to handle the meta actions, it uses the changeState to change the current state centrally. This class also maintains the number of attempts a debit card pin was entered for providing the security functionalities.

3.9. State:

This is an interface for all the state classes. There are currently 7 state classes which are uniquely identified with an integer number which is initialized using the constructor of state classes. This number is used by the changeState function in the GaspumpMdaEfsm and meta events in the state classes to change the current state.

3.10. State classes:

All the meta actions are called using output processor and state change using GasPumpMdaEfm class

3.10.1. InitialState:

Handles the active meta event by calling the meta actions storeData, StorePrices and changes state to S0

3.10.2. S0:

Handles the start meta event by calling the meta actions payMsg, InitilizeData and changes state to S1

3.10.3. S1:

Handles the meta event payCredit by calling the meta actions setW(0) and changes state to S2

Handles the meta event payCash by calling the meta actions setW(1),displayMenu and storeCash and changes state to S3

Handles the meta event payDebit by calling the meta actions setW(0),storePin and enterPinMsg and changes state to S6

3.10.4. S2:

Handles the meta event approved by calling the meta actions displayMenu ,ejectCard and changes state to S5

Handles the meta event reject by calling the meta actions rejectMsg, ejectCard and changes state to S0

3.10.5. S3:

Handles the meta event cancel by calling the meta actions cancelMsg, returnCash and changes state to S0

Handles the meta event selectGas by calling the meta actions setPrice and changes state to S5

Handles the meta event startPump by calling the meta actions setIntialvalues, readyMsg and changes state to S5

3.10.6. S5:

Handles the meta event pump by calling the meta actions pumpGasUnit, GaspumpMsg and changes state to S5

Handles the meta event stopPump by calling the meta actions stopMsg, printReceipt, returnCash and changes state to S0

3.10.7. S6:

Handles the meta event correctPin by calling the meta actions ejectCard, displayMenu and changes state to S5

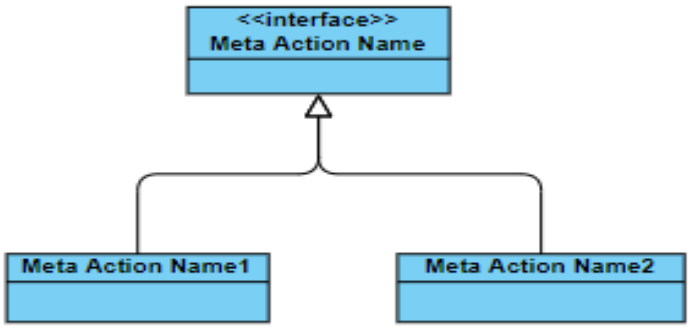
Handles the meta event InCorrectPin by calling the meta actions ejectCard, wrongPinMsg, increment the attempts variable and changes state to S0

3.11. OutputProcessor:

Calls the strategy class based on the factory class passed to it and supplies the temporary data class to these actions.

3.12. Strategy:

There are around 21 meta action classes where the actual work is done, these are structured as below

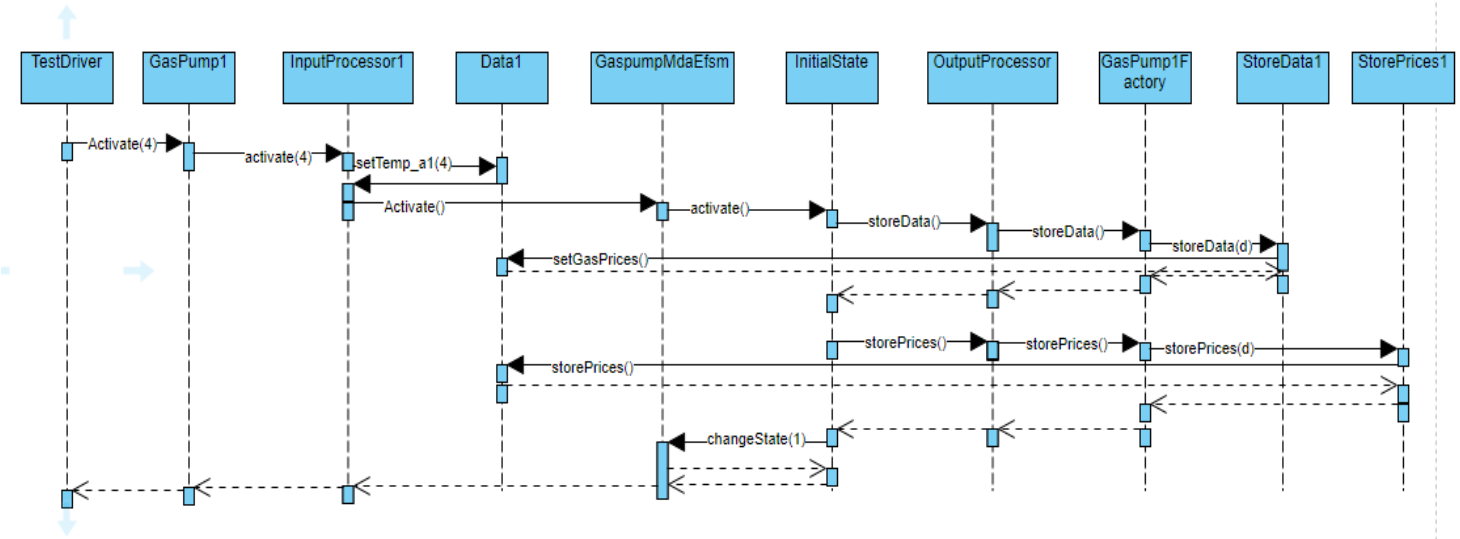


Every meta action will have interface and which will be implemented by two classes, the number in the below class indicates the gas pump for which it has been implemented.

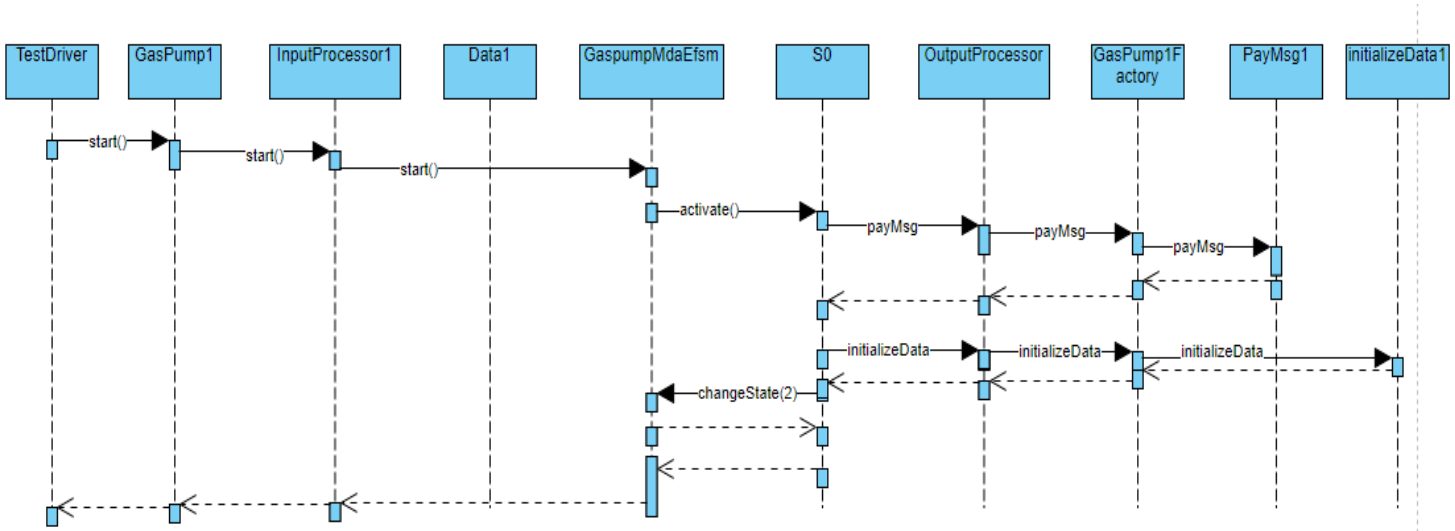
4. Sequence Diagrams:

4.1. Scenario-I:

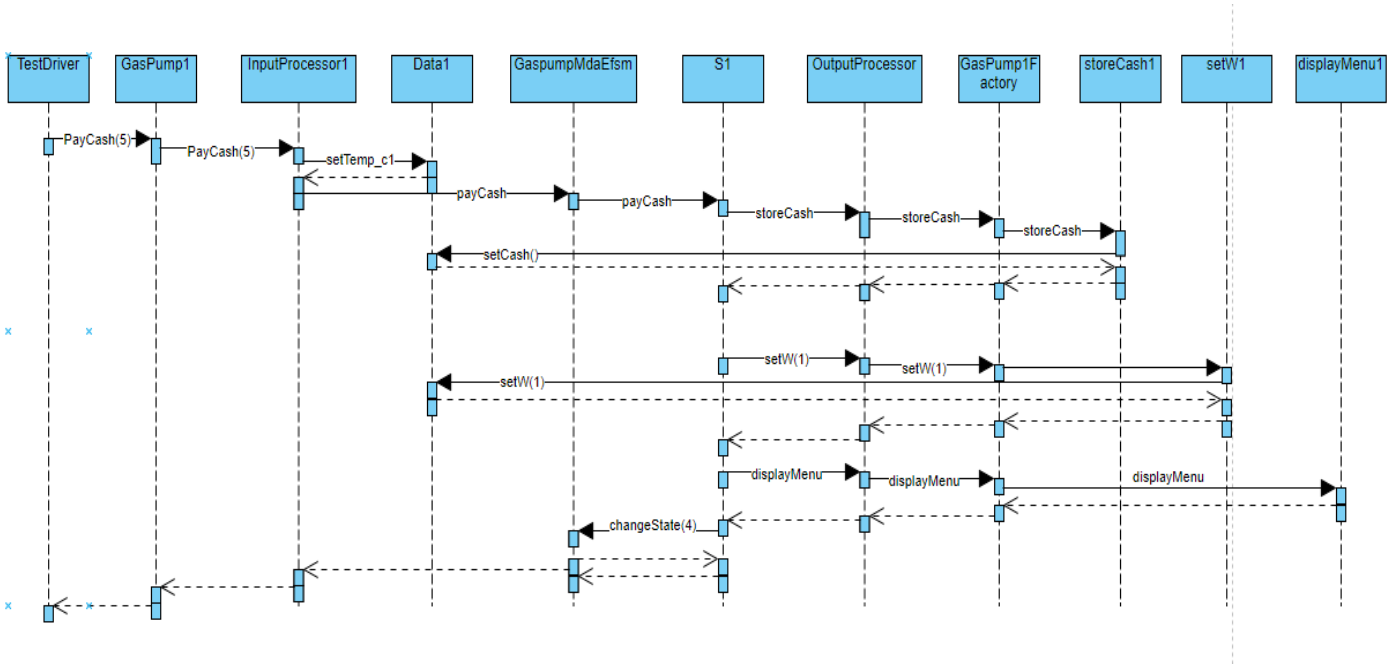
4.1.1. Activate:



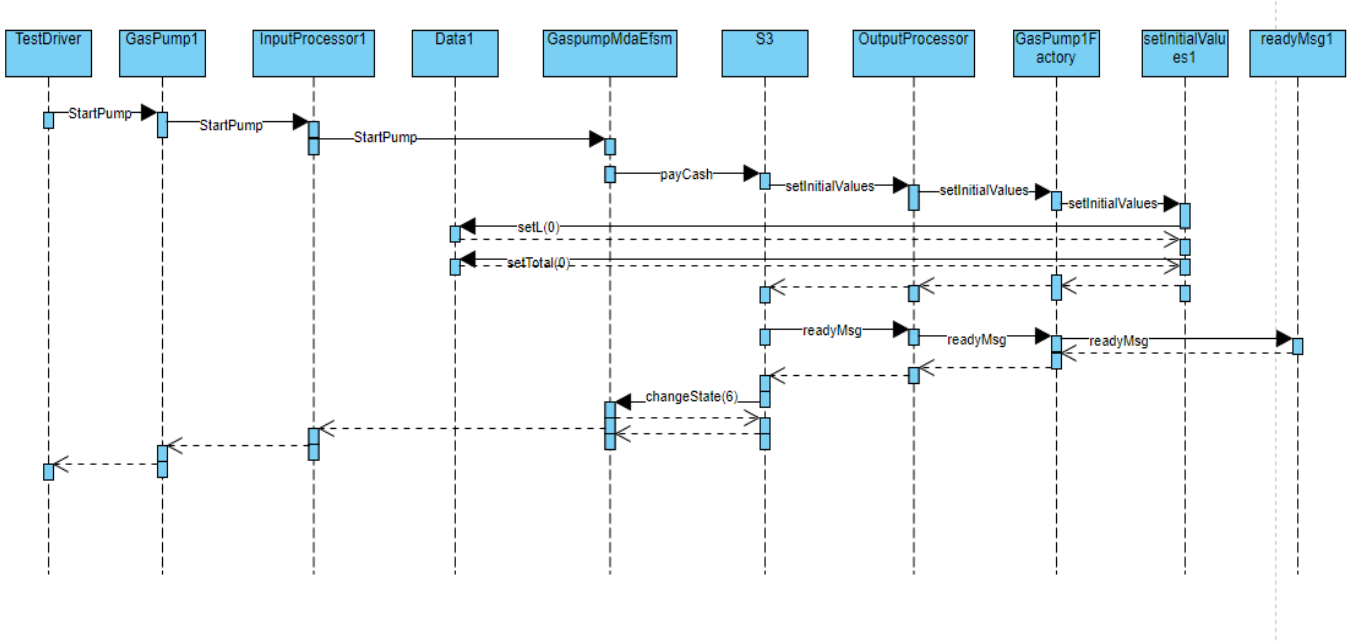
4.1.2. Start:



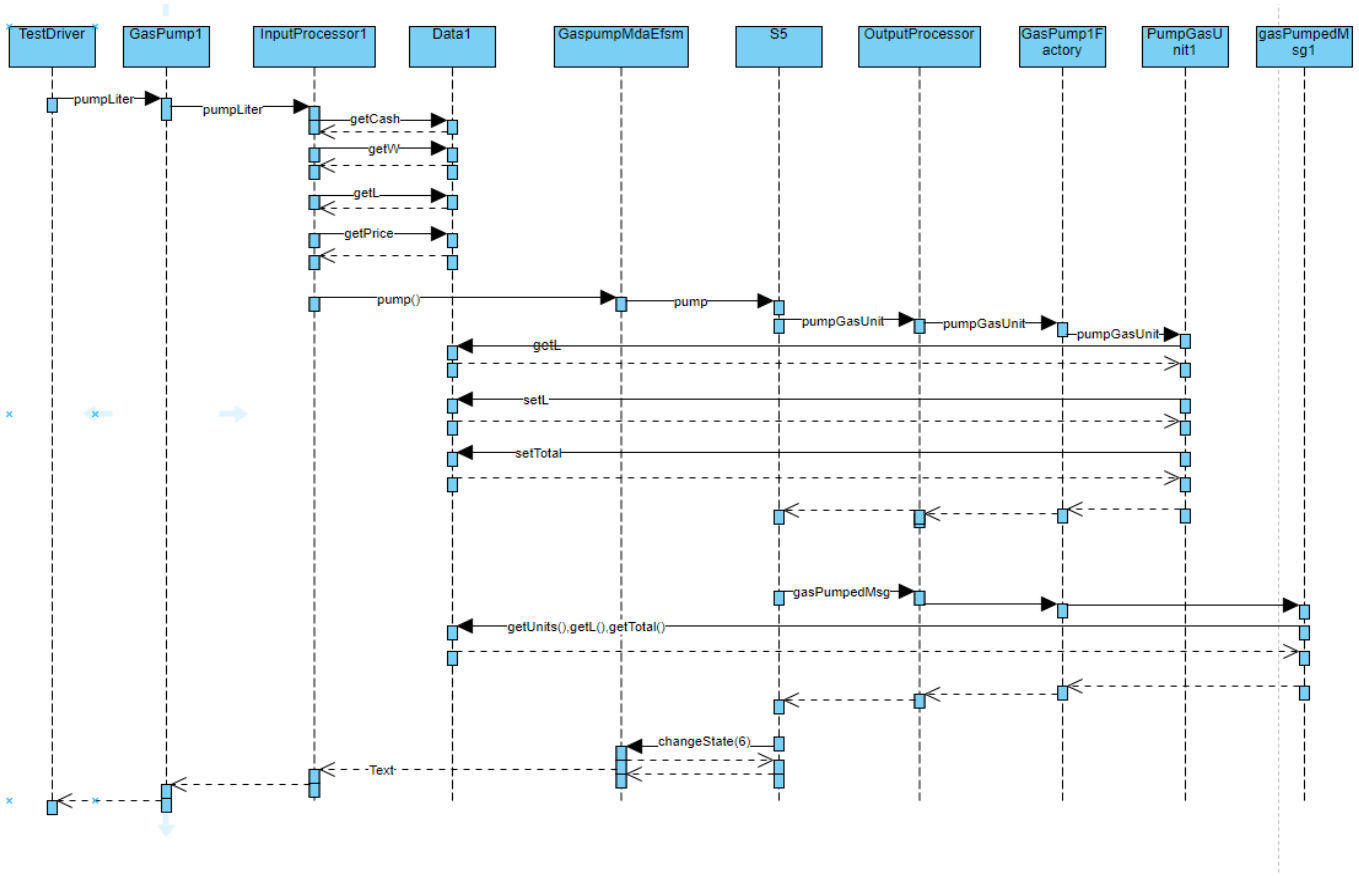
4.1.3. PayCash:



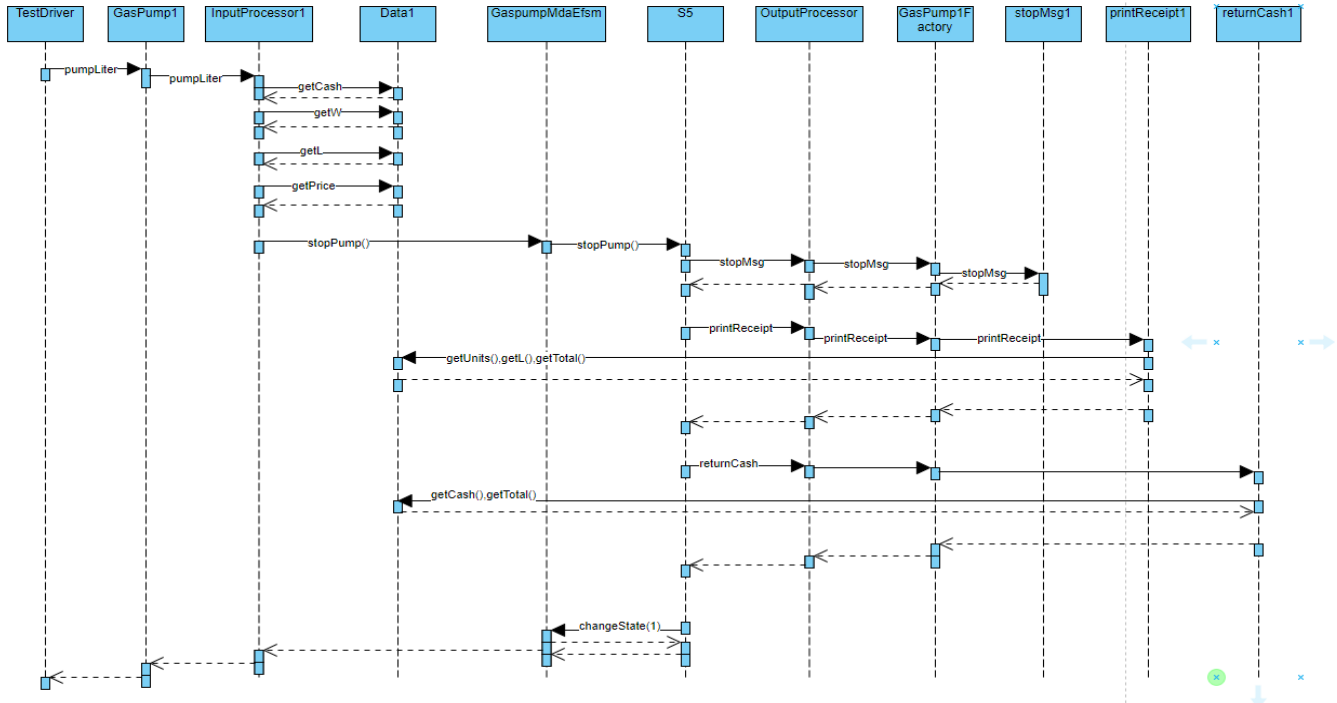
4.1.4. StartPump:



4.1.5. PumpLiter:

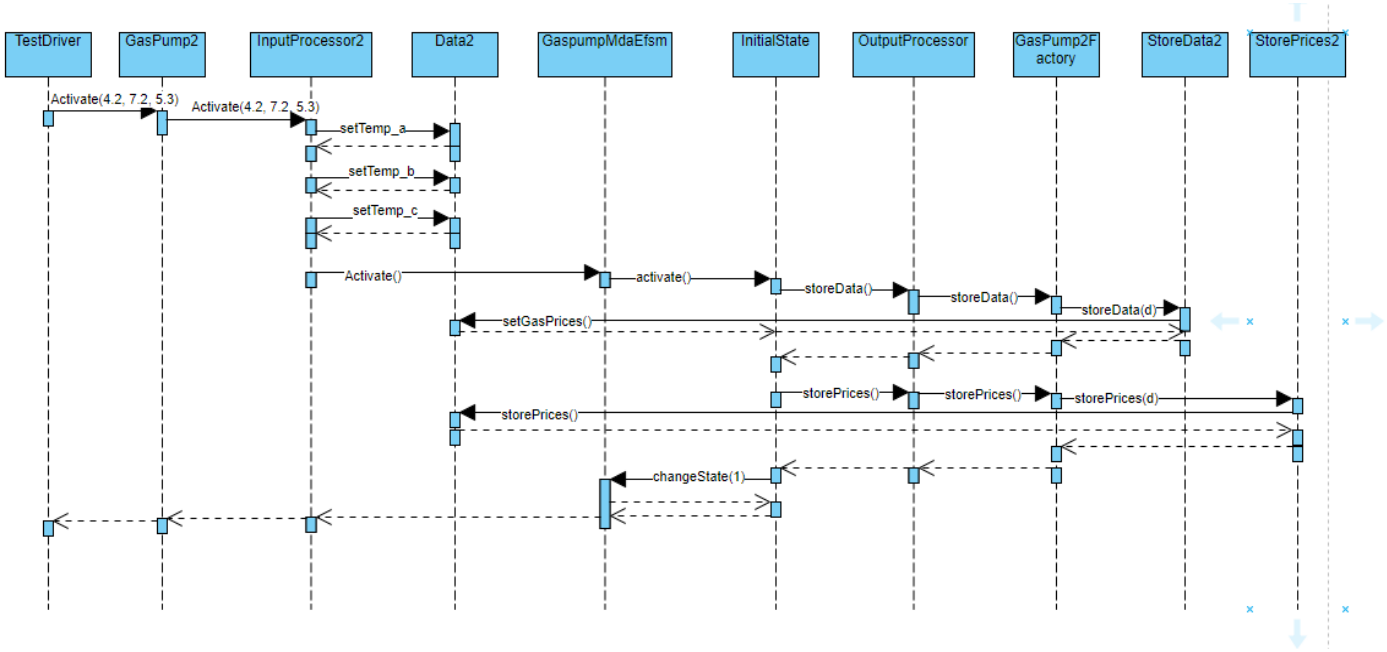


4.1.6. PumpLiter:

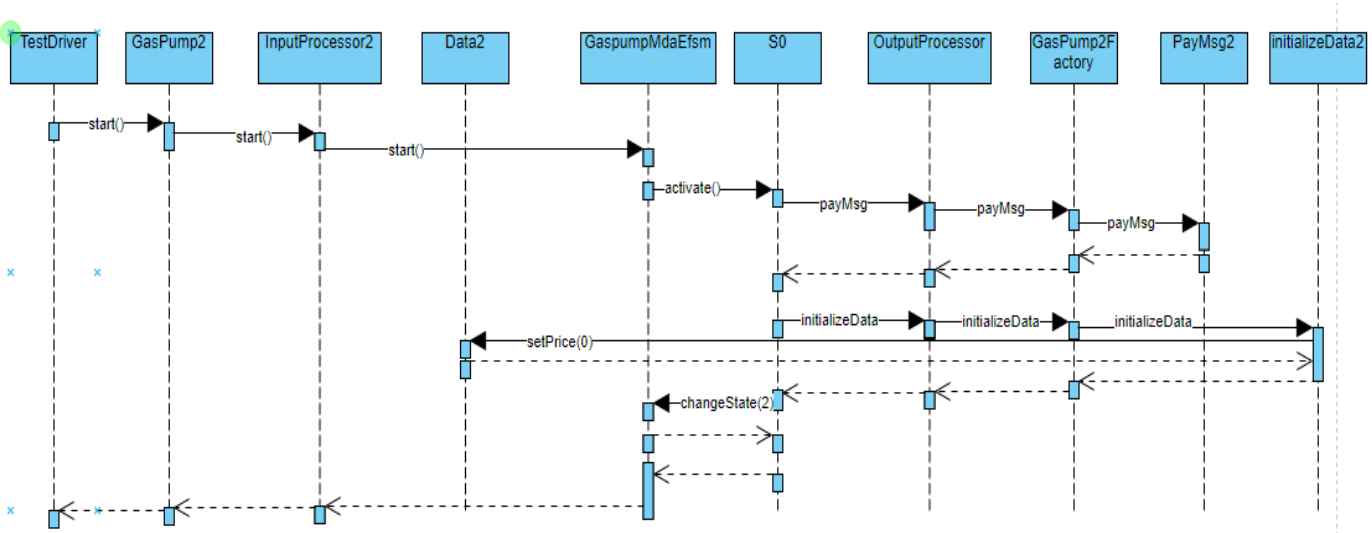


4.2. Scenario-II:

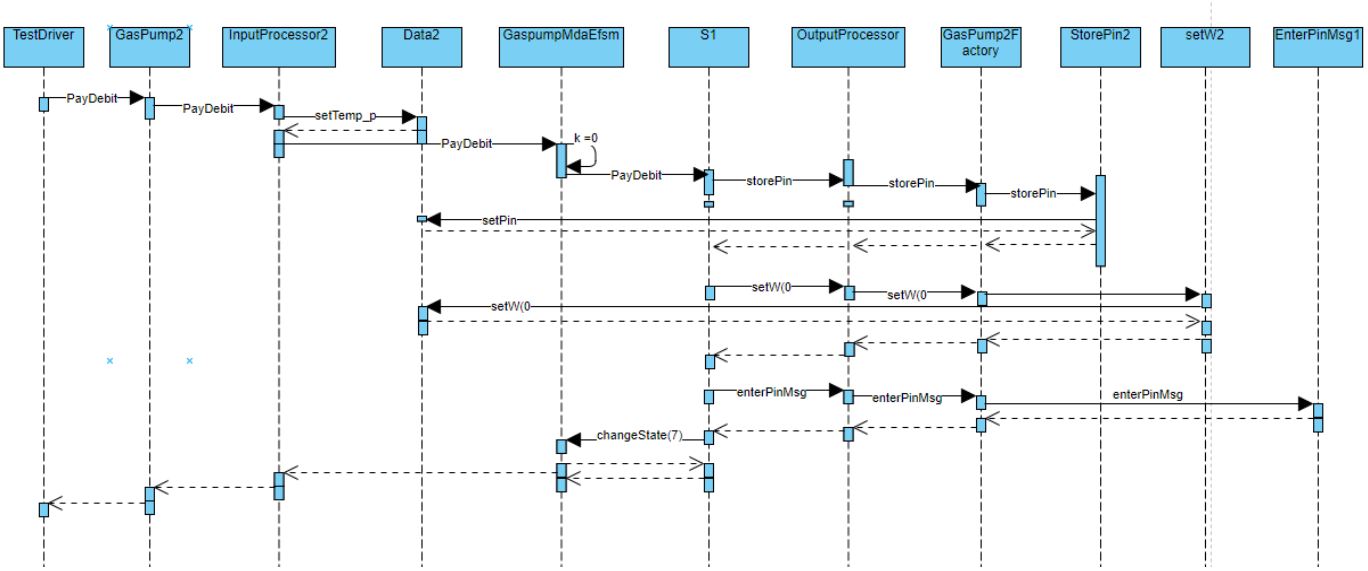
4.2.1. Activate:



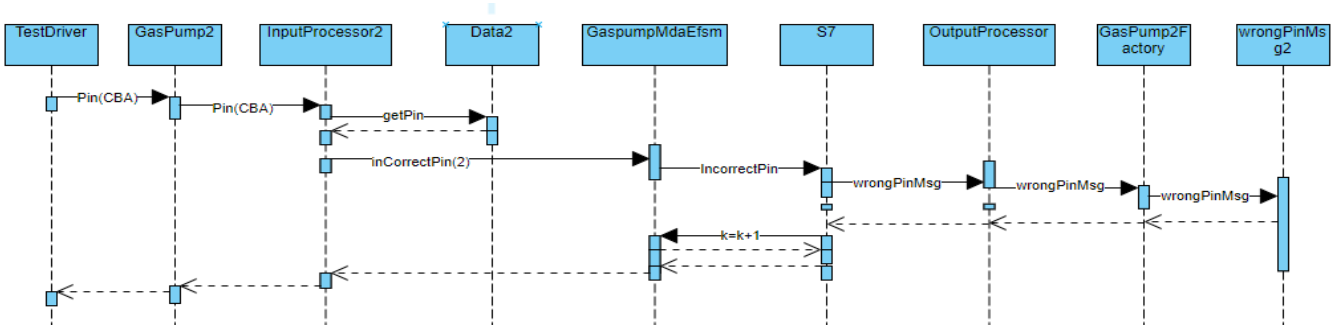
4.2.2. Start:



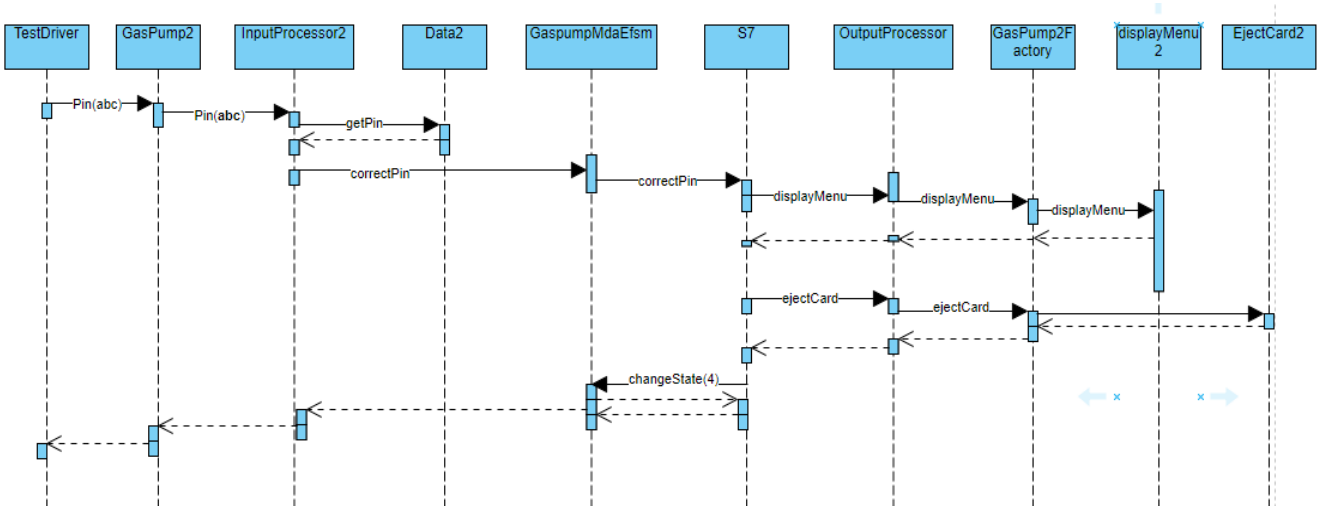
4.2.3. PayDebit:



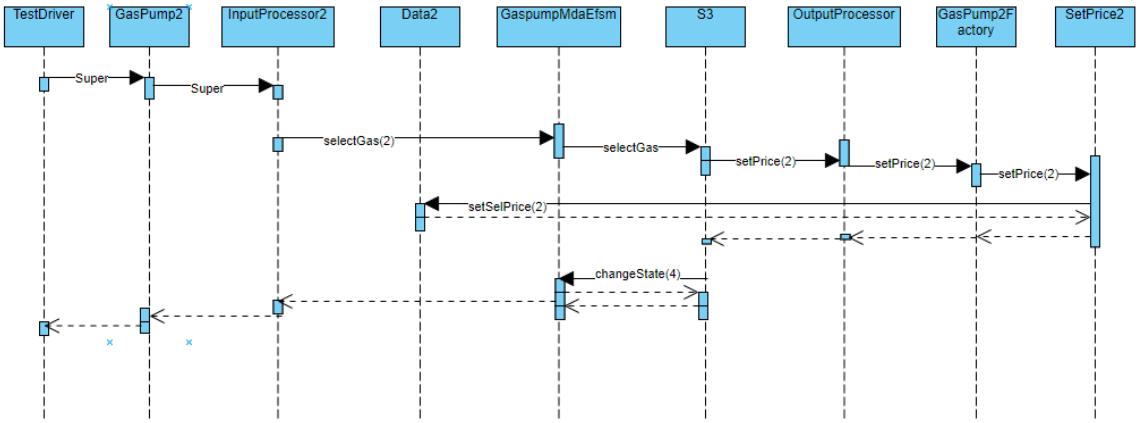
4.2.4. Pin(CBA):



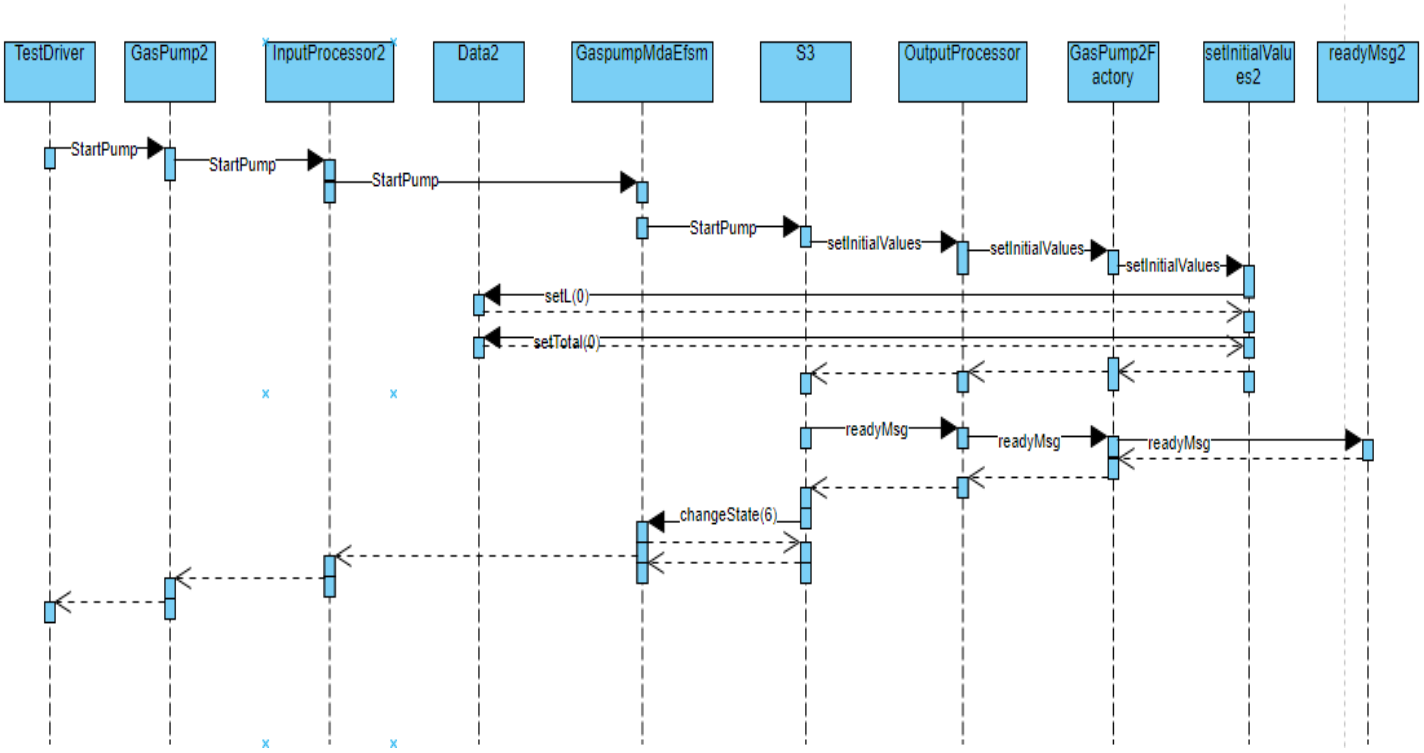
4.2.5. Pin(abc):



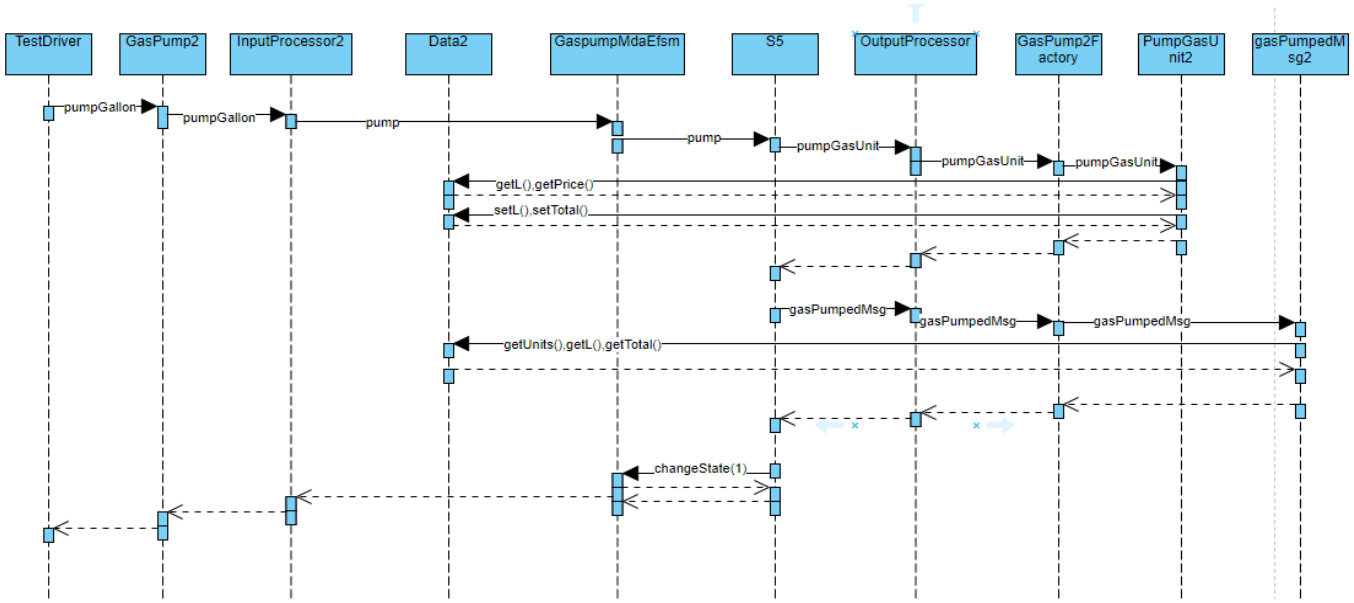
4.2.6. Super:



4.2.7. StartPump:



4.2.8. PumpGallon:



Typo in the above image changeState(6) not changeState(1)

4.2.9. FullTank: