Project Report

Software System Architectures (CS 586)

Professor: Bogdan Korel

Report by:

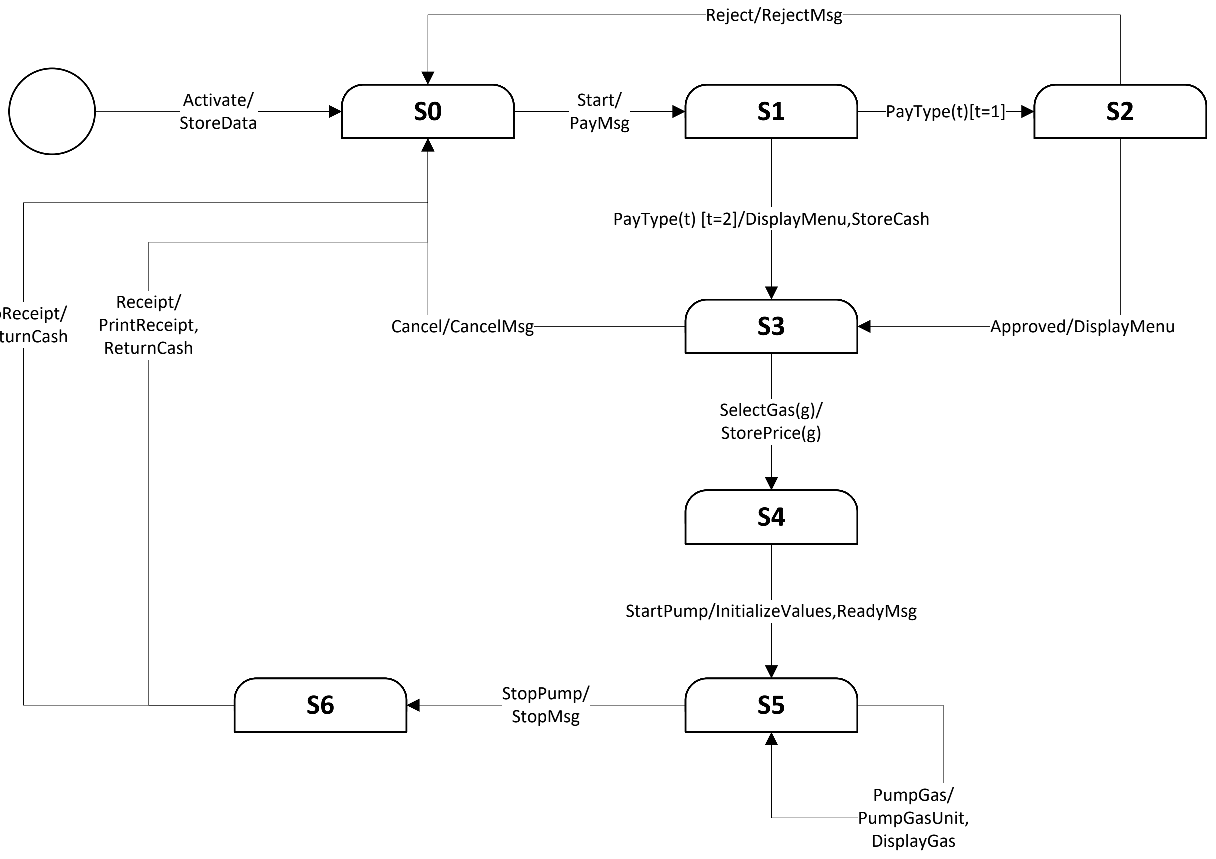
Mohammed Shethwala

A20368337

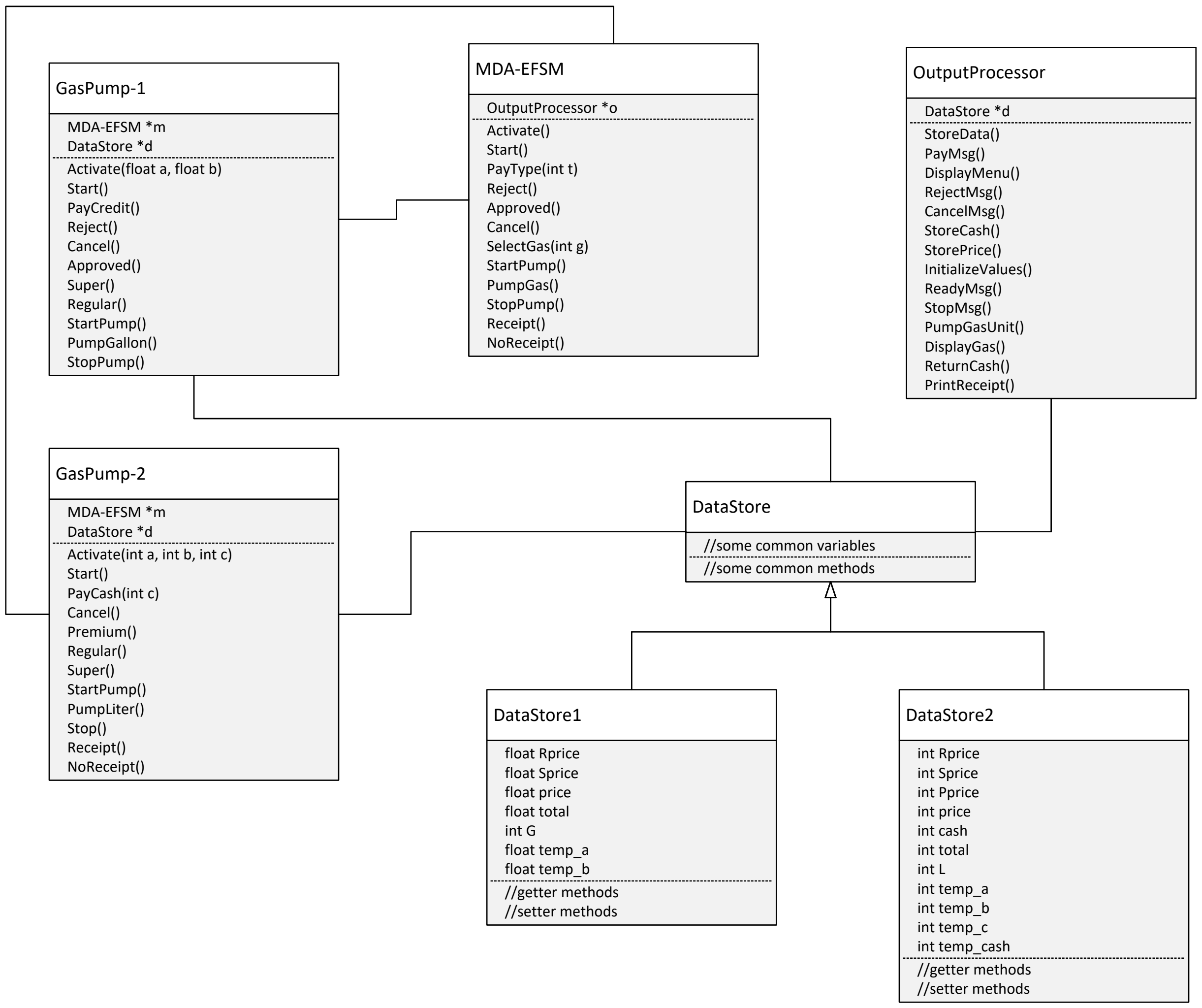
Table of Contents

1. MDA-EFSM model for the GasPump components

1.1. State Diagram


Reject/RejectMsg
Activate/ StoreData
S0
Start/ PayMsg
S1
PayType(t)[t=1]
S2
PayType(t) [t=2]/DisplayMenu,StoreCash
NoReceipt/ ReturnCash
Receipt/ PrintReceipt, ReturnCash
Cancel/CancelMsg
S3
Approved/DisplayMenu
SelectGas(g)/ StorePrice(g)
S4
StartPump/InitializeValues,ReadyMsg
S6
StopPump/ StopMsg
S5
PumpGas/ PumpGasUnit, DisplayGas

1.2. General MDA-EFSM Architecture (without design patterns) – Class Diagram



Notice:

DataStore1 is for the GasPump-1

DataStore2 is for the GasPump-2

1.3. List of Events

Events (Meta Events)
Activate()
Start()
PayType(int t) // credit: t=1; cash: t=2
Reject()
Approved()
Cancel()
SelectGas(int g)
StartPump()
PumpGas()
StopPump()
Receipt()
NoReceipt()

1.4. List of Actions

Actions (Meta Actions)	Responsibility
StoreData()	Stores prices for gas types in data store
PayMsg()	Displays type of payment
DisplayMenu()	Displays selection menu for types of gas (super, premium etc)
RejectMsg()	Displays a reject message if credit card is rejected
CancelMsg()	Displays a cancel message if purchase is cancelled
StoreCash()	Stores cash in data store
StorePrice(int g)	Store price for the selected gas type in data store
InitializeValues()	Sets the total units of gas to 0 (e.g: G=0 or L=0) and sets the total amount to 0 (e.g: total=0)
ReadyMsg()	Displays ‘ready for pumping’ message
StopMsg()	Displays stop pumping message & Receipt or NoRecipt message (optional)
PumpGasUnit()	Pumps 1 unit of gas and counts total units of gas pumped
DisplayGas()	Displays the total units of gas pumped
ReturnCash()	Returns the remaining cash
PrintReceipt()	Prints a receipt

1.5. GasPump1 Pseudocode

Notice:

d *is a pointer to the DataStore object*

m *is a pointer to MDA-EFSM object*

temp_a*,* ***temp_b*** *are used to store temporary data in the DataStore*

```
1. Activate(float a, float b)
        if(a>0 && b>0)
                d.temp_a = a;
                d.temp_b = b;
                m.Activate()
2. Start()
        m.Start()
3. PayCredit()
        m.PayType(1)
4. Reject()
        m.Reject()
5. Cancel()
        m.Cancel()
6. Approved()
        m.Approved()
7. Super()
        m.SelectGas(2)
8. Regular()
        m.SelectGas(1)
9. StartPump()
        m.StartPump()
```

```
10. PumpGallon()
        m.PumpGas()
11. StopPump()
        m.StopPump()
        m.Receipt()
```

1.6. GasPump2 Pseudocode

Notice:

d *is a pointer to the DataStore object*

m *is a pointer to MDA-EFSM object*

cash *contains the inserted cash in the DataStore*

price *contains the selected gas type’s price in the DataStore*

L *contains the total no. of liters pumped*

temp_a**, **temp_b**, **temp_c**, **temp_cash *are used to store temporary data in the DataStore*

```
1. 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.Activate()
2. Start()
        m.Start()
3. PayCash(int c)
        if(c>0)
                d.temp_cash = c
                m.PayType(2)
4. Cancel()
```

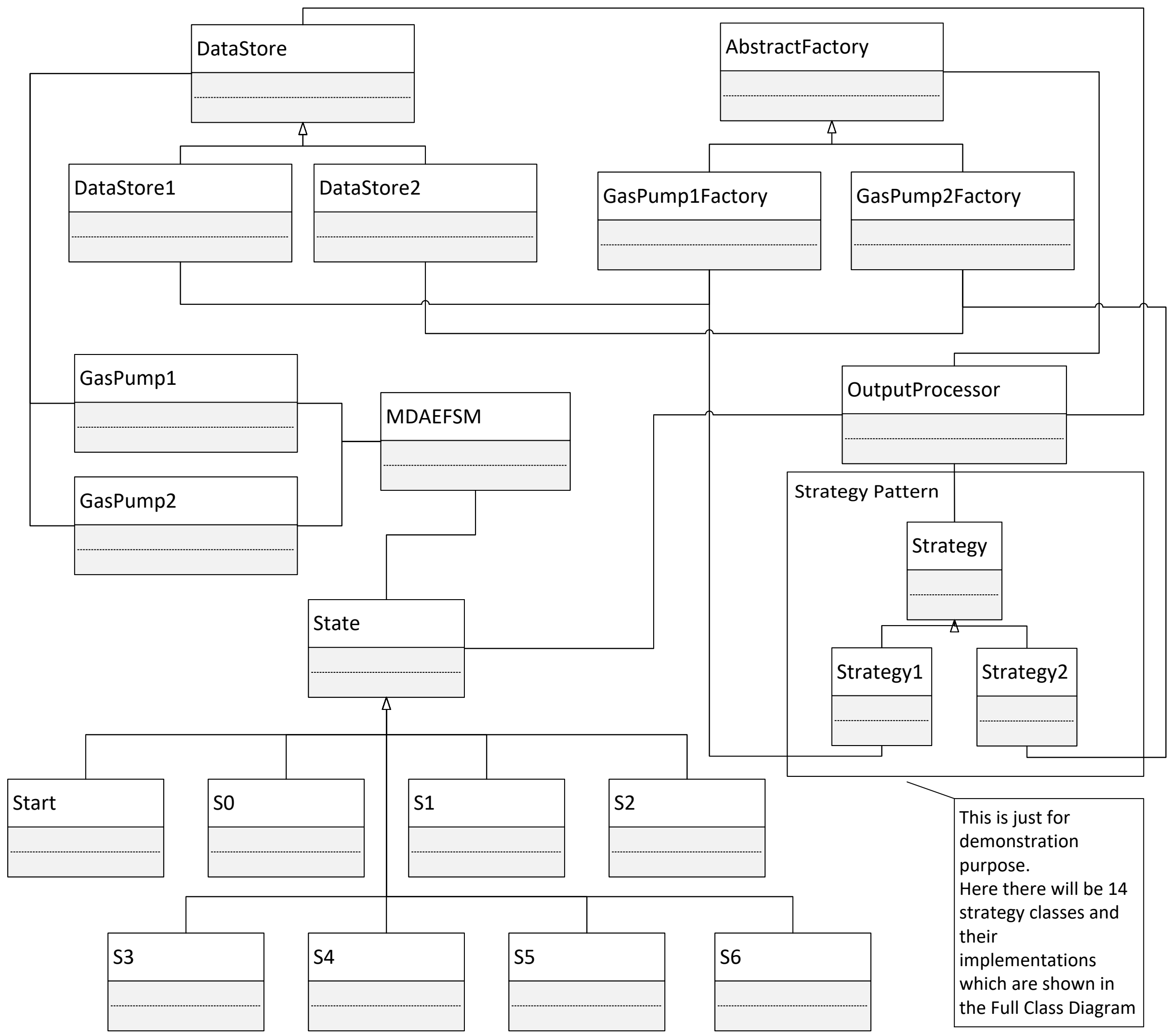
```
    m.Cancel()
5. Premium()
    m.SelectGas(3)
6. Regular()
    m.SelectGas(1)
7. Super()
    m.SelectGas(2)
8. StartPump()
    m.StartPump()
9. PumpLiter()
    if(d.cash < (d.L+1 * d.price))
        m.StopPump()
    else
        m.PumpGas()
10. Stop()
    m.StopPump()
11. Receipt()
    m.Receipt()
12. NoReceipt()
    m.NoReceipt()
```

2. Class Diagrams (MDA Architecture for the GasPump Components)

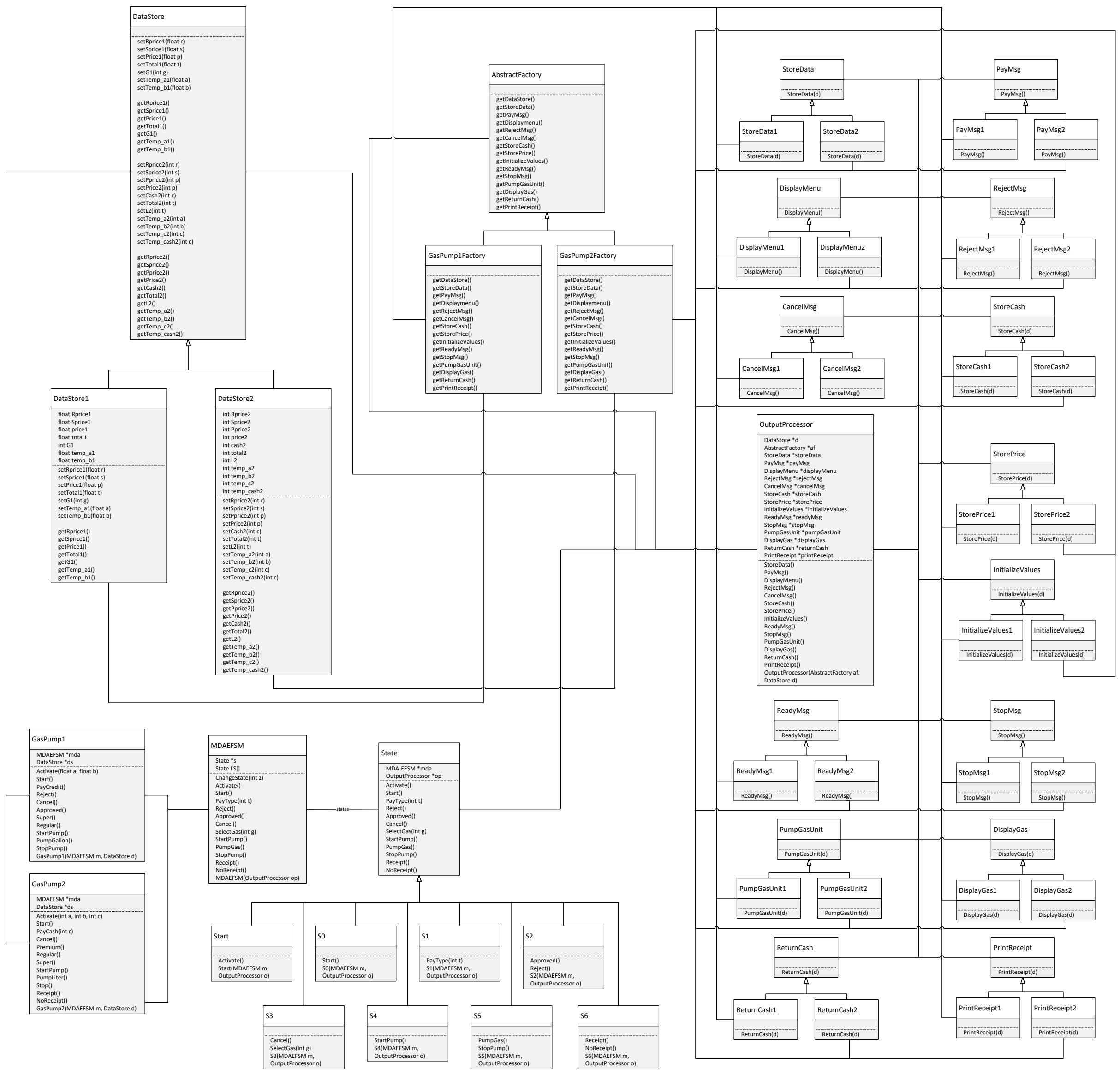
The Class Diagram is very wide and long and so it takes a lot of space. As there are a lot of classes, it is not easy to visualize the design patterns in this class diagram.

For the sake of presentation, I have further divided the class diagram in next sections according to the design patterns applied. There is one class diagram which shows the State Pattern and there are three class diagrams which shows Abstract Factory and Strategy Pattern among a set of classes. Section 2.1 gives the general class diagram showing the connections between all the classes. Note that this is just for demonstration purpose. Section 2.2 gives the complete class diagram consisting of all the classes. Section 2.3 gives the State Pattern presentation and Section 2.4 gives the Strategy and Abstract Factory pattern presentation.

2.1. General High Level Class Diagram

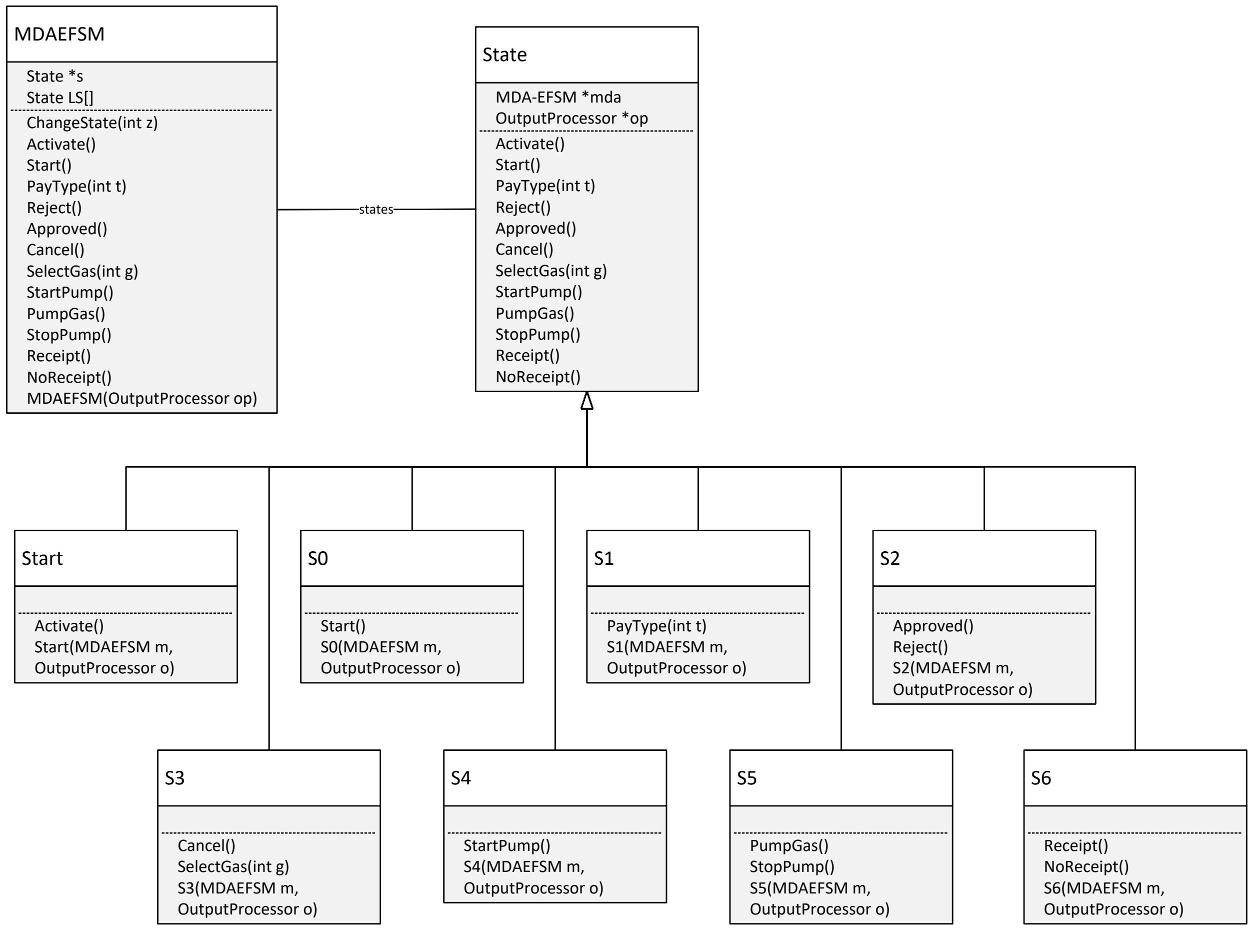


2.2. Complete Class Diagram



2.3 State Pattern Class Diagram

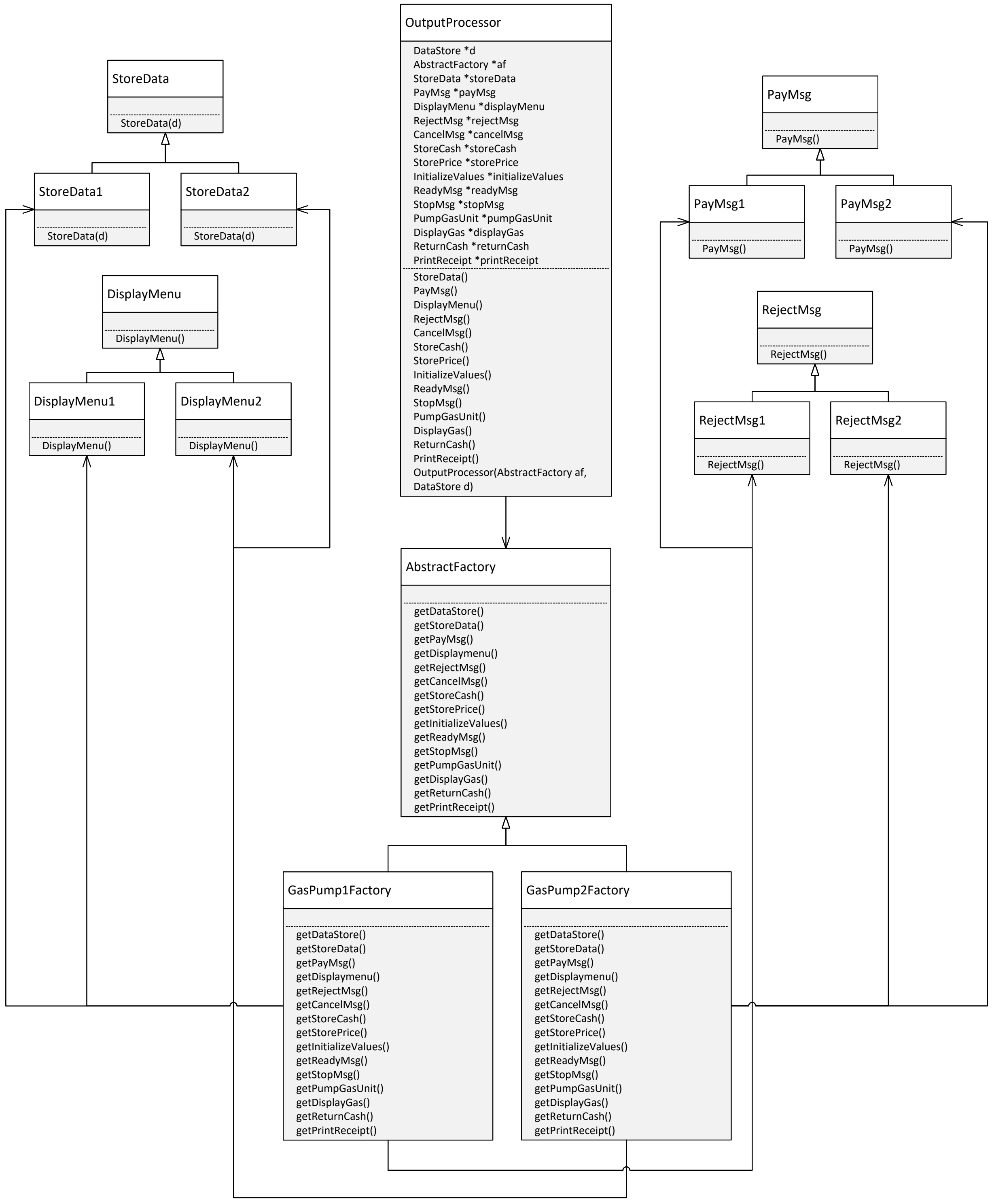
The De-Centralized version of the State Pattern has been applied in this architecture



2.4 Abstract Factory & Strategy Pattern

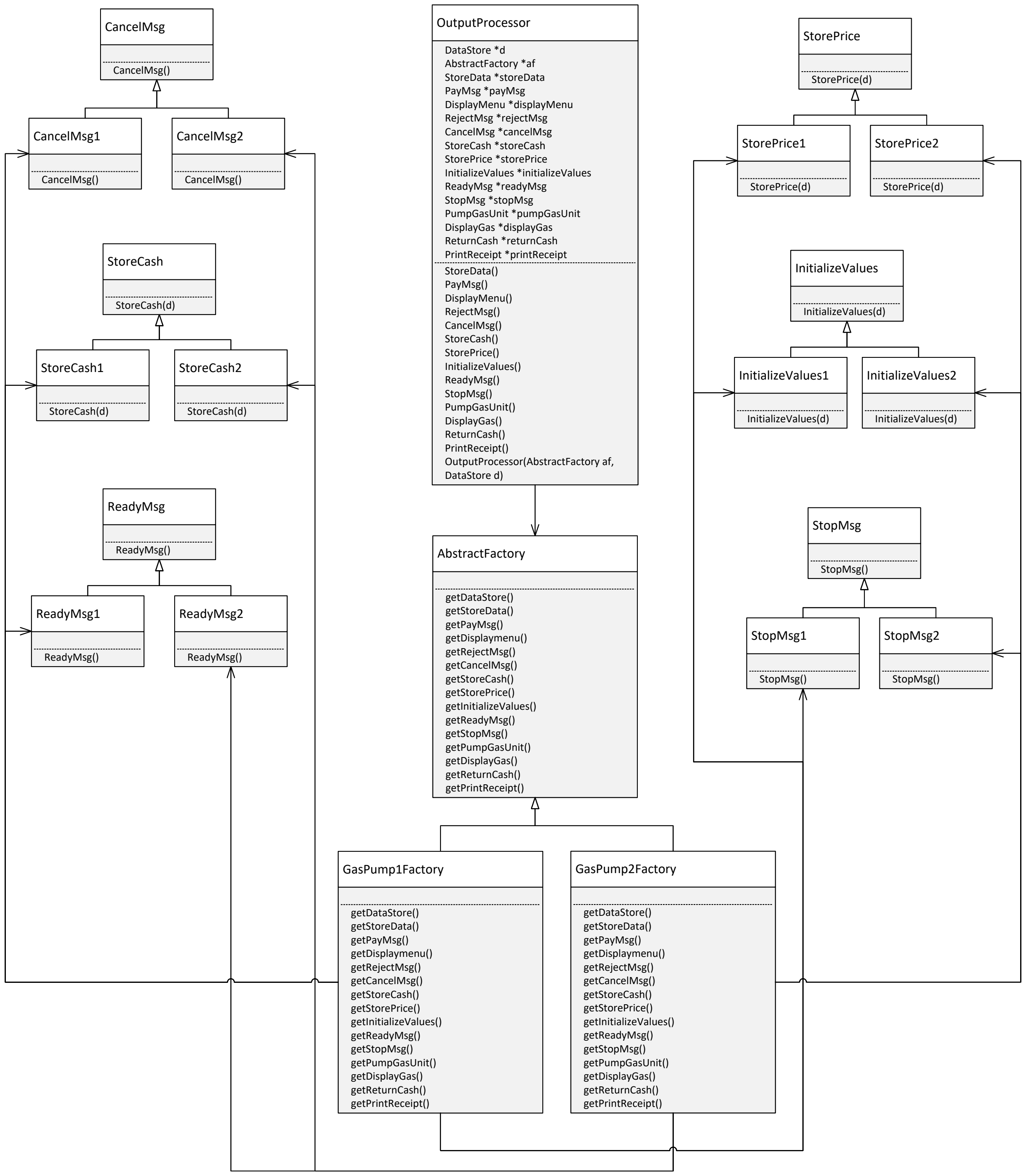
2.4.1. Class Diagram 1 for the following set of classes:

- Output Processor, Abstract Factory, StoreData, PayMsg, RejectMsg, Displaymenu



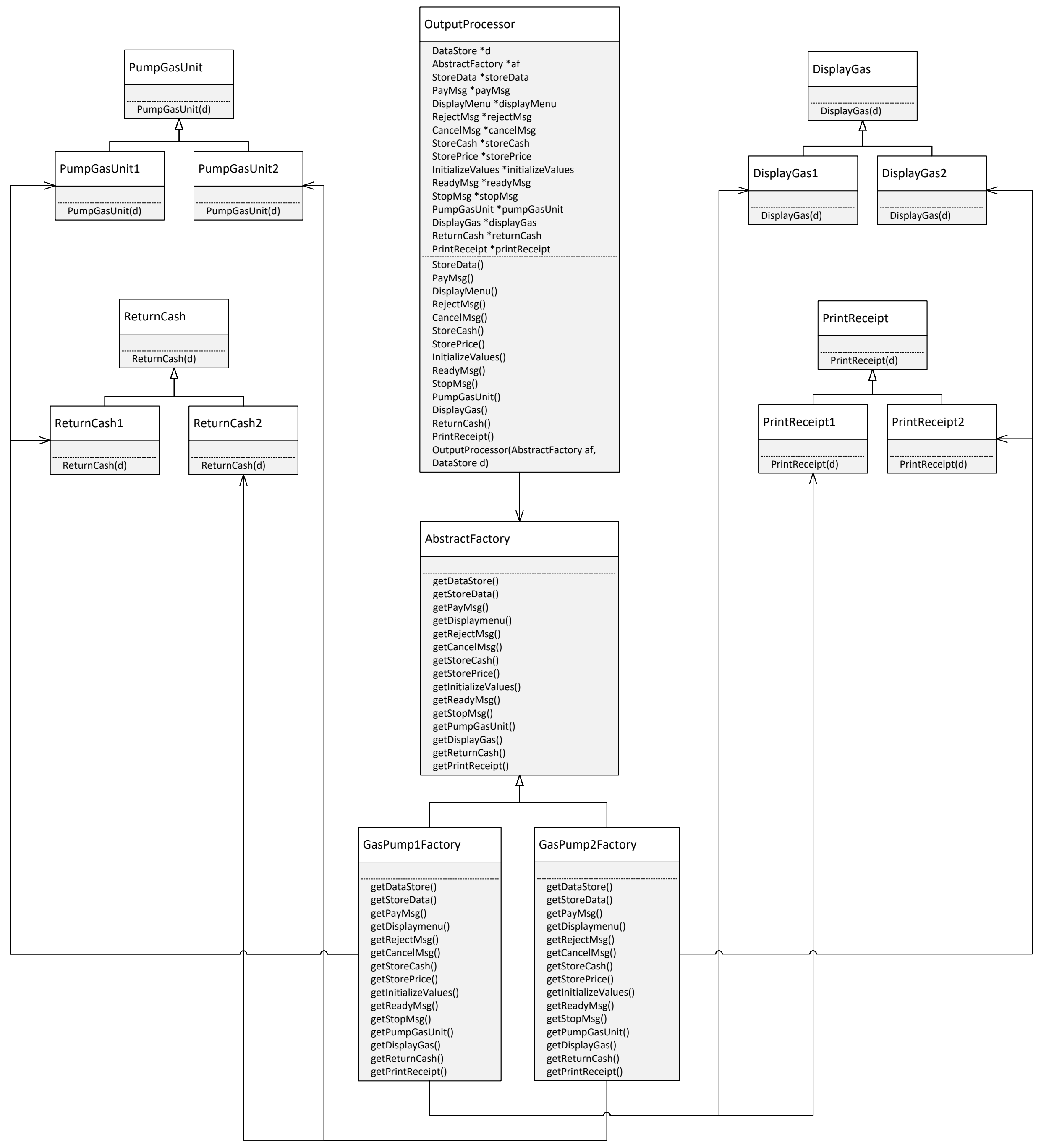
2.4.2. Class Diagram 2 for the following set of classes:

- OutputProcessor, AbstractFactory, CancelMsg, StoreCash, ReadyMsg, StorePrice, InitializeValues and StopMsg.



2.4.3. Class Diagram 3 for the following set of classes:

- OutputProcessor, AbstractFactory, PumpGasUnit, ReturnCash, DisplayGas & PrintReceipt



3. Purpose & Responsibilities

Class GasPump1

Purpose of the class: This is the Input Processor for the MDA Architecture. Operations to be performed on a GasPump1 are called on this class

Responsibility of each operation:

1. Activate(float a, float b): sets the temp variables in data store and calls Activate() on MDAEFSM class
2. Start(): calls Start() on MDAEFSM class
3. PayCredit(): calls PayType(1) on MDAEFSM class
4. Reject(): calls Reject() on MDAEFSM class
5. Cancel(): calls Cancel() on MDAEFSM class
6. Approved(): calls Approved() on MDAEFSM class
7. Super(): calls SelectGas(2) on MDAEFSM class
8. Regular(): calls SelectGas(1) on MDAEFSM class
9. StartPump(): calls StartPump() on MDAEFSM class
10. PumpGallon(): calls PumpGas() on MDAEFSM class
11. StopPump(): calls StopPump() and Receipt() on MDAEFSM class

Class GasPump2

Purpose of the class: This is the Input Processor for the MDA Architecture. Operations to be performed on a GasPump1 are called on this class

Responsibility of each operation:

1. Activate(int a, int b, int c): sets the temp variables in data store and calls Activate() on MDAEFSM class
2. Start(): calls Start() on MDAEFSM class
3. PayCash(int c): sets the value of c in DataStore and calls PayType(2) on MDAEFSM class
4. Cancel(): calls Cancel() on MDAEFSM class
5. Premium(): calls SelectGas (3) on MDAEFSM class
6. Super(): calls SelectGas(2) on MDAEFSM class
7. Regular(): calls SelectGas(1) on MDAEFSM class
8. StartPump(): calls StartPump() on MDAEFSM class

9. PumpLiter(): checks if there is enough cash to continue pumping. If enough cash, then calls PumpGas() on MDAEFSM class, else, calls StopPump() on MDAEFSM class

10. Stop(): calls StopPump() on MDAEFSM class

11. Receipt(): calls Receipt() on MDAEFSM class

12. NoReceipt: calls NoReceipt() on MDAEFSM class

Class AbstractFactory (Abstract Class)

Purpose of the class: This is an abstract class used to provide abstract methods which must be implemented by the concrete factory classes

Responsibility of each operation: All operations are abstract in this class

Class GasPump1Factory

Purpose of the class: This is the Concrete Factory for the GasPump1. Responsible for returning GasPump1 related objects from within the particular methods

Responsibility of each operation:

1. getDataStore(): return an object of DataStore1

2. getCancelMsg (): return an object of CancelMsg1

3. getDisplayGas (): return an object of DisplayGas1

4. getDisplayMenu (): return an object of DisplayMenu1

5. getInitializeValues (): return an object of InitializeValues1

6. getPayMsg (): return an object of PayMsg1

7. getPrintReceipt (): return an object of PrintReceipt1

8. getPumpGasUnit (): return an object of PumpGasUnit1

9. getReadyMsg (): return an object of ReadyMsg1

10. getRejectMsg (): return an object of RejectMsg1

11. getReturnCash (): return an object of ReturnCash1

12. getStopMsg (): return an object of StopMsg1

13. getStoreCash (): return an object of StoreCash1

14. getStoreData (): return an object of StoreData1

15. getStorePrice (): return an object of StorePrice1

Class GasPump2Factory

Purpose of the class: This is the Concrete Factory for the GasPump2. Responsible for returning GasPump2 related objects from within the particular methods

Responsibility of each operation:

1. getDataStore(): return an object of DataStore2

2. getCancelMsg (): return an object of CancelMsg2

3. getDisplayGas (): return an object of DisplayGas2

4. getDisplayMenu (): return an object of DisplayMenu2

5. getInitializeValues (): return an object of InitializeValues2

6. getPayMsg (): return an object of PayMsg2

7. getPrintReceipt (): return an object of PrintReceipt2

8. getPumpGasUnit (): return an object of PumpGasUnit2

9. getReadyMsg (): return an object of ReadyMsg2

10. getRejectMsg (): return an object of RejectMsg2

11. getReturnCash (): return an object of ReturnCash2

12. getStopMsg (): return an object of StopMsg2

13. getStoreCash (): return an object of StoreCash2

14. getStoreData (): return an object of StoreData2

15. getStorePrice (): return an object of StorePrice2

Class DataStore

Purpose of the class: This is an abstract DataStore class

Responsibility of each operation: All the operations here are abstract

Class DataStore1

Purpose of the class: This is the concrete DataStore1 for the GasPump1. Responsible for storing the temporary and permanent variables for the GasPump1 execution

Responsibility of each operation: All the operations are getter and setter methods for getting and setting the variables respectively

Class DataStore2

Purpose of the class: This is the concrete DataStore2 for the GasPump2. Responsible for storing the temporary and permanent variables for the GasPump2 execution

Responsibility of each operation: All the operations are getter and setter methods for getting and setting the variables respectively

Class MDAEFSM

Purpose of the class: This is the MDA-EFSM class which acts a context class for the state pattern. The operations delegate the work to respective state classes.

Responsibility of each operation:

1. ChangeState(int z): changes the current state to LS[z]
2. Activate(): delegate the call to current state's Activate() method
3. Start(): delegate the call to current state's Start() method
4. PayType(int t): delegate the call to current state's PayType(int t) method
5. Reject(): delegate the call to current state's Reject() method
6. Approved(): delegate the call to current state's Approved() method
7. Cancel(): delegate the call to current state's Cancel() method
8. SelectGas(int g): delegate the call to current state's SelectGas(int g) method
9. StartPump(): delegate the call to current state's StartPump() method
10. PumpGas(): delegate the call to current state's PumpGas() method
11. StopPump(): delegate the call to current state's StopPump() method
12. Receipt(): delegate the call to current state's Receipt() method
13. NoReceipt(): delegate the call to current state's NoReceipt() method

Class State

Purpose of the class: Abstract class used to provide abstract methods which must be implemented by the state classes

Responsibility of each operation: All the operations are abstract

Class Start

Purpose of the class: Responsible for providing the implementation of Start state

Responsibility of each operation:

1. Activate(): calls StoreData() on OutputProcessor and changes the current state to S0

Class S0

Purpose of the class: Responsible for providing the implementation of S0 state

Responsibility of each operation:

1. Start(): calls PayMsg() on OutputProcessor and changes the current state to S1

Class S1

Purpose of the class: Responsible for providing the implementation of S1 state

Responsibility of each operation:

1. PayType(): if t is 1 (credit card), then changes the state to S2. else if t is 2 then calls DisplayMenu() and StoreCash() on OutputProcessor and changes the state to S3

Class S2

Purpose of the class: Responsible for providing the implementation of S2 state

Responsibility of each operation:

1. Approved(): calls DisplayMenu() on OutputProcessor and changes the current state to S3

2. Reject(): calls RejectMsg() on OutputProcessor and changes the current state to S0

Class S3

Purpose of the class: Responsible for providing the implementation of S3 state

Responsibility of each operation:

1. Cancel(): calls CancelMsg() on OutputProcessor and changes the current state to S0

2. SelectGas(int g): calls StorePrice(g) on OutputProcessor and changes the current state to S4

Class S4

Purpose of the class: Responsible for providing the implementation of S4 state

Responsibility of each operation:

1. StartPump(): calls InitializeValues() and ReadyMsg() on OutputProcessor and changes the current state to S5

Class S5

Purpose of the class: Responsible for providing the implementation of S5 state

Responsibility of each operation:

1. PumpGas(): calls PumpGasUnit() and DisplayGas() on OutputProcessor

2. StopPump(): calls StopMsg() on OutputProcessor and changes the current state to S6

Class S6

Purpose of the class: Responsible for providing the implementation of S6 state

Responsibility of each operation:

1. Receipt(): calls PrintReceipt() and ReturnCash() on OutputProcessor and changes current state to S0

2. NoReceipt(): calls ReturnCash() on OutputProcessor and changes the current state to S0

Class OutputProcessor

Purpose of the class: This is the OutputProcessor class which is responsible for performing the actual actions of the GasPump components

Responsibility of each operation:

1. StoreData(): calls StoreData() on StoreData (may be StoreData1 or StoreData2 depending on the GasPump component being used) class by passing an object of DataStore

2. PayMsg (): calls PayMsg () on PayMsg (may be PayMsg1 or PayMsg2 depending on the GasPump component being used) class by passing an object of DataStore

3. DisplayMenu (): calls DisplayMenu () on DisplayMenu (may be DisplayMenu1 or DisplayMenu2 depending on the GasPump component being used) class by passing an object of DataStore

4. RejectMsg (): calls RejectMsg () on RejectMsg (may be RejectMsg1 or RejectMsg2 depending on the GasPump component being used) class by passing an object of DataStore

5. CancelMsg (): calls CancelMsg () on CancelMsg (may be CancelMsg1 or CancelMsg2 depending on the GasPump component being used) class by passing an object of DataStore

6. StoreCash (): calls StoreCash () on StoreCash (may be StoreCash1 or StoreCash2 depending on the GasPump component being used) class by passing an object of DataStore

7. StorePrice (int g): calls StorePrice (int g) on StorePrice (may be StorePrice1 or StorePrice2 depending on the GasPump component being used) class by passing an object of DataStore

8. InitializeValues (): calls InitializeValues () on InitializeValues (may be InitializeValues1 or InitializeValues2 depending on the GasPump component being used) class by passing an object of DataStore

9. ReadyMsg (): calls ReadyMsg () on ReadyMsg (may be ReadyMsg1 or ReadyMsg2 depending on the GasPump component being used) class by passing an object of DataStore

10. StopMsg (): calls StopMsg () on StopMsg (may be StopMsg1 or StopMsg2 depending on the GasPump component being used) class by passing an object of DataStore

11. PumpGasUnit (): calls PumpGasUnit () on PumpGasUnit (may be PumpGasUnit1 or PumpGasUnit2 depending on the GasPump component being used) class by passing an object of DataStore

12. DisplayGas (): calls DisplayGas () on DisplayGas (may be DisplayGas1 or DisplayGas2 depending on the GasPump component being used) class by passing an object of DataStore

13. ReturnCash (): calls ReturnCash () on ReturnCash (may be ReturnCash1 or ReturnCash2 depending on the GasPump component being used) class by passing an object of DataStore

14. PrintReceipt (): calls PrintReceipt () on PrintReceipt (may be PrintReceipt1 or PrintReceipt2 depending on the GasPump component being used) class by passing an object of DataStore

Class CancelMsg, CancelMsg1 & CancelMsg2

Purpose & Responsibility of the class and it's operations: A set of classes used to perform the cancelling of the GasPumps. It display a cancel message in both gas pumps

Class DisplayGas, DisplayGas1 & DisplayGas2

Purpose & Responsibility of the class and it's operations: A set of classes used to display the current amount of gas disposed. DisplayGas1 displays it in gallons, whereas DisplayGas2 displays it in Liters

Class DisplayMenu, DisplayMenu1 & DisplayMenu2

Purpose & Responsibility of the class and it's operations: A set of classes used to display the main menu to the user. DisplayMenu1 displays 2 types of selections whereas DisplayMenu2 displays 3 types of selections

Class InitializeValues, InitializeValues1 & InitializeValues2

Purpose & Responsibility of the class and it's operations: A set of classes used to initialize the initial values to the DataStore. InitializeValues1 initializes G and Total to 0 whereas InitializeValues2 initializes L and Total to 0

Class PayMsg, PayMsg1 & PayMsg2

Purpose & Responsibility of the class and it's operations: A set of classes used to display a pay message. PayMsg1 displays 1 selection of 'PayCredit' whereas PayMsg2 displays 1 selection of 'PayCash' to the user

Class PrintReceipt, PrintReceipt1 & PrintReceipt2

Purpose & Responsibility of the class and it's operations: A set of classes used to print a receipt to the user. PrintReceipt1 prints the receipt in Gallons and Total price whereas PrintReceipt2 primts receipt in Liters and Total price.

Class PumpGasUnit, PumpGasUnit1 & PumpGasUnit2

Purpose & Responsibility of the class and it's operations: A set of classes used to pump gas in the vehicle. PumpGasUnit1 pumps 1 gallon of gas whereas PumpGasUnit2 pumps 1 liter of gas

Class ReadyMsg, ReadyMsg1 & ReadyMsg2

Purpose & Responsibility of the class and it's operations: A set of classes used to alert the user by displaying ready message when the pump is ready for pumping.

Class RejectMsg, RejectMsg1 & RejectMsg2

Purpose & Responsibility of the class and it's operations: A set of classes used to display a reject message if a credit card is rejected. RejectMsg1 displays a reject message whereas RejectMsg2 does nothing because GasPump2 does not support credit cards

Class ReturnCash, ReturnCash1 & ReturnCash2

Purpose & Responsibility of the class and it's operations: A set of classes used to return the remaining cash to the user. ReturnCash1 does nothing because cash is not supported by GasPump1 whereas ReturnCash2 returns the remaining amount to the user

Class StopMsg, StopMsg1 & StopMsg2

Purpose & Responsibility of the class and it's operations: A set of classes used to display a stop message to the user. Both StopMsg1 and StopMsg2 displays stop message to the user

Class StoreCash, StoreCash1 & StoreCash2

Purpose & Responsibility of the class and it's operations: A set of classes used to store the cash entered by the user. StoreCash1 does nothing because it does not support cash whereas StoreCash2 stores the cash in the DataStore

Class StoreData, StoreData1 & StoreData2

Purpose & Responsibility of the class and it's operations: A set of classes used to store the data from temp variables into the main variables of DataStore. StoreData1 and StoreData2 both saves the respected initial data to the DataStore

Class StorePrice, StorePrice1 & StorePrice2

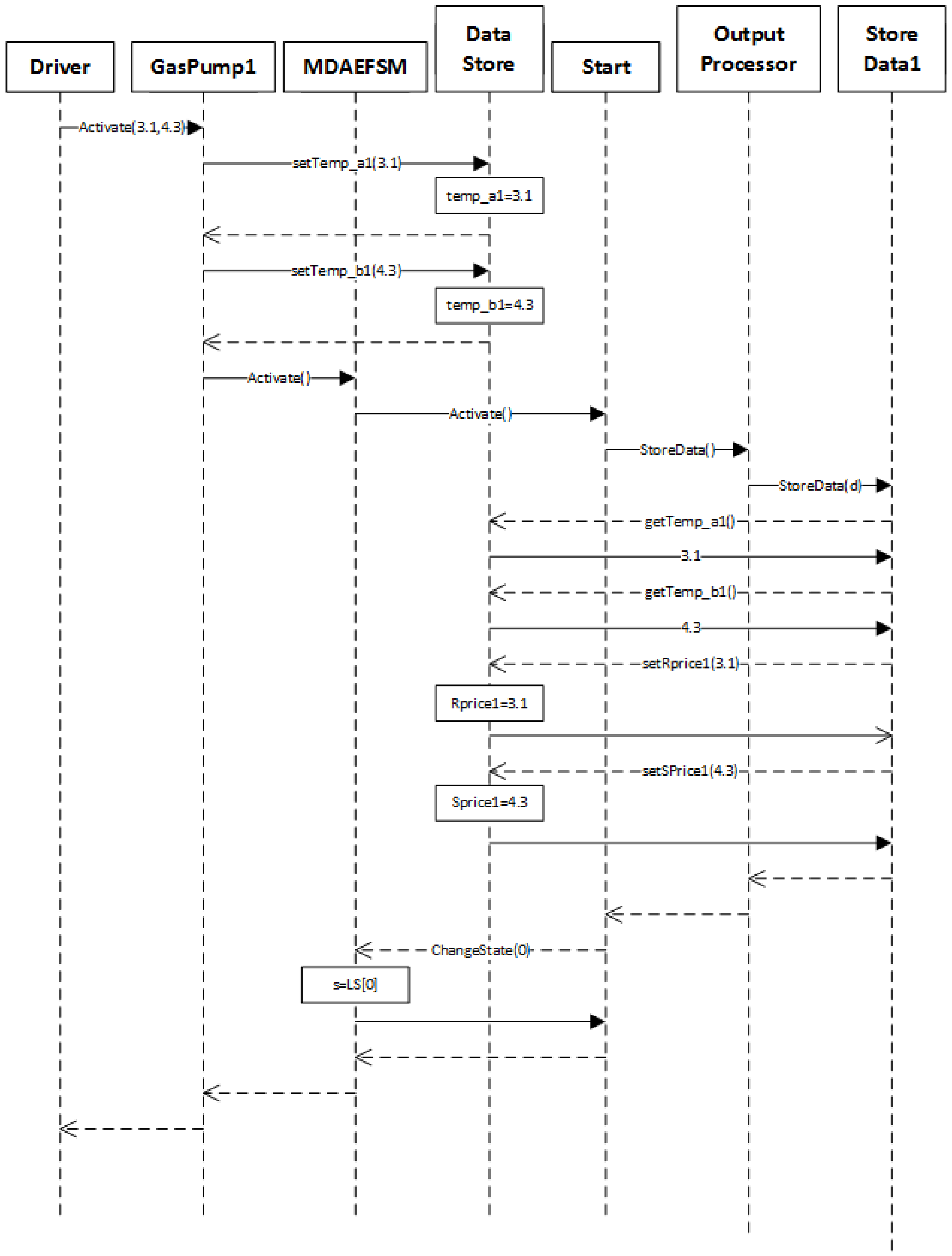
Purpose & Responsibility of the class and it's operations: A set of classes used to store the price of the selected gas type in the DataStore for further calculations. StorePrice1 and StorePrice2 both stores the price of the selected gas type in DataStore

4. Sequence Diagrams

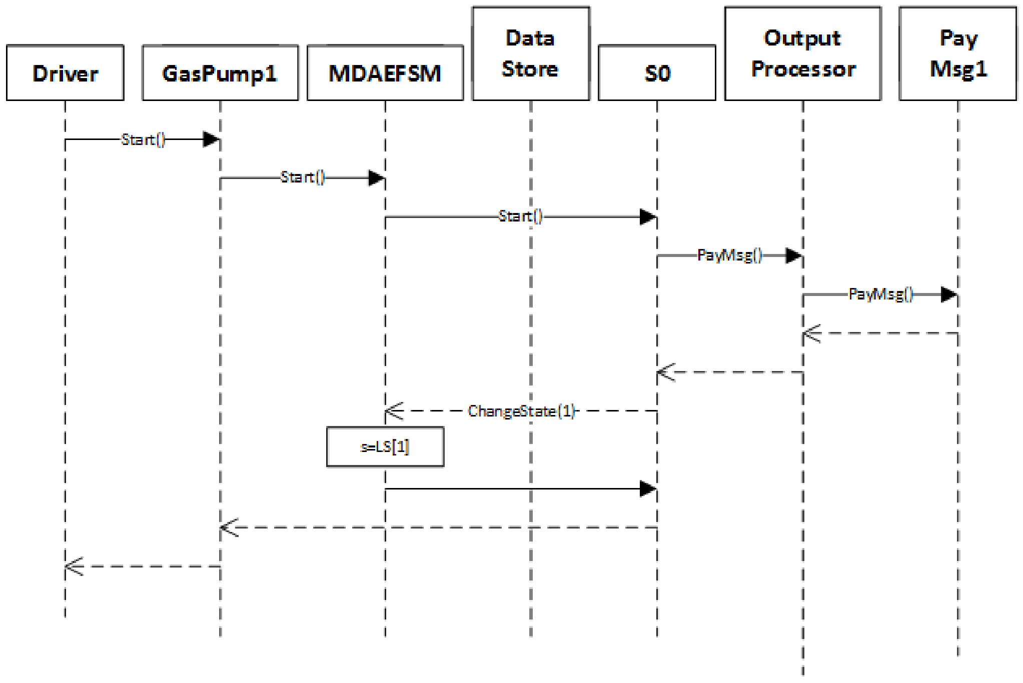
- The sequence diagrams for Scenario 1 and Scenario 2 are very long and wide.
- For the sake of presentation and visibility, I have divided the Scenario 1 sequence diagram into 7 different sequence diagrams and Scenario 2 into 6 different sequence diagrams.
- Each sequence diagram either represents one or two operations from the given scenario.
- All the 7 diagrams are arranged in the section hierarchically (The first diagram is the starting point and the last diagram is the end point of the complete sequence diagram).
- Also, in a particular sequence diagram, I have only included the objects which are active during the flow of that sequence diagram.
- All the sequence diagrams start from the next page

4.1. Scenario 1 Sequence Diagram:

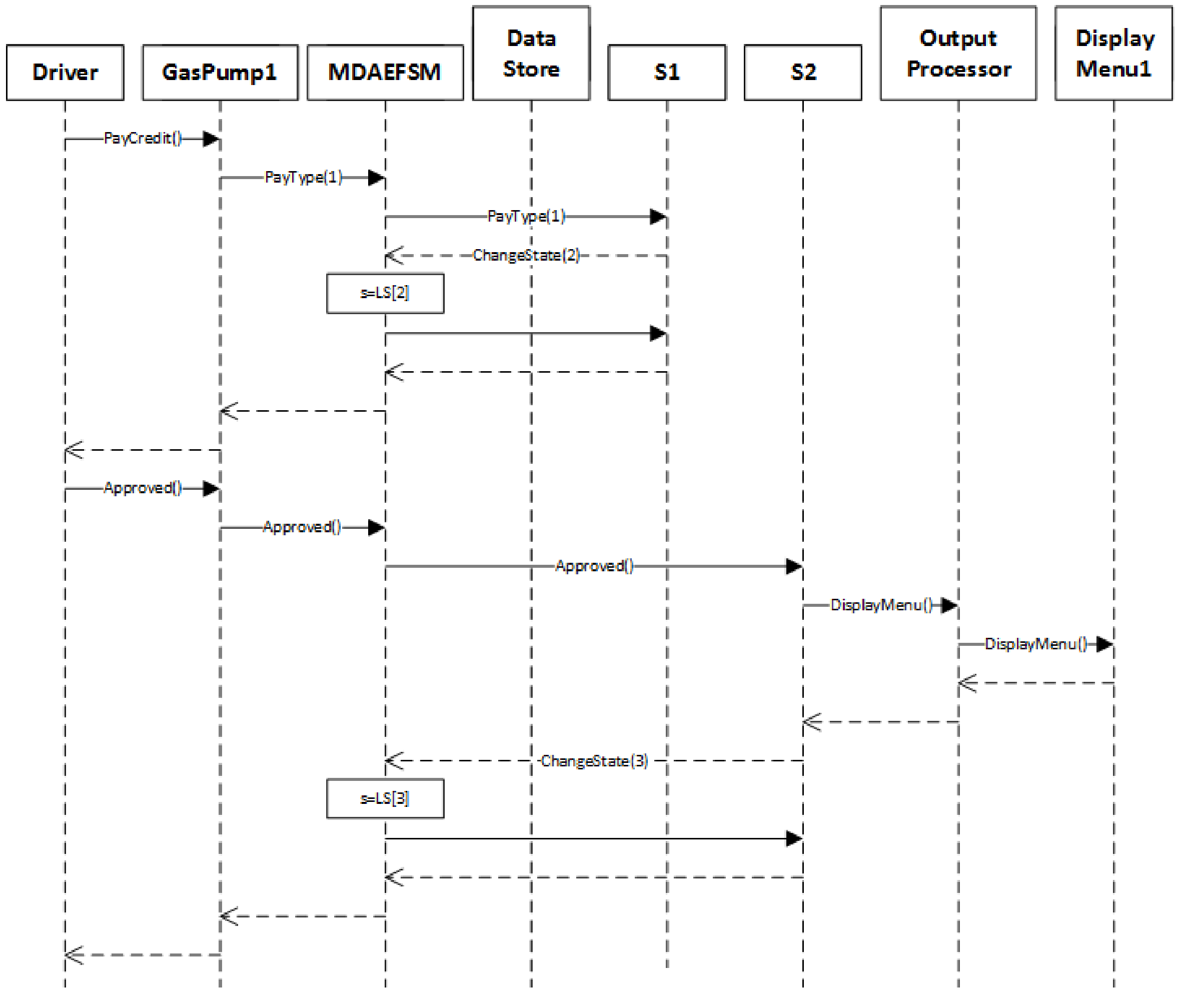
4.1.1. Operations: Activate(3.1,4.3)



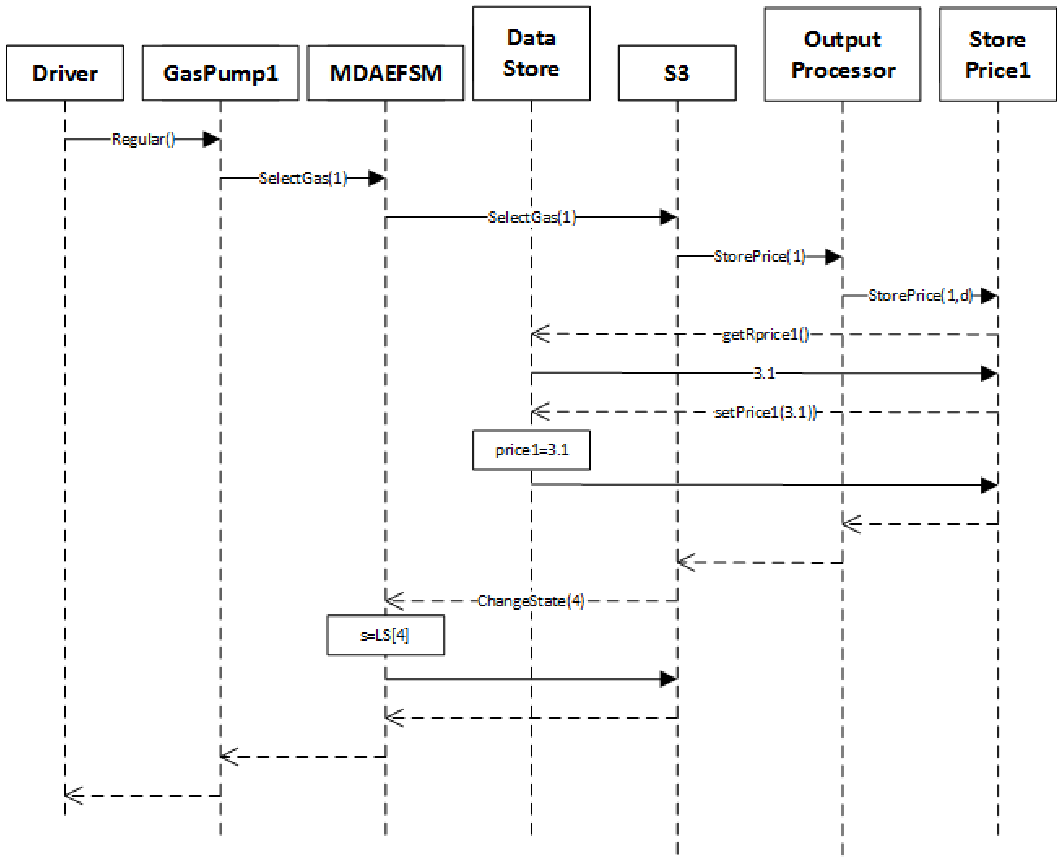
4.1.2. Operations: Start()



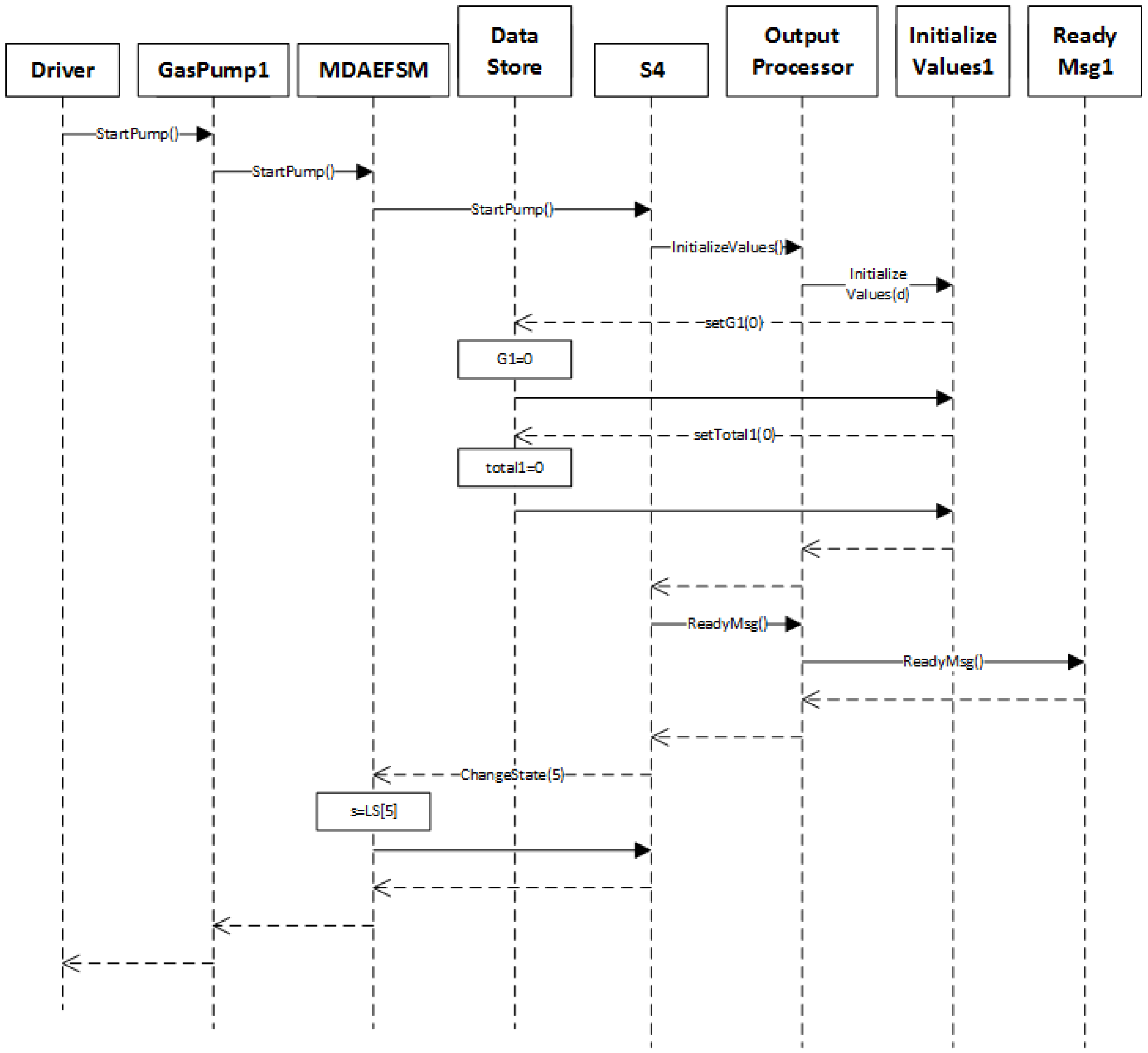
4.1.3. Operations: PayCredit(), Approved()



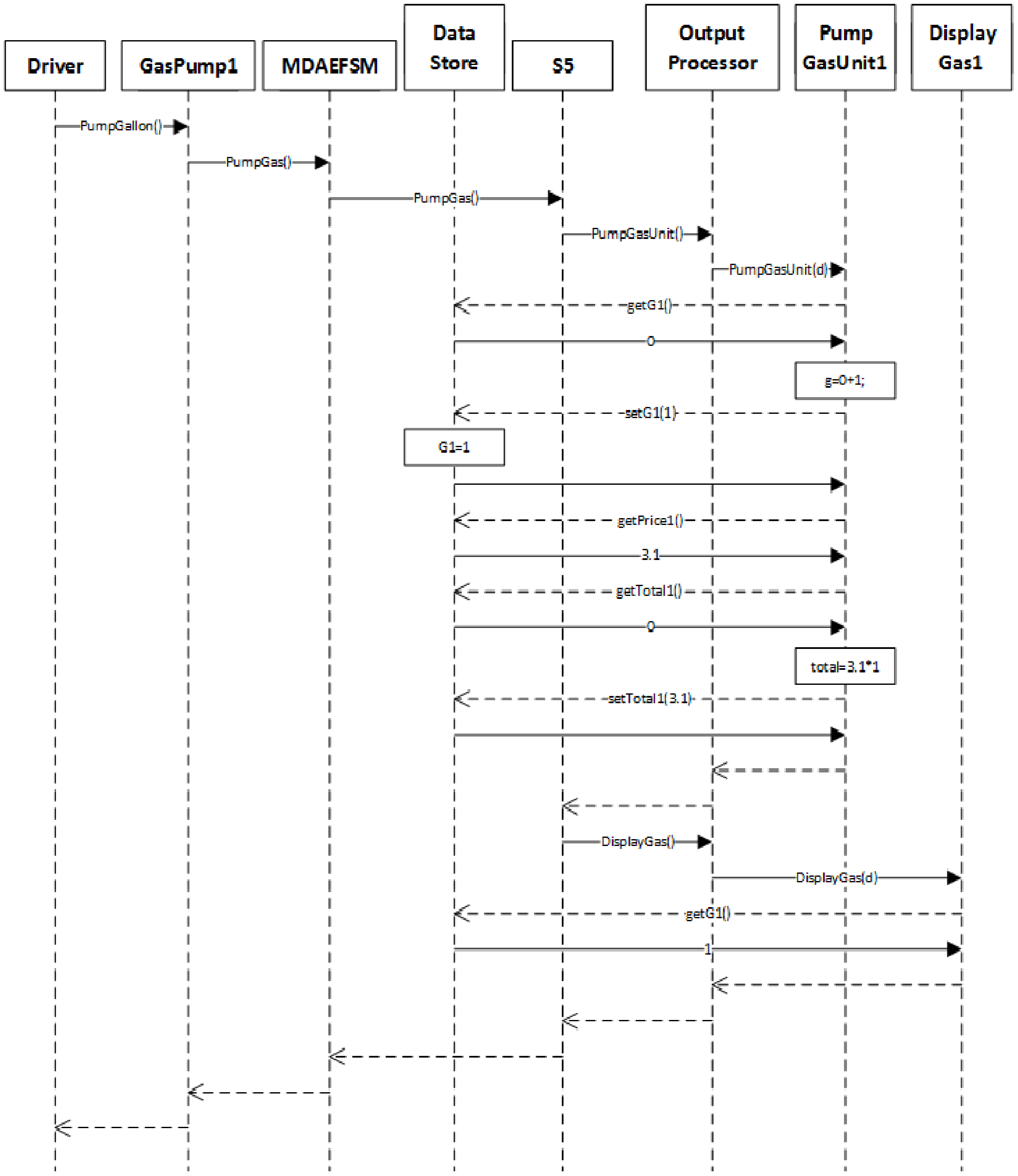
4.1.4. Operations: Regular()



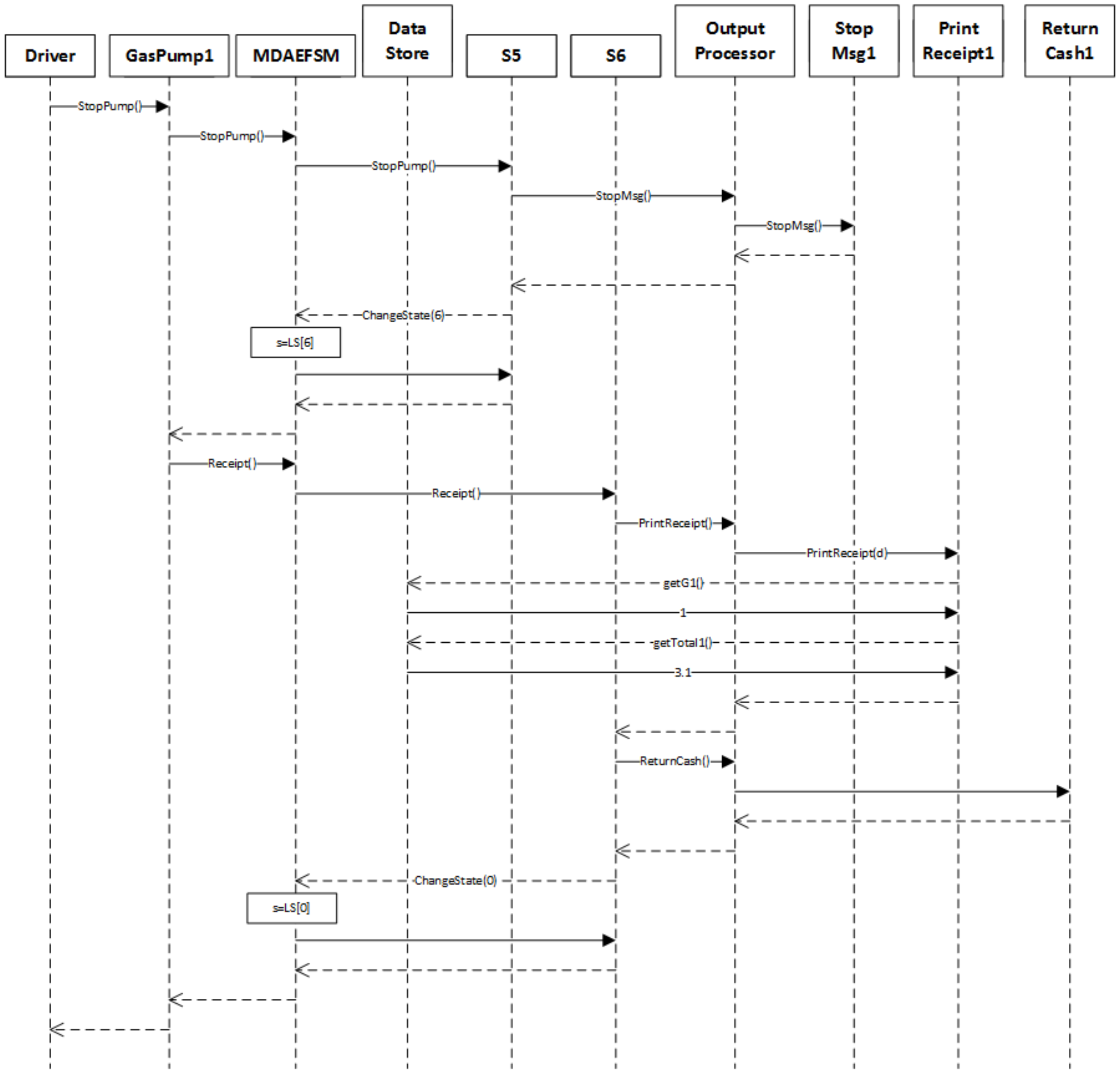
4.1.5. Operations: StartPump()



4.1.6. Operations: PumpGallon()

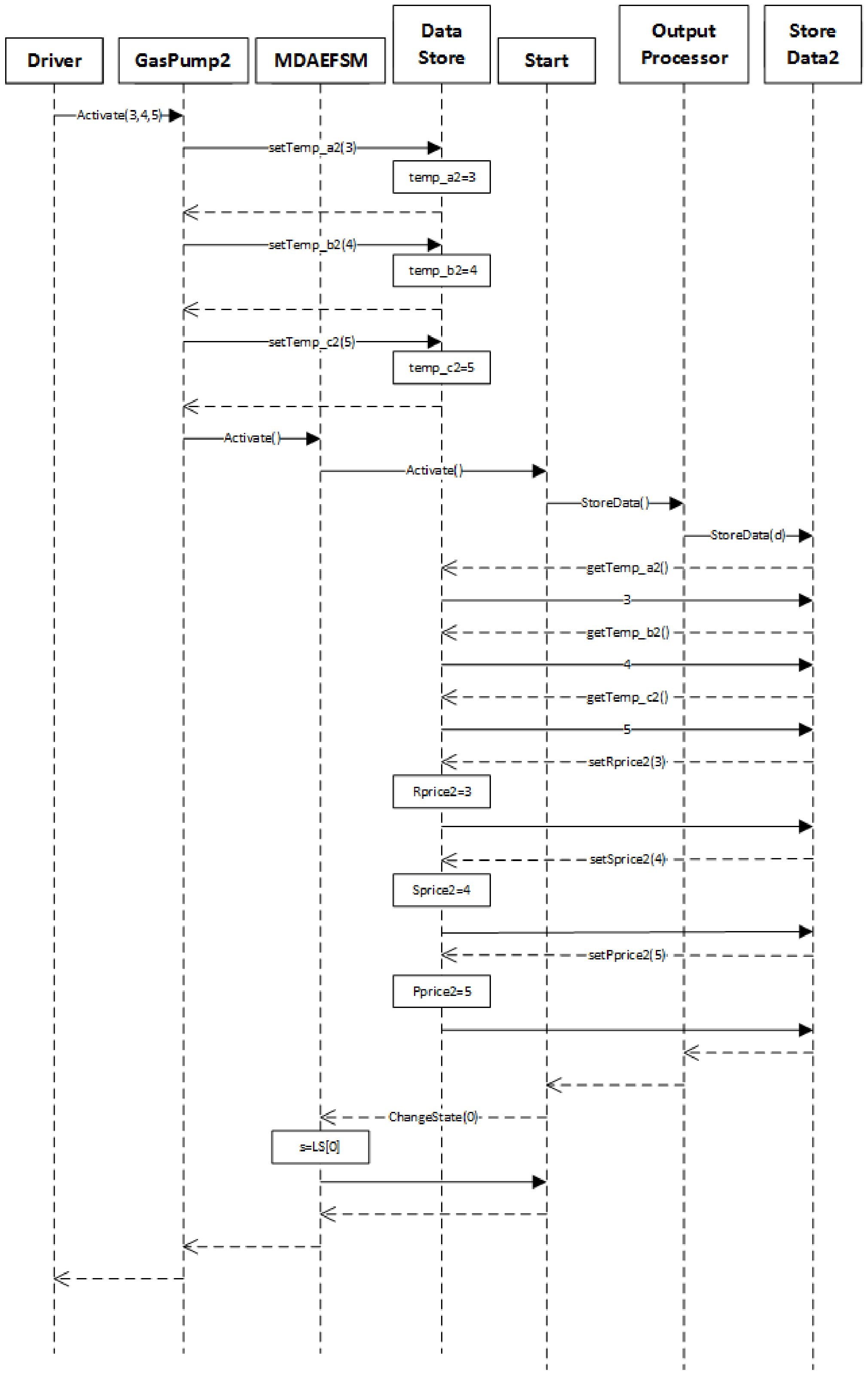


4.1.7. Operations: StopPump()

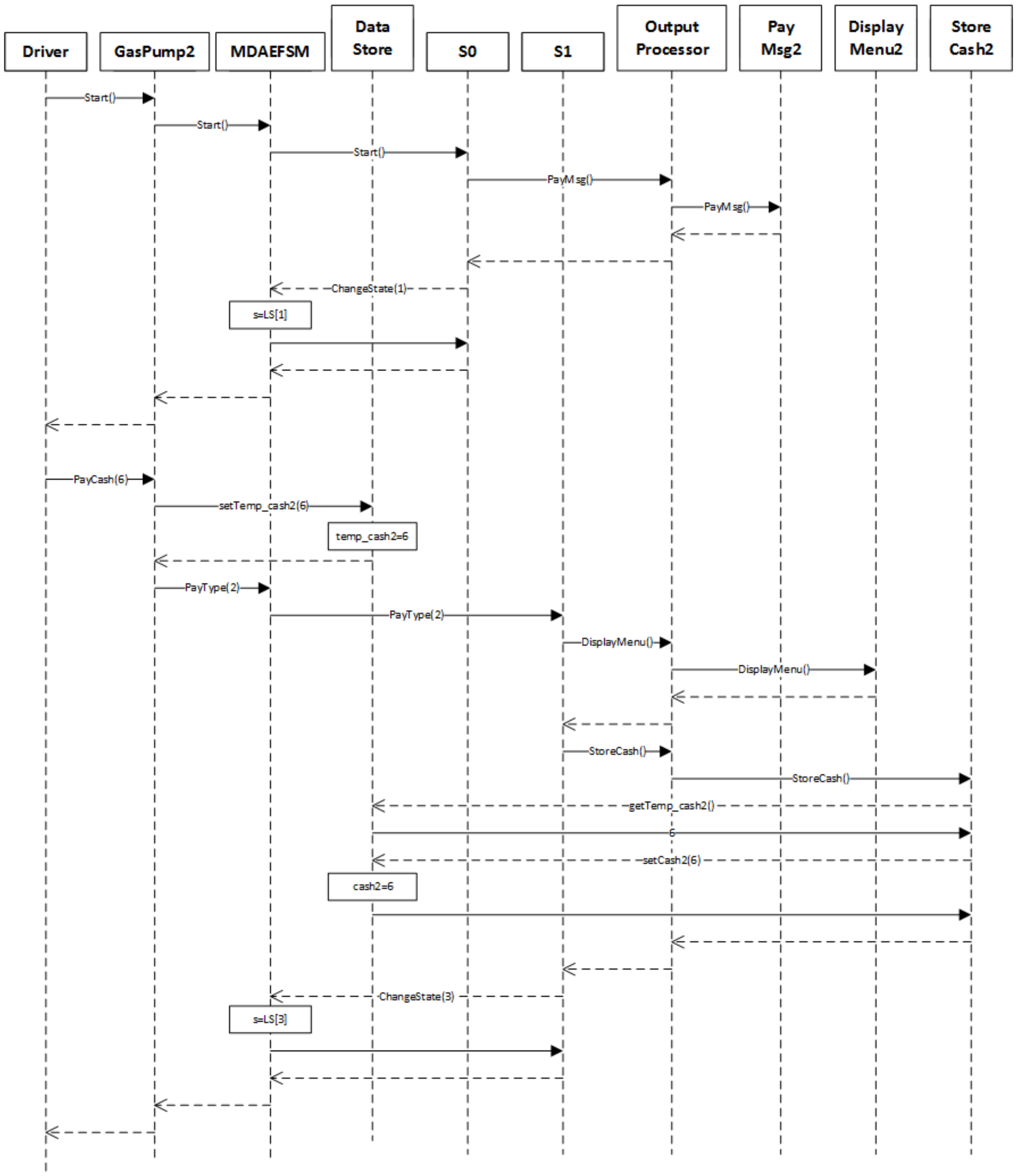


4.2. Scenario 2 Sequence Diagram

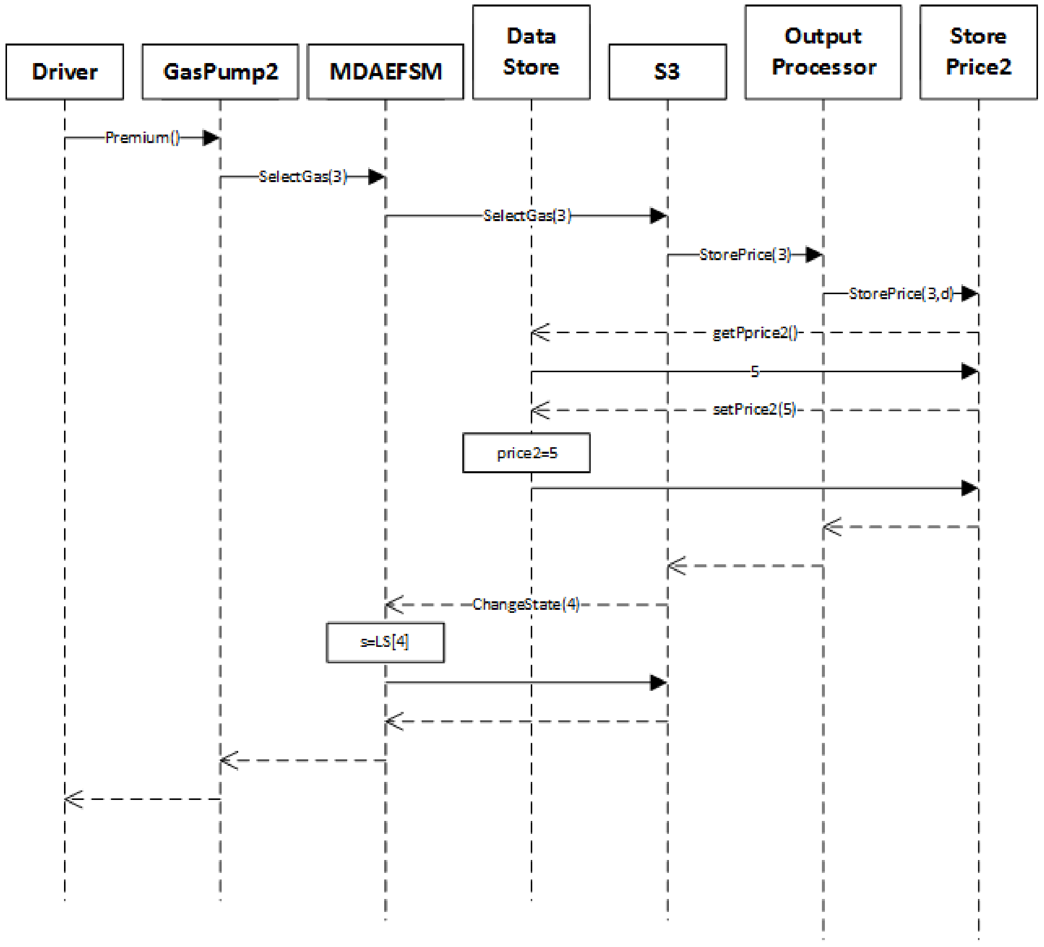
4.2.1. Operations: Activate(3,4,5)



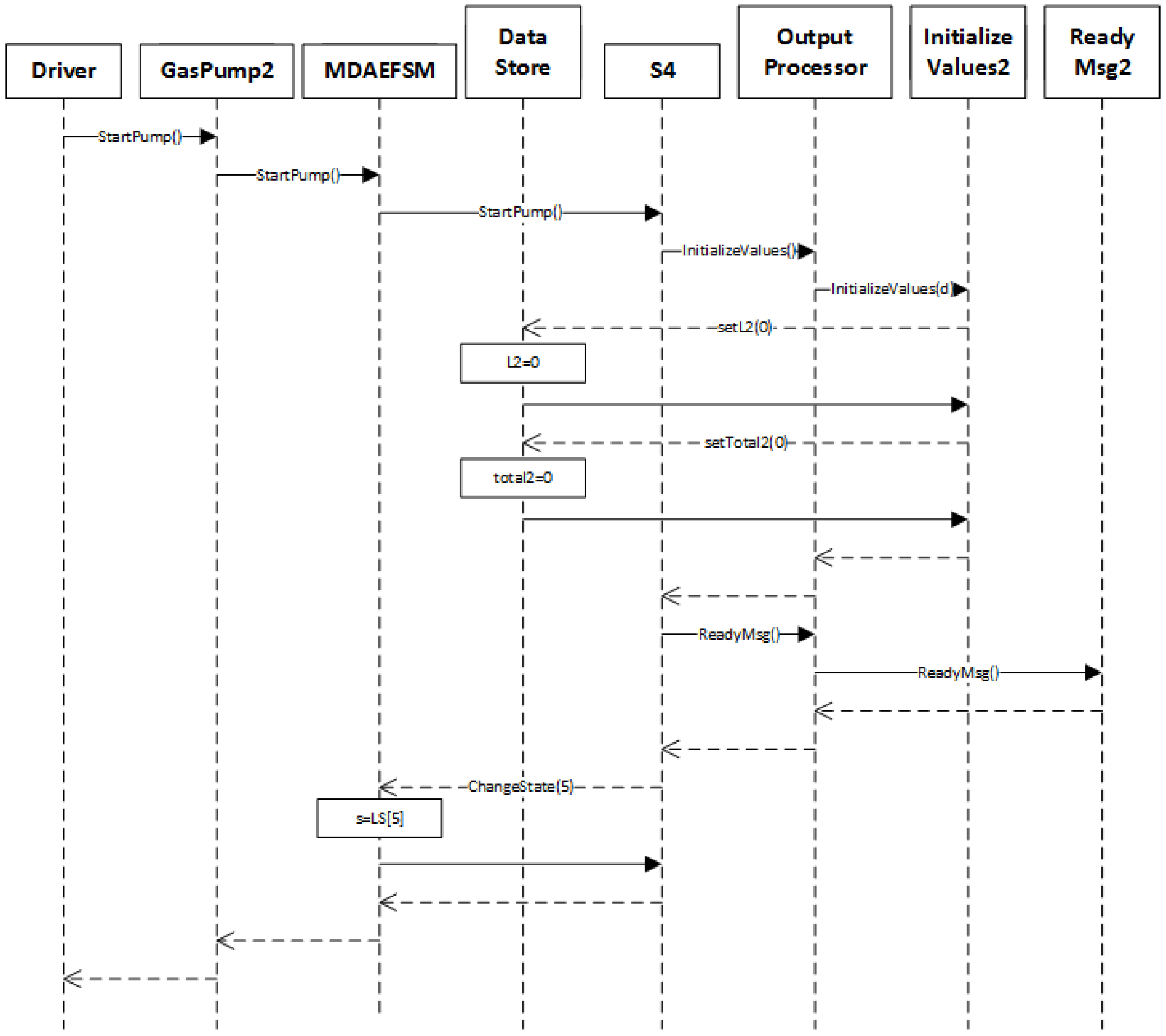
4.2.2. Operations: Start(), PayCash(6)



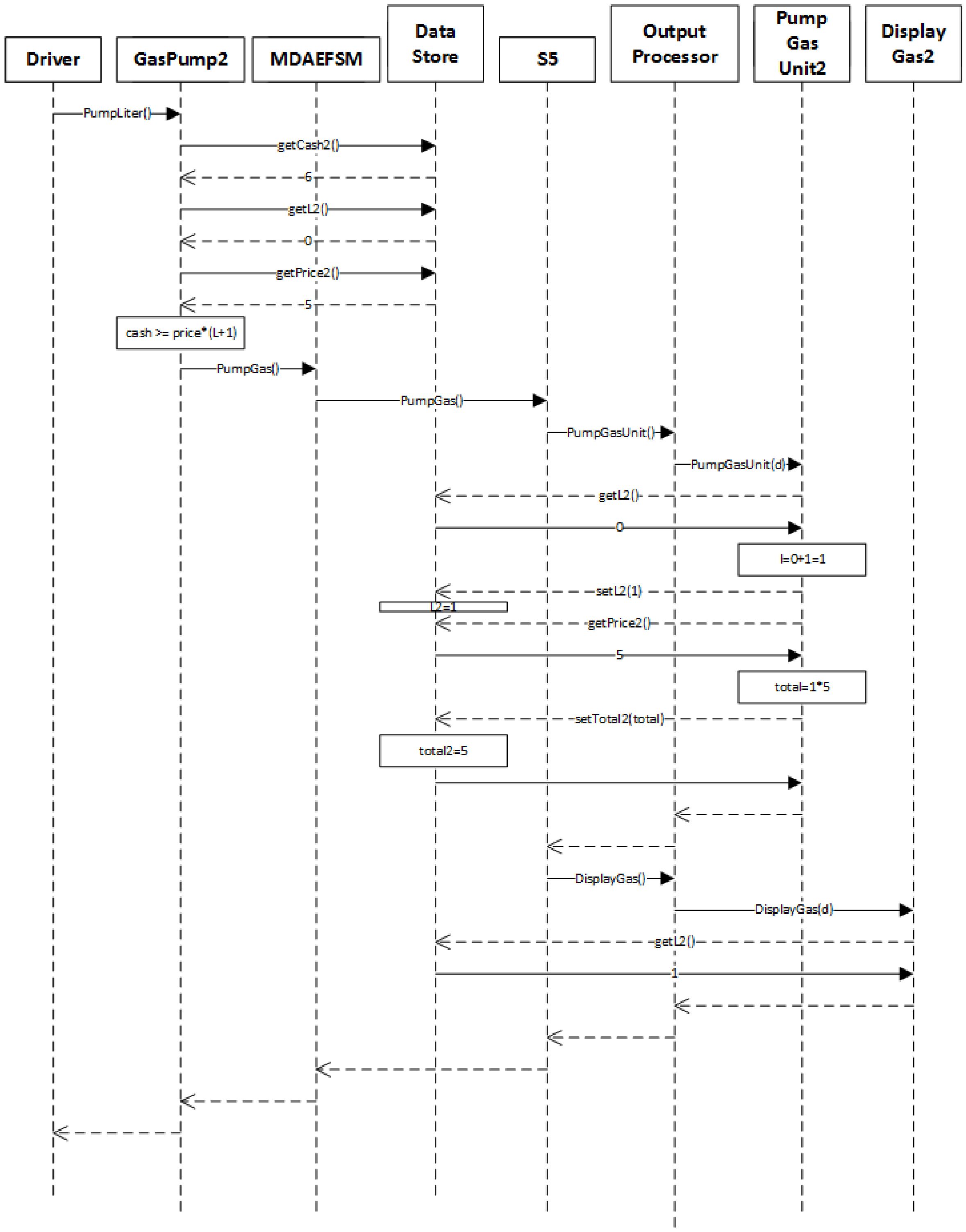
4.2.3. Operations: Premium()



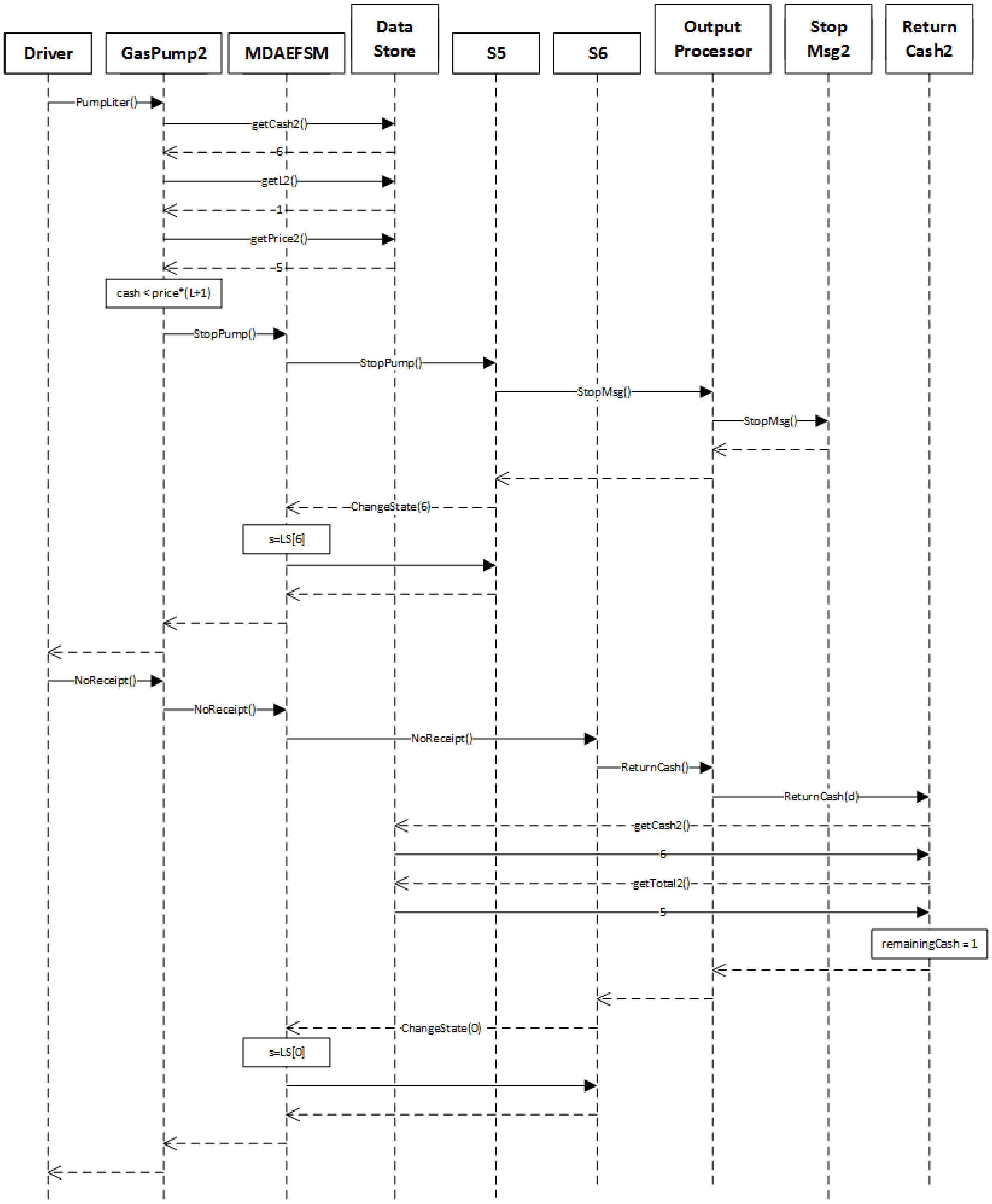
4.2.4. Operations: StartPump()



4.2.5. Operations: PumpLiter()



4.2.6. Operations: PumpLiter(), NoReceipt()



5. Source Code and Patterns:

The source code has been divided into multiple java packages inside which the classes linked to each other are stored. There are also three packages for the three design patterns implemented in the source code. Below is the categorization of the design patterns with their respective packages and classes:

5.1. State Pattern: The classes inside the packages 'MDA' and 'MDA.StatePattern' are all a part of the state pattern.

5.2. Abstract Factory Pattern: The classes inside the package 'AbstractFactoryPackage' are all a part of the AbstractFactory design pattern.

5.3. Strategy Pattern: The classes inside the packages 'OutputProcessor' and 'OutputProcessor.Strategy' are all a part of the Strategy design pattern.