Name: Brijesh Mavani
CWID: A20406960
University: Illinois Institute of Technology
Course: Software System Architectures (CS 586)
Project Report

Table of Content

MDA-EFSM model for the GasPump components:

The MDS-EFSM model provided by professor has been used.

A list of meta events for the MDA-EFSM:

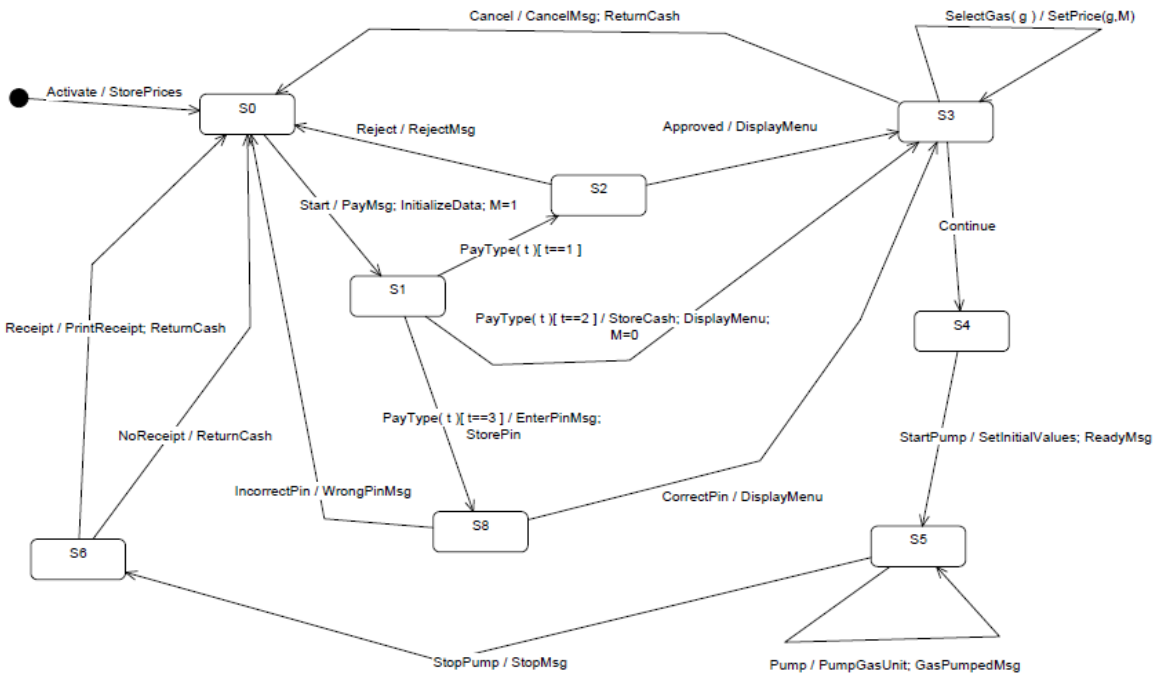
Meta-Events	Comments
Activate()	
Start()	
PayType(int t)	//credit: t=1; cash: t=2; debit: t=3
Reject()	
Cancel()	
Approved()	
StartPump()	
Pump()	
StopPump()	
SelectGas(int g)	// Regular: g=1; Super: g=2; Premium: g=3; Diesel: g=4
Receipt()	
NoReceipt()	
CorrectPin()	
IncorrectPin()	
Continue()	

A list of meta-actions for the MDA-EFSM with their descriptions

MDA-EFSM Actions	Role
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, 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
ReadyMsg	displays the ready for pumping message
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
StopMsg	stop pump message and receipt? msg (optionally)
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 and cash to 0

A state diagram of the MDA-EFSM:



MDA-EFSM for Gas Pumps

Pseudo-code of all operations of Input Processors of GasPump-1 and GasPump-2:

GasPump-1:

Operations of the Input Processor (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();
}

PayDebit(string p) {
        d->temp_p=p;
        m->PayType(3);
}

Pin(string x) {
if (d->pin==x) {m->CorrectPin();}
else{ m->InCorrectPin();}
}

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

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

Diesel() {
        m->SelectGas(4)
}

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

StartPump() {
if (d->price>0) {
        m->Continue();
        m->StartPump();
        }
}

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

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

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

Notice:
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

GasPump-2:

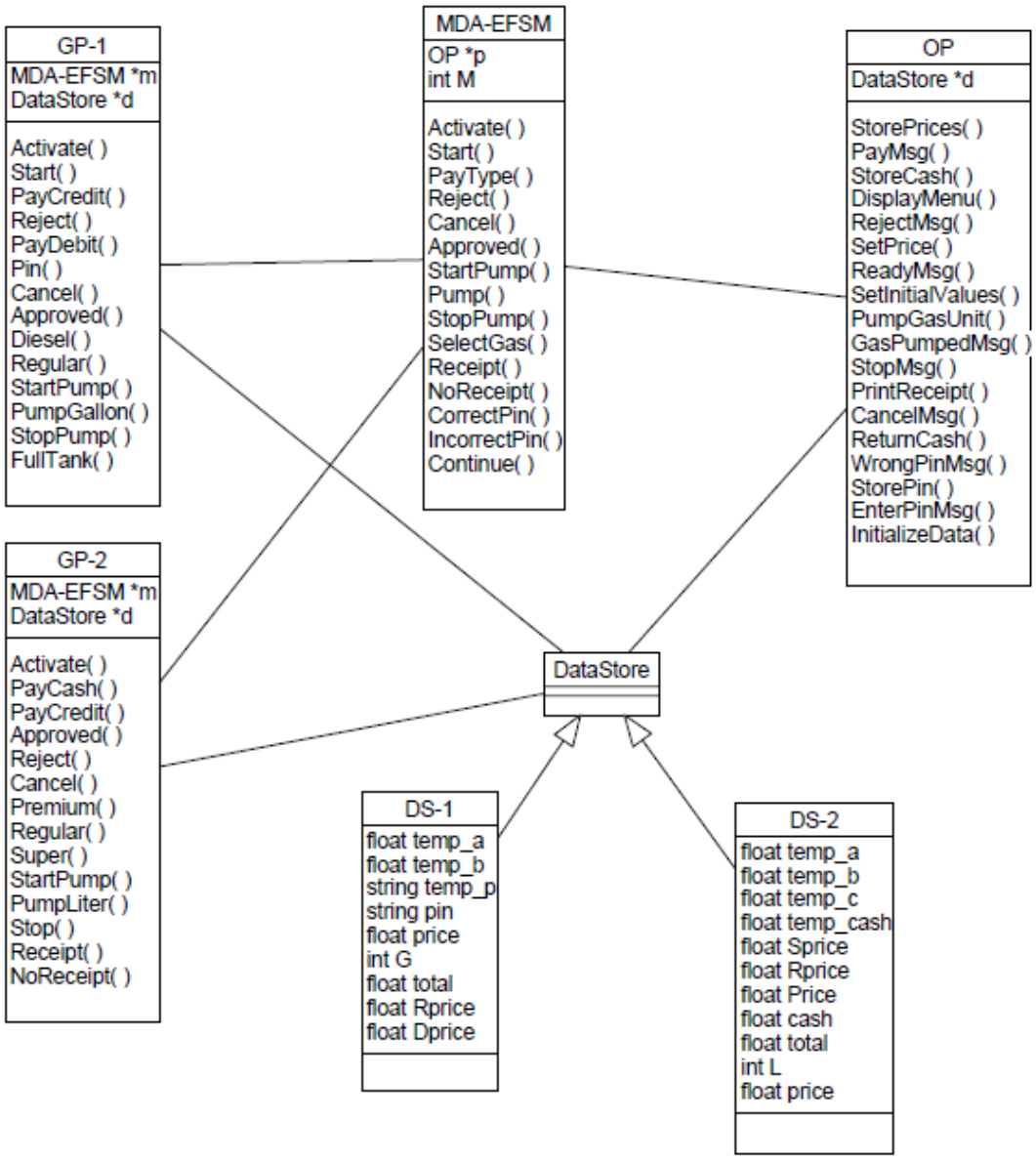
Operations of the Input Processor (GasPump-2)

```
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->Activate();
        }
}

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

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

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

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

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

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

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

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

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

PumpLiter() {
if (d->cash>0)&&(d->cash < d->price*(d->L+1)) {
        m->StopPump(); }
else {m->Pump();}
}

Stop() {
        m->StopPump();
}

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

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

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already Pumped

cash , L, price are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

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

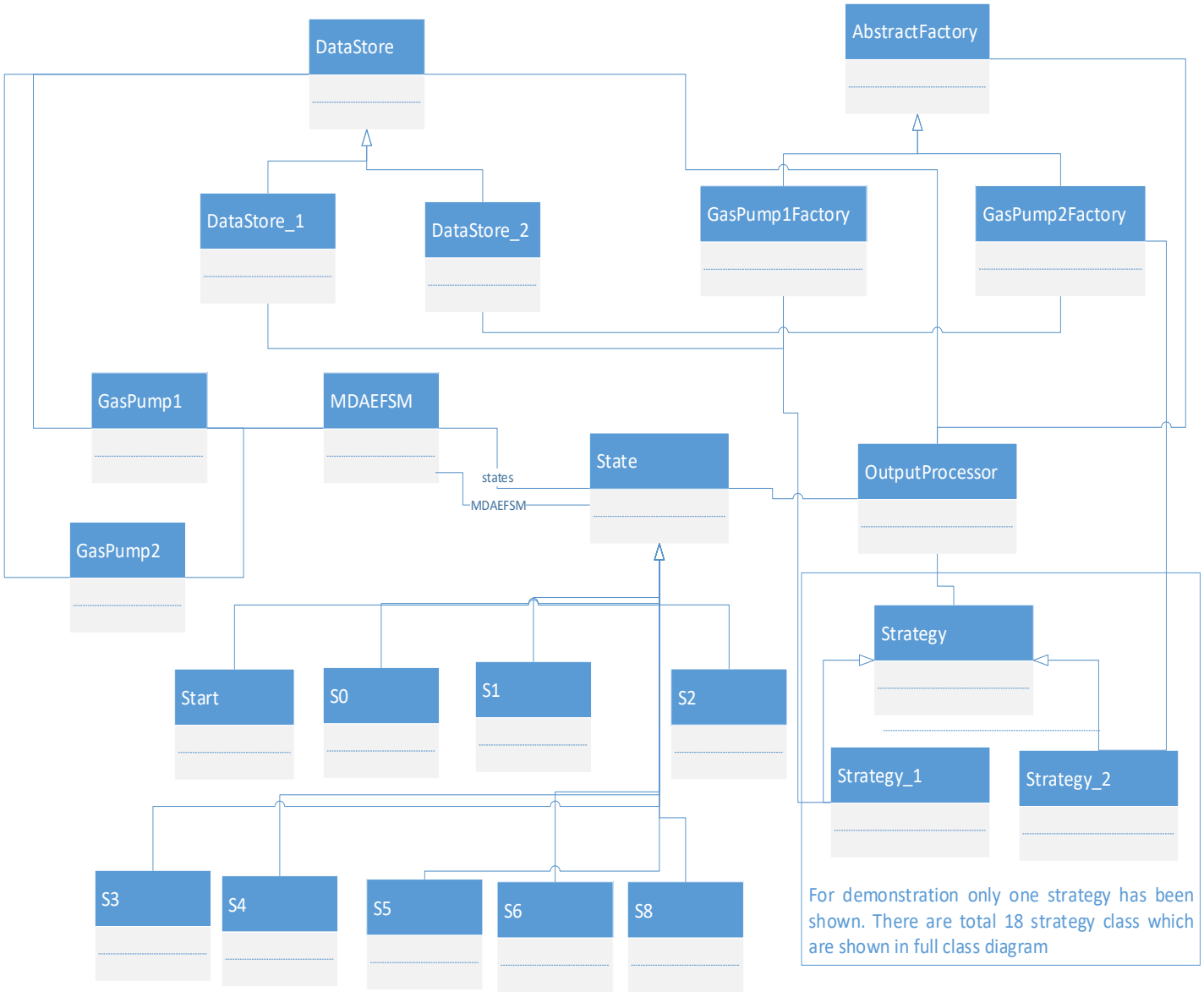


Class diagram(s) of the MDA of the GasPump components:

As class diagram is too big to fit into one page which can be easily visible, I have divided it into 3 sections based on design patterns which we are implementing.

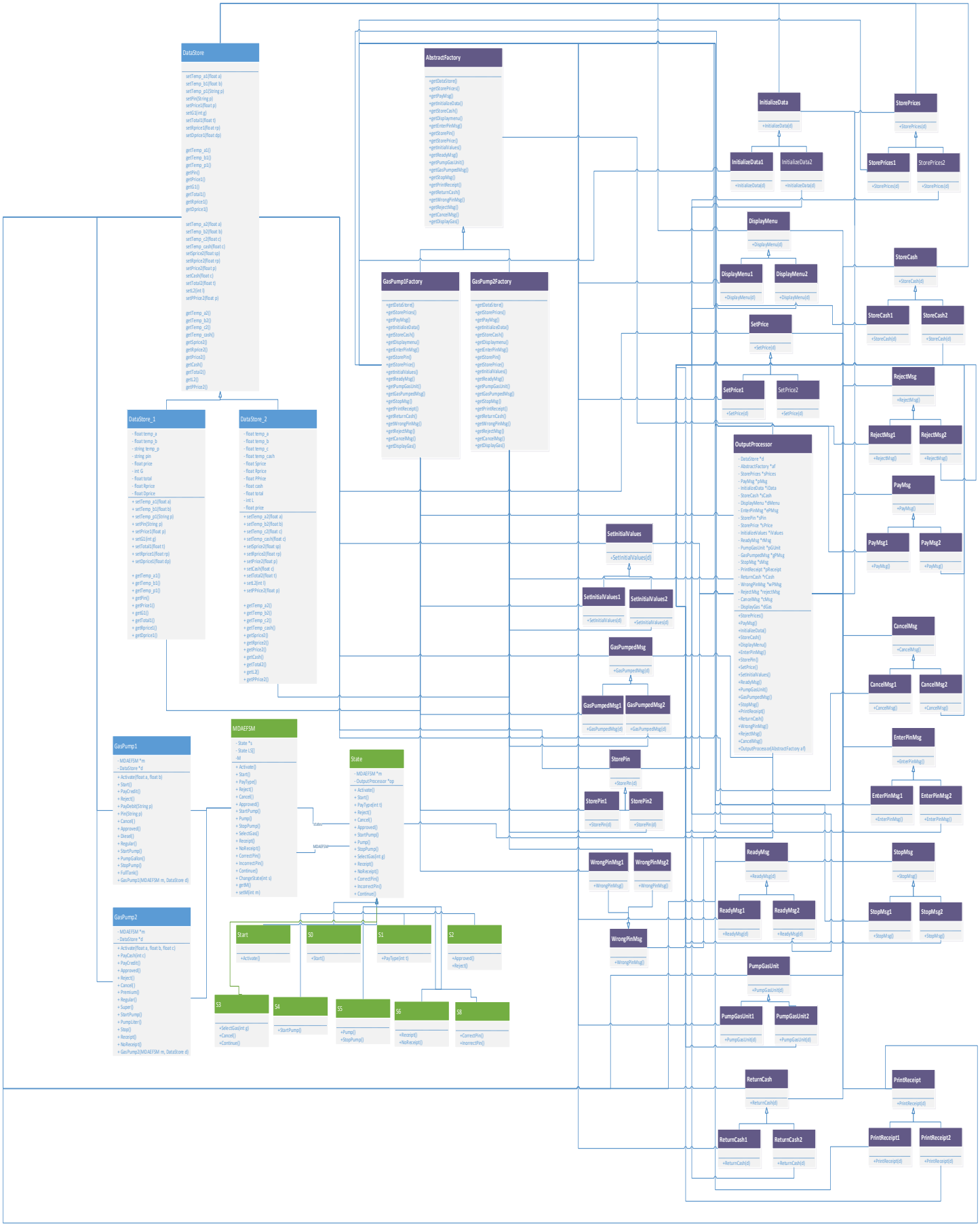
General High-Level Class Diagram:

The general high-level class diagram has been provided without mentioning about variables and functions. This is just for a demonstration purpose. The full class diagram with all class and its functionality will be provided in further sections.



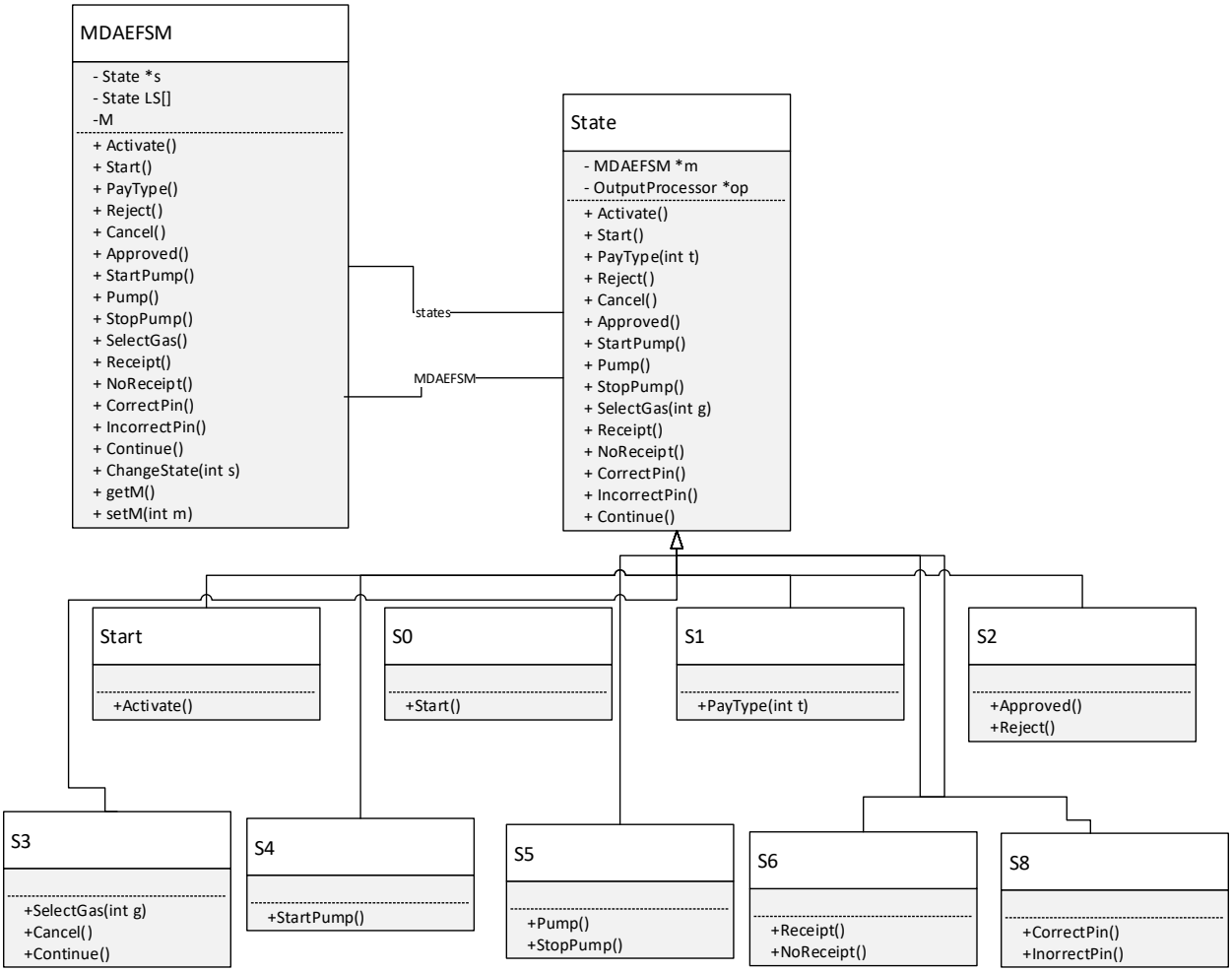
Name: Brijesh Mavani CWID: A20406960 Project Report

Complete Class Diagram: Please zoom to 150/200% to have a clear view



State pattern:

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



The classes belong to this pattern are highlighted in **green** color in full class diagram,

Abstract factory pattern and Strategy pattern:

As there are many classes, no separate class diagram has been created, the classes belong to this pattern are highlighted in purple color in full class diagram.

Purpose and Responsibilities:

Class AbstractFactory (abstract class)

Purpose of the class: This is a superclass for all abstract factory classes. It provides the abstract methods which must be implemented by the child abstract factory classes concrete factory classes.

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

Class AbstractFactory1 (For GasPump1)

Purpose of the class: It provides the required driver objects for Gas Pump 1. It Instantiates the proper action strategies with the shared data structure. The methods provided by this class will be called by output processor to bind actions specific to Gas Pump 1.

Responsibility of each operation supported by each class: It has following operations:

Operation	Responsibility
getDataStore	returns an object of DataStore1
getStorePrices	returns an object of StorePrices1
getPayMsg	returns an object of PayMsg1
getInitializeData	returns an object of InitializeData1
getStoreCash	returns an object of StoreCash1
getDisplayMenu	returns an object of DisplayMenu1
getEnterPinMsg	returns an object of EnterPinMsg1
getStorePin	returns an object of StorePin1
getSetPrice	returns an object of SetPrice1
getSetInitialValues	returns an object of SetInitialValues1
getReadyMsg	returns an object of ReadyMsg1
getPumpGasUnit	returns an object of PumpGasUnit1
getGasPumpedMsg	returns an object of GasPumpedMsg1
getStopMsg	returns an object of StopMsg1
getPrintReceipt	returns an object of PrintReceipt1
getreturnsCash	returns an object of returnsCash1
getWrongPinMsg	returns an object of WrongPinMsg1
getRejectMsg	returns an object of RejectMsg1
getCancelMsg	returns an object of CancelMsg1

Class AbstractFactory2 (For GasPump2)

Purpose of the class: It provides the required driver objects for Gas Pump 2. It Instantiates the proper action strategies with the shared data structure. The methods provided by this class will be called by output processor to bind actions specific to Gas Pump 2.

Responsibility of each operation supported by each class: It has following operations:

Operation	Responsibility
getDataStore	returns an object of DataStore2
getStorePrices	returns an object of StorePrices2
getPayMsg	returns an object of PayMsg2
getInitializeData	returns an object of InitializeData2
getStoreCash	returns an object of StoreCash2
getDisplayMenu	returns an object of DisplayMenu2
getEnterPinMsg	returns an object of EnterPinMsg2
getStorePin	returns an object of StorePin2
getSetPrice	returns an object of SetPrice2
getSetInitialValues	returns an object of SetInitialValues2
getReadyMsg	returns an object of ReadyMsg2
getPumpGasUnit	returns an object of PumpGasUnit2
getGasPumpedMsg	returns an object of GasPumpedMsg2
getStopMsg	returns an object of StopMsg2
getPrintReceipt	returns an object of PrintReceipt2
getreturnsCash	returns an object of returnsCash2
getWrongPinMsg	returns an object of WrongPinMsg2
getRejectMsg	returns an object of RejectMsg2
getCancelMsg	returns an object of CancelMsg2

Class DataStore (abstract class)

Purpose of the class: This is an abstract class.

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

Class DataGasPump1

Purpose of the class: This is a concrete Data Store for GasPump1. It contains temporary and final variables.

Responsibility of each operation supported by each class: All operations are getters and setter for getting and setting variables respectively.

Class DataGasPump2

Purpose of the class: This is a concrete Data Store for GasPump2. It contains temporary and final variables.

Responsibility of each operation supported by each class: All operations are getters and setter for getting and setting variables respectively.

Class GasPump

Purpose of the class: This is an abstract super class of the Input Processor for the MDA Architecture. It provides the constructor which will be used by child classes to build their own driver and required objects.

Responsibility of each operation supported by each class: It has constructor which returns the objects. All other operations are abstract operations.

Class GasPump1

Purpose of the class: It is an input processors or MDA architecture design. Operations which can be performed on GasPump1 are called in this class.

Responsibility of each operation supported by each class:

Operation	Responsibility
Activate(float a, float b)	if values are > 0, Store data into temp variables and calls Activate() method of MDAEFSM
Start()	Calls start() method of MDAEFSM
PayCredit()	Calls payType() method of MDAEFSM with input as 1
PayDebit(String p)	Check the pin length and store pin to the main variable from temp variable and calls payType() method of MDAEFSM with input as 3
Pin(String p)	Compare the input pin with store pin and if matched calls correctPin() method of MDAEFSM else calls incorrectPin() method
Approved()	Calls approved() method of MDAEFSM
Reject()	Calls reject() method of MDAEFSM
Cancel()	Calls cancel() method of MDAEFSM
Regular()	Calls selectGas() method of MDAEFSM with input as 1
Diesel()	Calls selectGas() method of MDAEFSM with input as 4
StartPump()	If price is >0 calls Continue() and startPump() methods of MDAEFSM
PumpGallon()	Calls pump() method of MDAEFSM
StopPump()	Calls stopPump() and receipt() methods of MDAEFSM
FullTank()	Calls stopPump() and receipt() methods of MDAEFSM

Class GasPump2

Purpose of the class: It is an input processor or MDA architecture design. Operations which can be performed on GasPump2are called in this class.

Responsibility of each operation supported by each class:

Operation	Responsibility
Activate(float a, float b, float c)	if values are > 0 , Store data into temp variables and calls Activate() method of MDAEFSM
Start()	Calls start() method of MDAEFSM
PayCredit()	Calls start() and payType(1) methods of MDAEFSM
PayCash(float c)	Check the cash value. If >0 then store it into data store. Calls start() and payType(2) methods of MDAEFSM
Approved()	Calls approved() method of MDAEFSM
Reject()	Calls reject() method of MDAEFSM
Cancel()	Calls cancel() method of MDAEFSM
Regular()	Calls selectGas() method of MDAEFSM with input as 1
Super()	Calls selectGas() method of MDAEFSM with input as 2
Premium()	Calls selectGas() method of MDAEFSM with input as 3
StartPump()	Calls startPump() method of MDAEFSM
PumpLiter()	If the payment type is cash(m==0), check if there is a sufficient cash for pumping one liter of gas. If yes call pump() function of MDAEFSM else call stopPump(). If the payment type is credit call pump().
Stop()	Calls stopPump() method of MDAEFSM
Receipt()	Calls receipt() method of MDAEFSM
NoReceipt()	Calls noReceipt() method of MDAEFSM

Class GasPumpDriver

Purpose of the class: It implements the main class for GasPump System.

Responsibility of each operation supported by each class: It implements the main method for GasPump System. It takes input from the user about operation it likes to perform and call a respective operation on GasPump1/ GasPump2.

Class MDAEFSM

Purpose of the class: It 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 supported by each class:

Operation	Responsibility
activate()	Passes call to the current state's activate() method

start()	Passes call to the current state's start() method
payType(int t)	Passes call to the current state's payType(int t) method
approve()	Passes call to the current state's approve() method
reject()	Passes call to the current state's reject() method
cancel()	Passes call to the current state's cancel() method
selectGas(int g)	Passes call to the current state's selectGas(int g) method
startPump()	Passes call to the current state's startPump() method
pump()	Passes call to the current state's pump() method
stopPump()	Passes call to the current state's stopPump() method
receipt()	Passes call to the current state's receipt() method
noReceipt()	Passes call to the current state's noReceipt() method
correctPin()	Passes call to the current state's correctPin() method
incorrectPin()	Passes call to the current state's incorrectPin() method
Continue()	Passes call to the current state's Continue() method
getM()	Returns the value of M
setM(int m)	Sets value of M = m

Class S0

Purpose of the class: Provides an implementation of State S0

Responsibility of each operation supported by each class:

Operation	Responsibility
start()	perform actions initialize Data and M, calls PayMsg () on Output Processor then change state to S1.

Class S1

Purpose of the class: Provides an implementation of State S1

Responsibility of each operation supported by each class:

Operation	Responsibility
payType(int t)	if t ==1, change state to S2. else if t==2, calls StoreCash() and DisplayMenu() on output processor. Then change state to S3. if t==3, calls EnterPinMsg() and StorePin() on output processor and then changes state to S8

Class S2

Purpose of the class: Provides an implementation of State S2

Responsibility of each operation supported by each class:

Operation	Responsibility
approve()	Calls DisplayMenu() on output processor. Then change state to S3.
reject()	Calls RejectMsg() on output processor. Then change state to S0.

Class S3

Purpose of the class: Provides an implementation of State S3

Responsibility of each operation supported by each class:

Operation	Responsibility
selectGas(int g)	Fetches M and then Calls SetPrice(g,m) on output processor
cancel()	Calls CancelMsg() and ReturnCash() on output processor and then changes state to S0

Class S4

Purpose of the class: Provides an implementation of State S4

Responsibility of each operation supported by each class:

Operation	Responsibility
startPump()	Calls SetInitialValues() and ReadyMsg() on output processor then changes state to S5

Class S5

Purpose of the class: Provides an implementation of State S5

Responsibility of each operation supported by each class:

Operation	Responsibility
pump()	Calls PumpGasUnit() and GasPumpedMsg() on output processor
stopPump()	Calls StopMsg() on output processor then changes state to S6

Class S6

Purpose of the class: Provides an implementation of State S6

Responsibility of each operation supported by each class:

Operation	Responsibility
receipt()	Calls PrintReceipt() and ReturnCash() on output processor then changes state to S0
noReceipt()	Calls ReturnCash() on output processor then changes state to S0

Class S8

Purpose of the class: Provides an implementation of State S8

Responsibility of each operation supported by each class:

Operation	Responsibility
incorrectPin()	Calls WrongPinMsg() on output processor then changes state to S0
correctPin()	Calls DisplayMenu() on output processor then changes state to S3

Class Start

Purpose of the class: Provides an implementation of State Start

Responsibility of each operation supported by each class:

Operation	Responsibility
activate()	Calls StorePrices() on output processor then changes state to S0

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 supported by each class: All operation are abstract operations.

Class OutputProcessor

Purpose of the class: This class is responsible for performing the actual operation on GasPump.

Responsibility of each operation supported by each class:

Operation	Responsibility
CancelMsg()	Calls cancelMsg() of class CancelMsg (may be CancelMsg 1 CancelMsg 2 depending on the GasPump component being used)
DisplayMenu()	Calls displayMenu() of class DisplayMenu (may be DisplayMenu 1 DisplayMenu 2 depending on the GasPump component being used)
GasPumpedMsg()	Calls gasPumpedMsg() of class GasPumpedMsg (may be GasPumpedMsg 1 GasPumpedMsg 2 depending on the GasPump component being used)
PayMsg()	Calls payMsg() of class PayMsg (may be PayMsg 1 PayMsg 2 depending on the GasPump component being used)
PrintReceipt()	Calls printReceipt() of class PrintReceipt (may be PrintReceipt 1 PrintReceipt 2 depending on the GasPump component being used)
PumpGasUnit()	Calls pumpGasUnit() of class PumpGasUnit (maybe PumpGasUnit 1 PumpGasUnit 2 depending on the GasPump component being used)
ReadyMsg()	Calls readyMsg() of class ReadyMsg (may be ReadyMsg 1 ReadyMsg 2 depending on the GasPump component being used)
RejectMsg()	Calls rejectMsg() of class RejectMsg (may be RejectMsg 1 RejectMsg 2 depending on the GasPump component being used)
ReturnCash()	Calls returnCash() of class ReturnCash (may be ReturnCash 1 ReturnCash 2 depending on the GasPump component being used)

SetInitialValues()	Calls setInitialValues() of class SetInitialValues (may be SetInitialValues 1 SetInitialValues 2 depending on the GasPump component being used)
SetPrice(int g,int m)	Calls setPrice(int g,int m) of class SetPrice (may be SetPrice 1 SetPrice 2 depending on the GasPump component being used)
StopMsg()	Calls stopMsg() of class StopMsg (may be StopMsg 1 StopMsg 2 depending on the GasPump component being used)
StoreCash()	Calls storeCash() of class StoreCash (may be StoreCash 1 StoreCash 2 depending on the GasPump component being used)
StorePrices()	Calls storePrices() of class StorePrices (may be StorePrices 1 StorePrices 2 depending on the GasPump component being used)
InitializeData()	Calls initializeData() of class InitializeData (may be InitializeData 1 InitializeData 2 depending on the GasPump component being used)
EnterPinMsg()	Calls enterPinMsg() of class EnterPinMsg (may be EnterPinMsg 1 EnterPinMsg 2 depending on the GasPump component being used)
WrongPinMsg()	Calls wrongPinMsg() of class WrongPinMsg (may be WrongPinMsg 1 WrongPinMsg 2 depending on the GasPump component being used)
StorePin()	Calls storePin() of class StorePin (may be StorePin 1 StorePin 2 depending on the GasPump component being used)

Class CancelMsg, CancelMsg1 and CancelMsg2

Purpose and Responsibility of the class: A set of classes used to perform the canceling of the GasPumps. It displays a cancel message in both gas pumps.

Class DisplayMenu, DisplayMenu1 and DisplayMenu2

Purpose and Responsibility of the class: A set of classes used to display available options to the user. DisplayMenu1 prints 2 options and DisplayMenu2 prints 3 options.

Class EnterPinMsg, EnterPinMsg1 and EnterPinMsg2

Purpose and Responsibility of the class: A set of classes used to display enter pin message to the user. EnterPinMsg1 prints the message while EnterPinMsg2 do not perform any action.

Class GasPumpedMsg, GasPumpedMsg1 and GasPumpedMsg2

Purpose and Responsibility of the class: 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 InitializeData, InitializeData1 and InitializeData2

Purpose and Responsibility of the class: A set of classes used to initialize the initial values to the DataStore. InitializeValues1 initializes price to 0 whereas InitializeValues2 initializes price and cash to 0.

Class PayMsg, PayMsg1 and PayMsg2

Purpose and Responsibility of the class: A set of classes used to display a pay message. PayMsg1 displays 2 options ‘PayCredit’ and ‘PayDebit’ whereas PayMsg2 displays 1 selection.

Class PrintReceipt, PrintReceipt1 and PrintReceipt2

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

Class PumpGasUnit, PumpGasUnit1 & PumpGasUnit2

Purpose and Responsibility of the class: A set of classes used to pump gas. PumpGasUnit1 pumps 1 gallon of selected gas whereas PumpGasUnit2 pumps 1 liter of selected gas.

Class ReadyMsg, ReadyMsg1 & ReadyMsg2

Purpose and Responsibility of the class: A set of classes used to warn the user by notifying him/her that pump is ready for pumping gas.

Class RejectMsg, RejectMsg1 & RejectMsg2

Purpose and Responsibility of the class: A set of classes used to display a reject message if a credit card is rejected. RejectMsg1 and RejectMsg1 both prints same reject message.

Class ReturnCash, ReturnCash1 & ReturnCash2

Purpose and Responsibility of the class: A set of classes used to return the remaining cash to the user. ReturnCash1 performs no operation as GP-1 do not support cash payments. Whereas, ReturnCash2 returns the remaining amount to the user.

Class SetInitialValues, SetInitialValues1 and SetInitialValues

Purpose and Responsibility of the class: A set of classes used to initialize the initial values to the DataStore. SetInitialValues1 initializes G and total to 0 whereas SetInitialValues2 initializes L and total to 0.

Class SetPrice, SetPrice1 & SetPrice2

Purpose and Responsibility of the class: A set of classes used to store the price of the selected gas type in the DataStore for further calculations. SetPrice1 and SetPrice2 both stores the price of the selected gas type in DataStore.

Class StopMsg, StopMsg1 & StopMsg2

Purpose and Responsibility of the class: 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 and Responsibility of the class: A set of classes used to store the cash entered by the user. ReturnCash1 performs no operation as GP-1 do not support cash payments. Whereas StoreCash2 stores the cash in the DataStore.

Class StorePin, StorePin1 & StorePin2

Purpose and Responsibility of the class: A set of classes used to store the data from temp variables into the main variables of DataStore. StorePin1 store the pin from temp variable to the main variable. StorePin2 do not perform any action as GP-2 do not have debit card payment method.

Class StorePrice, StorePrice1 & StorePrice2

Purpose and Responsibility of the class: A set of classes used to store the data from temp variables into the main variables of DataStore. StorePrice1 and StorePrice2 both save the respected initial data to the DataStore.

Class WrongPinMsg, WrongPinMsg1 & WrongPinMsg2

Purpose and Responsibility of the class: A set of classes used to display a wrong pin message to the user. WrongPinMsg1 displays message to the user that it has entered the wrong pin. WrongPinMsg2 do not perform any action as GP-2 do not have debit card payment method.

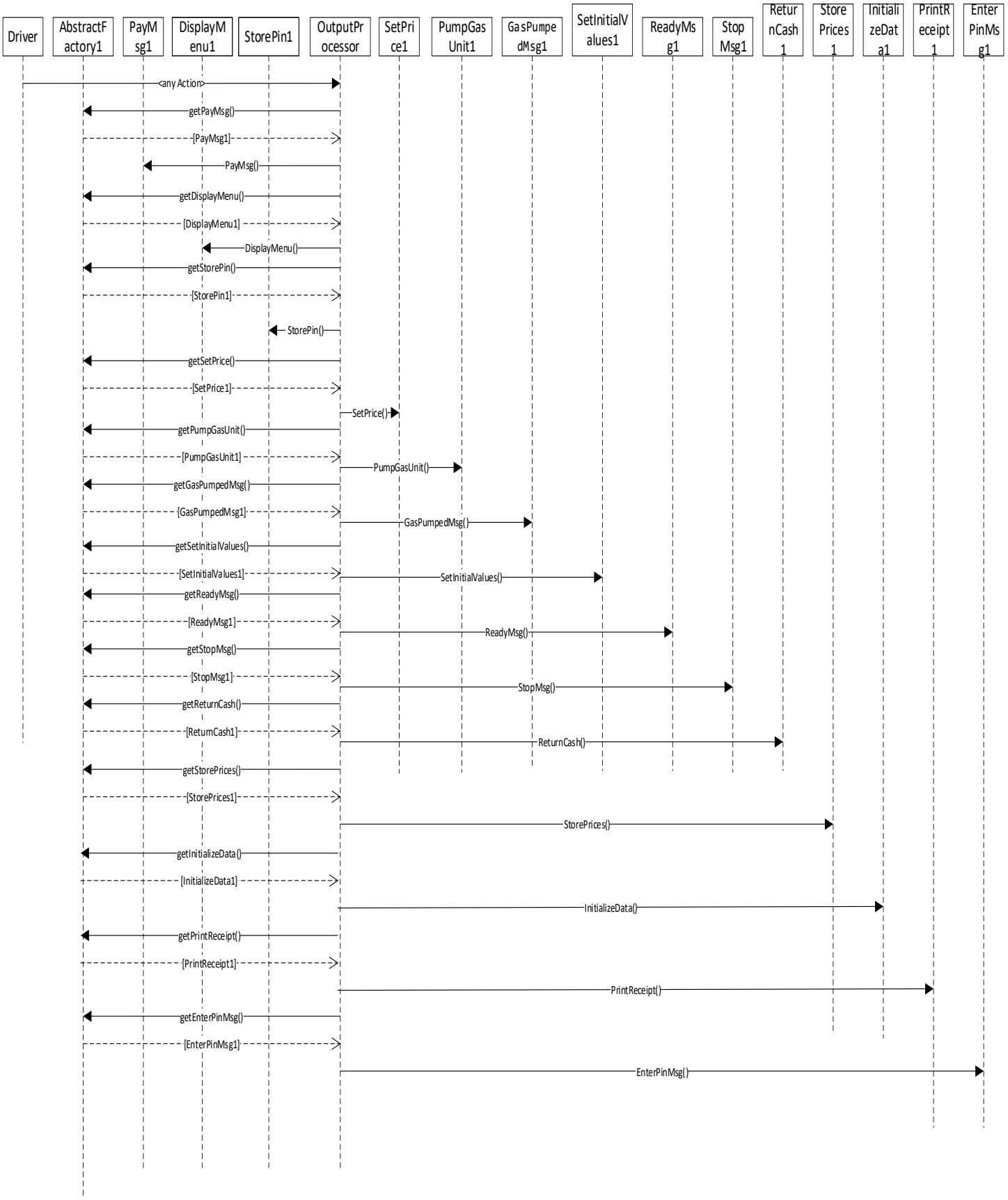
Sequence diagrams:

As sequence diagram is too big, it has been broken into various part to have a better visibility. Each part is responsible for one action. For scenario 1, total 9 diagrams have been created and provided. Starting from Initialization to FullTank(). For scenario 2, total 7 diagrams have been created and provided. Starting from Initialization to NoReceipt().

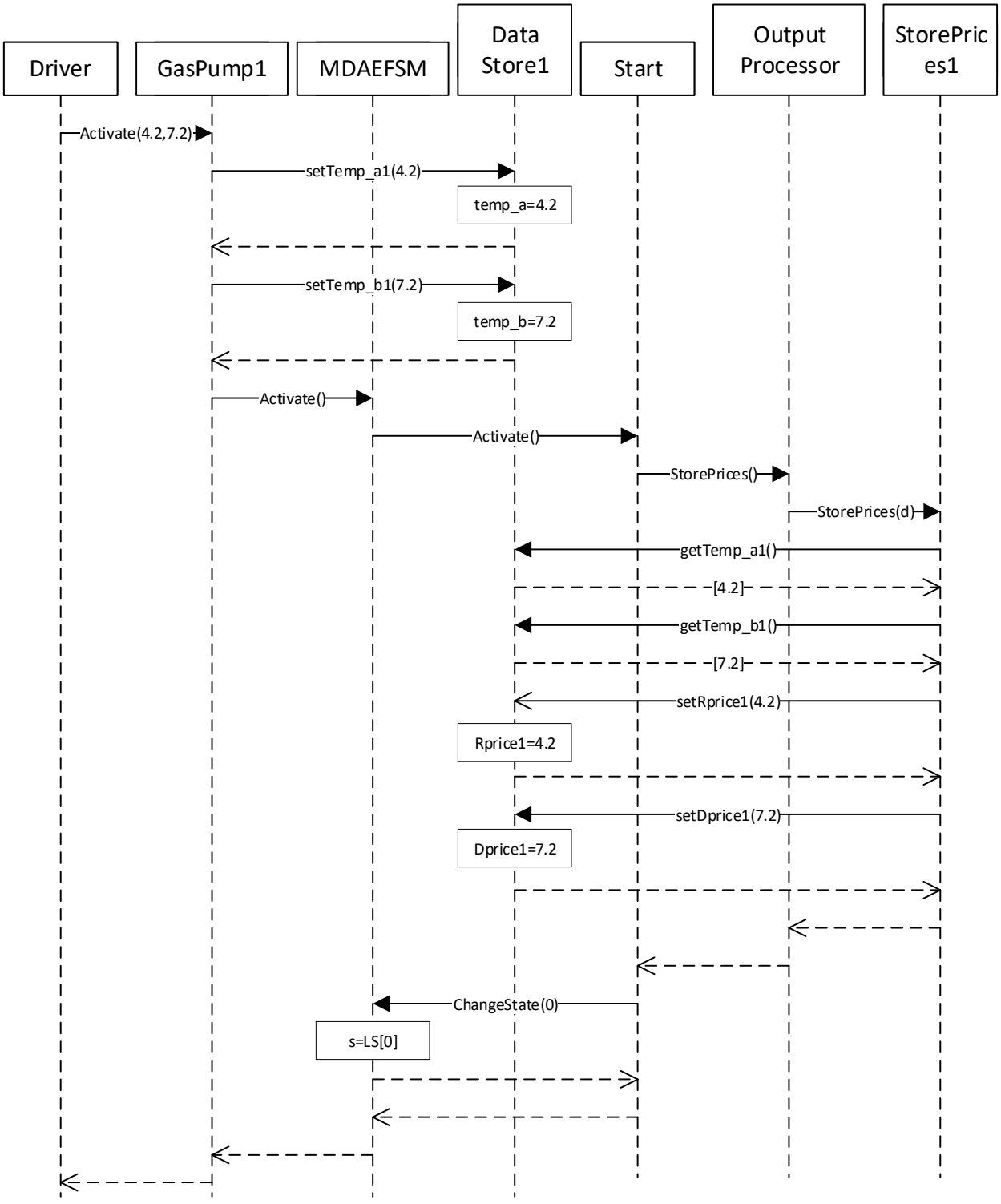
Scenario-I:

Initialization:

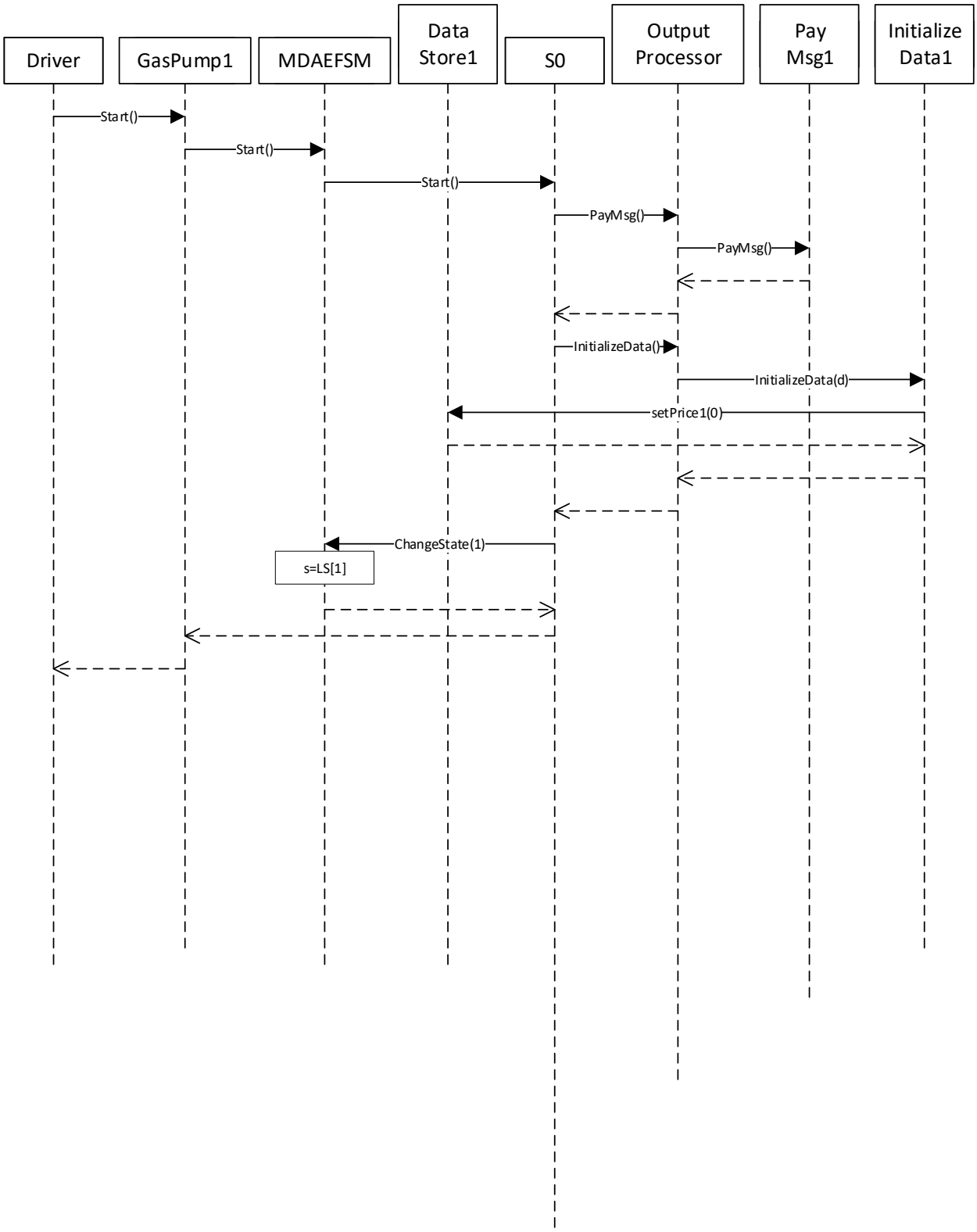
Following sequence diagram shows how Outputprocessor retrieves the objects from AbstractFactory. The diagram shows object creation for classes which are invoked in scenario1. Now, output processor has all the object it can invoke the methods directly. For demonstration only, method calls from outputprocessor are shown. Actual process will be shown in main sequence diagram.


Driver
AbstractFactory1
PayMsg1
DisplayMenu1
StorePin1
OutputProcessor
SetPrice1
PumpGasUnit1
GasPumpedMsg1
SetInitialValues1
ReadyMsg1
StopMsg1
ReturnCash1
StorePrices1
InitializeData1
PrintReceipt1
EnterPinMsg1
<any Action>
getPayMsg()
[PayMsg1]
PayMsg()
getDisplayMenu()
[DisplayMenu1]
DisplayMenu()
getStorePin()
[StorePin1]
StorePin()
getSetPrice()
[SetPrice1]
SetPrice()
getPumpGasUnit()
[PumpGasUnit1]
PumpGasUnit()
getGasPumpedMsg()
[GasPumpedMsg1]
GasPumpedMsg()
getSetInitialValues()
[SetInitialValues1]
SetInitialValues()
getReadyMsg()
[ReadyMsg1]
ReadyMsg()
getStopMsg()
[StopMsg1]
StopMsg()
getReturnCash()
[ReturnCash1]
ReturnCash()
getStorePrices()
[StorePrices1]
StorePrices()
getInitializeData()
[InitializeData1]
InitializeData()
getPrintReceipt()
[PrintReceipt1]
PrintReceipt()
getEnterPinMsg()
[EnterPinMsg1]
EnterPinMsg()

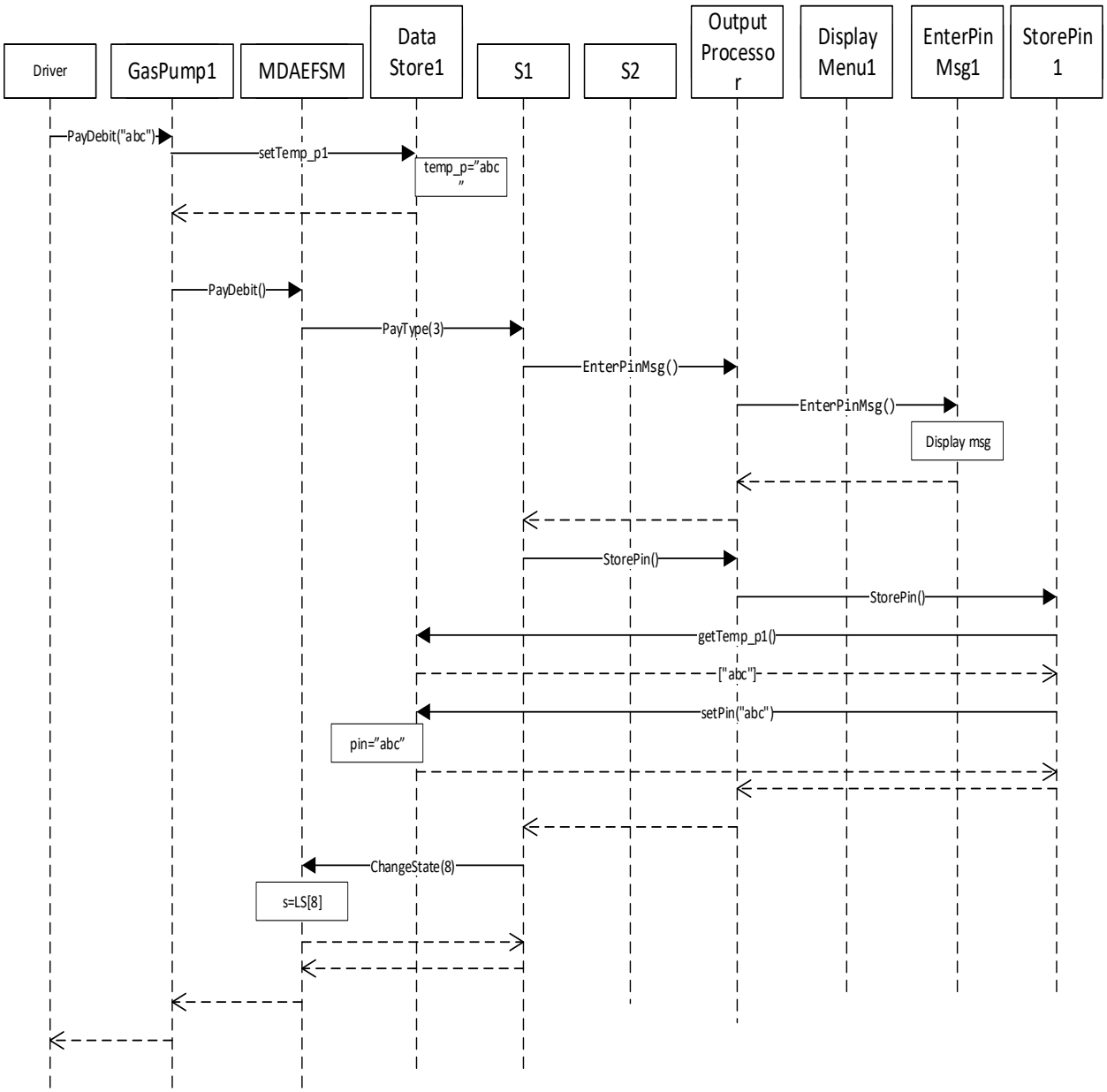
Operation Activate(4.2,7.2)



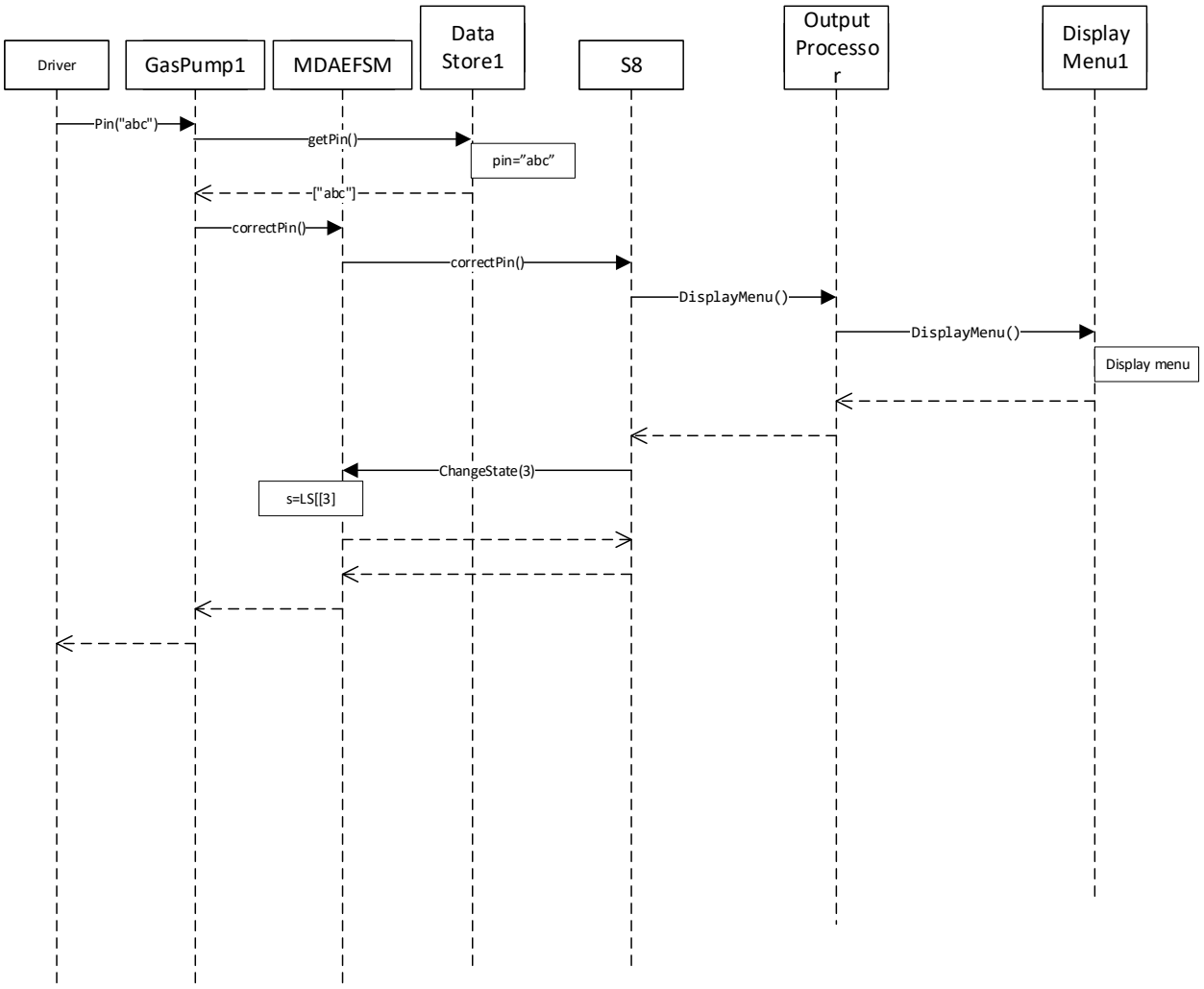
Operation Start():



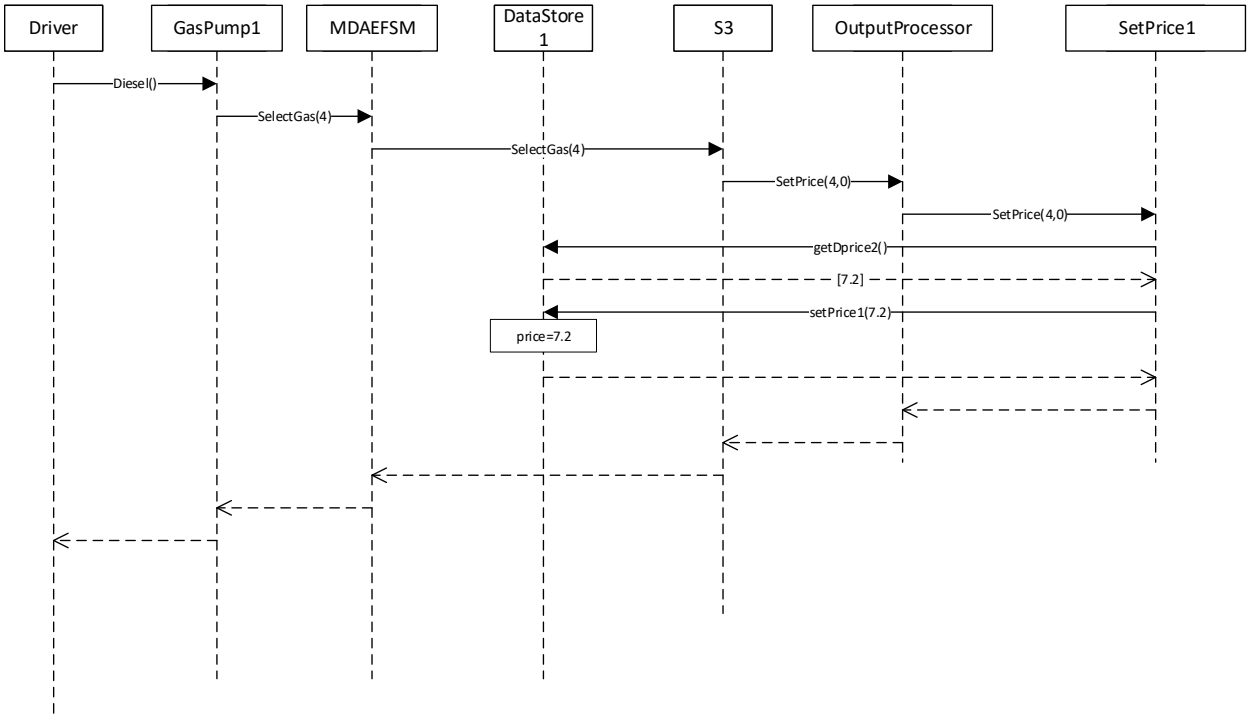
Operation PayDebit("abc"):



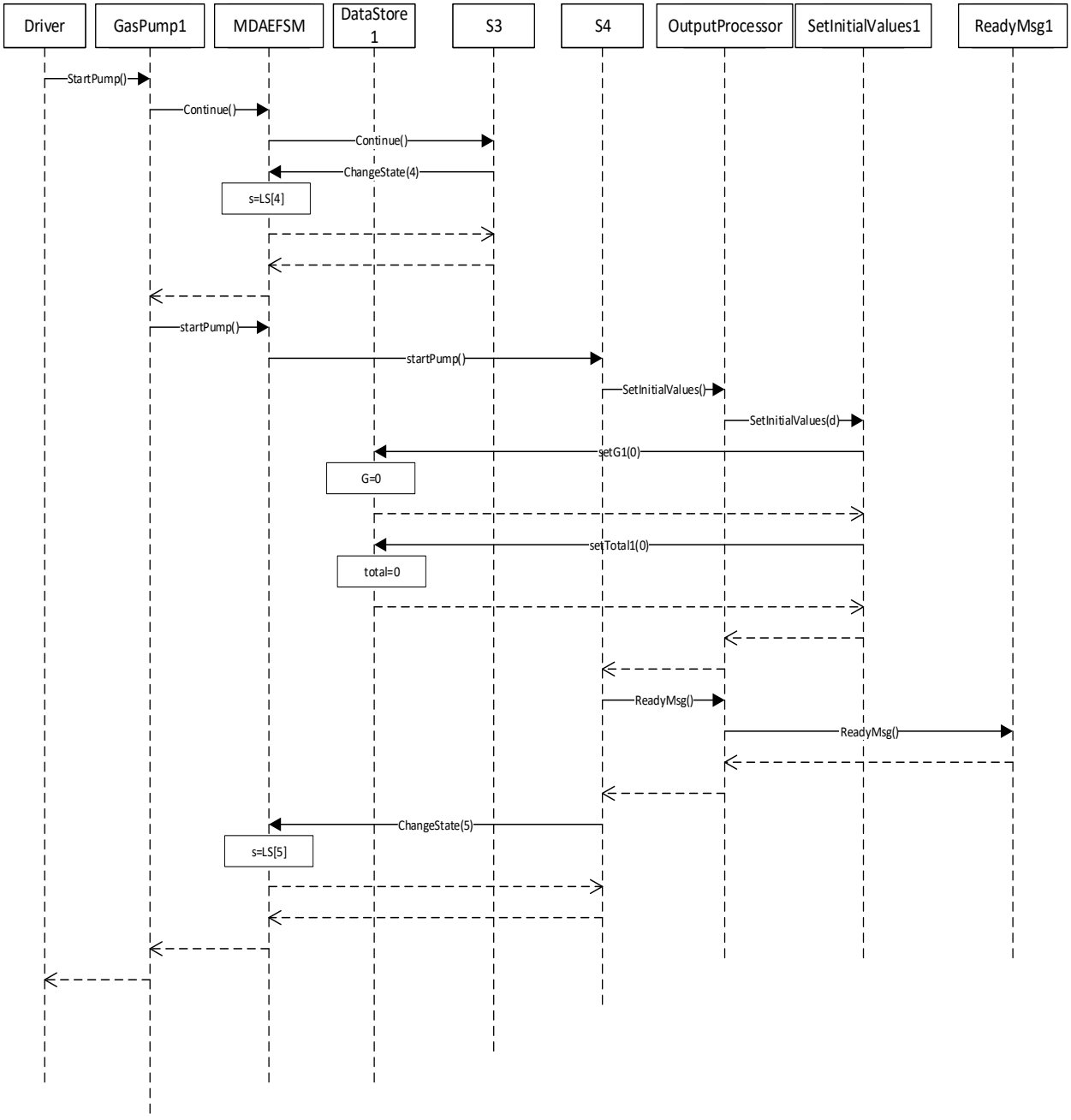
Operation Pin("abc"):



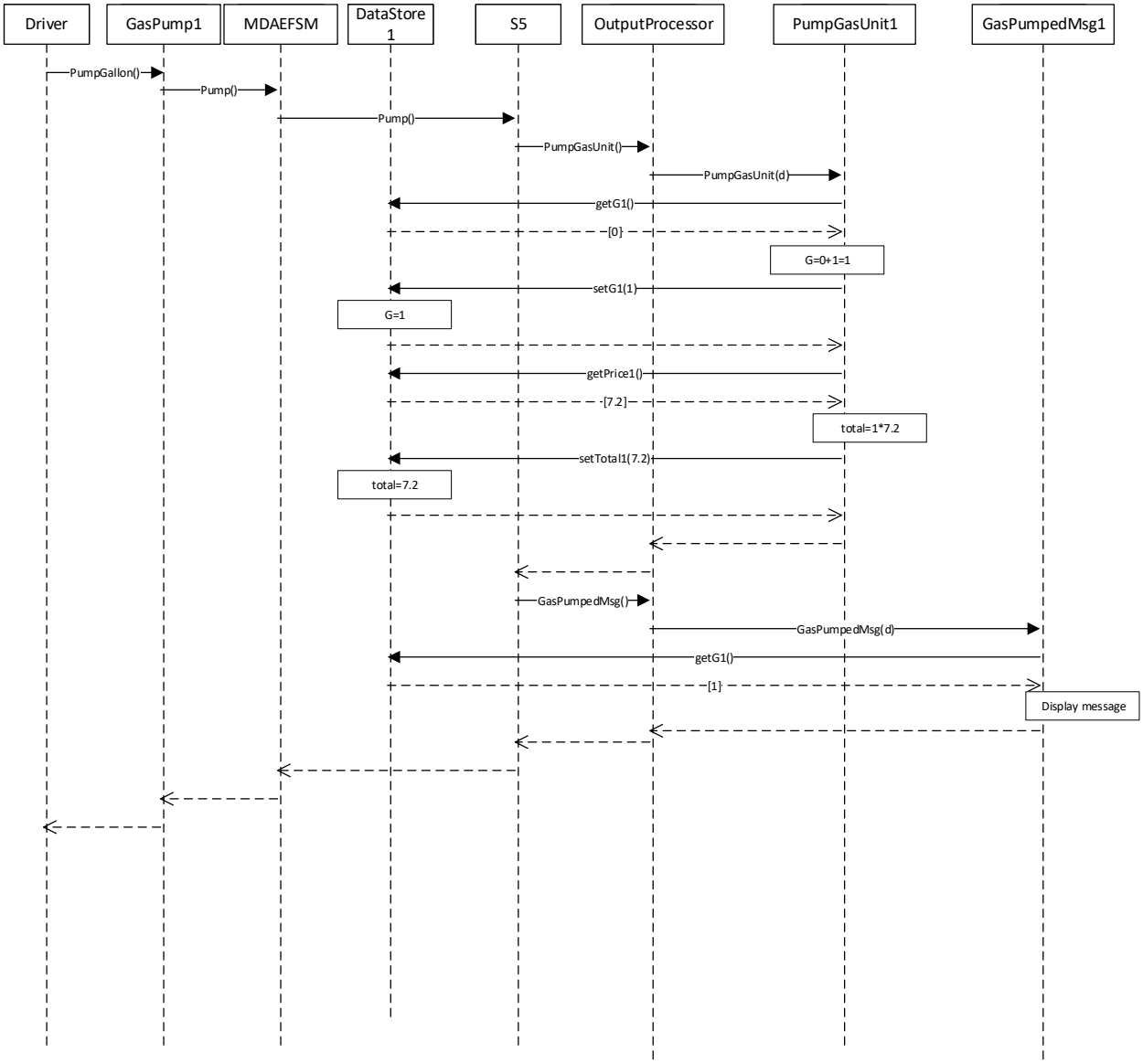
Operation Diesel():



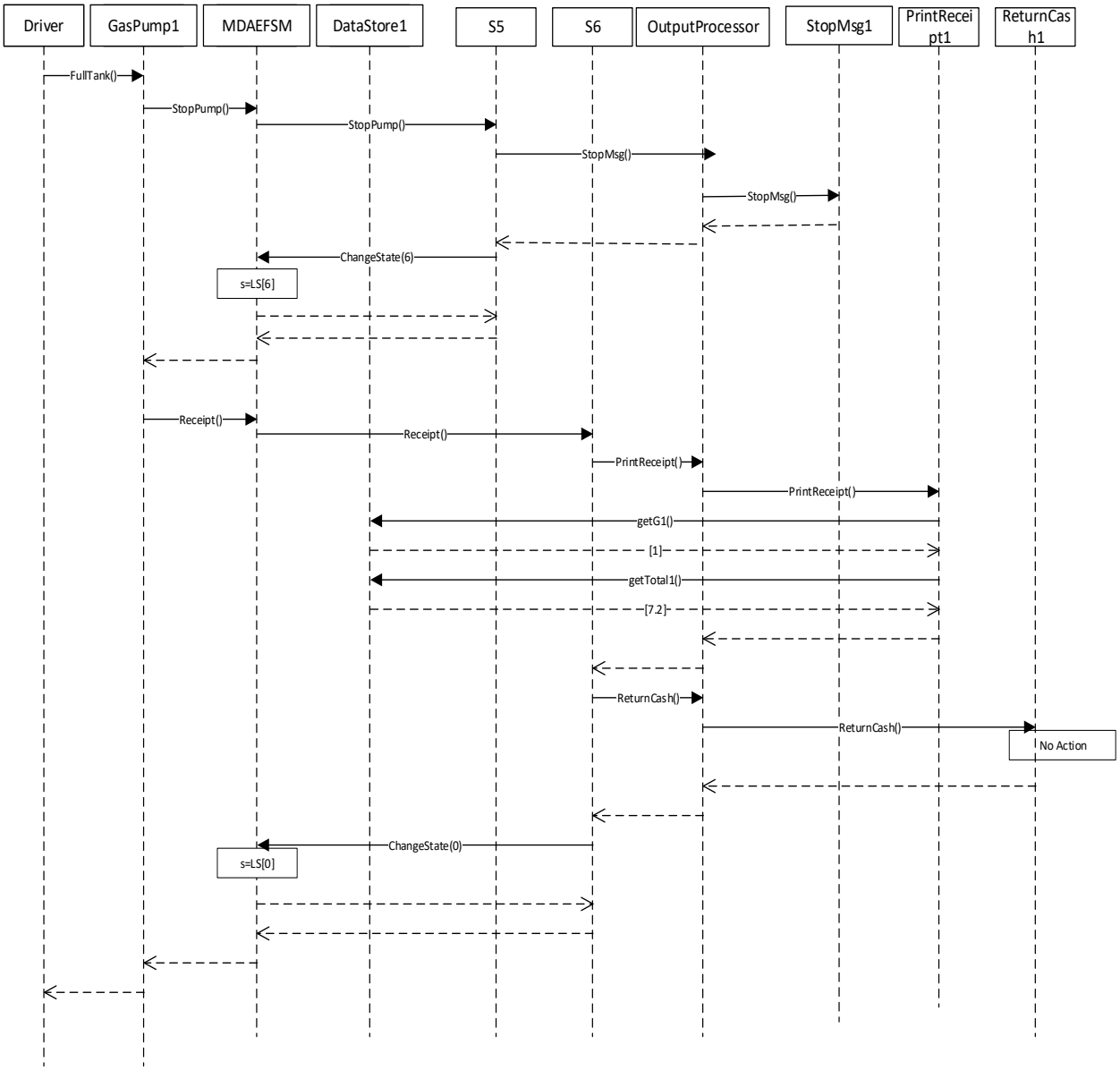
Operation StartPump():



Operation PumpGallon():



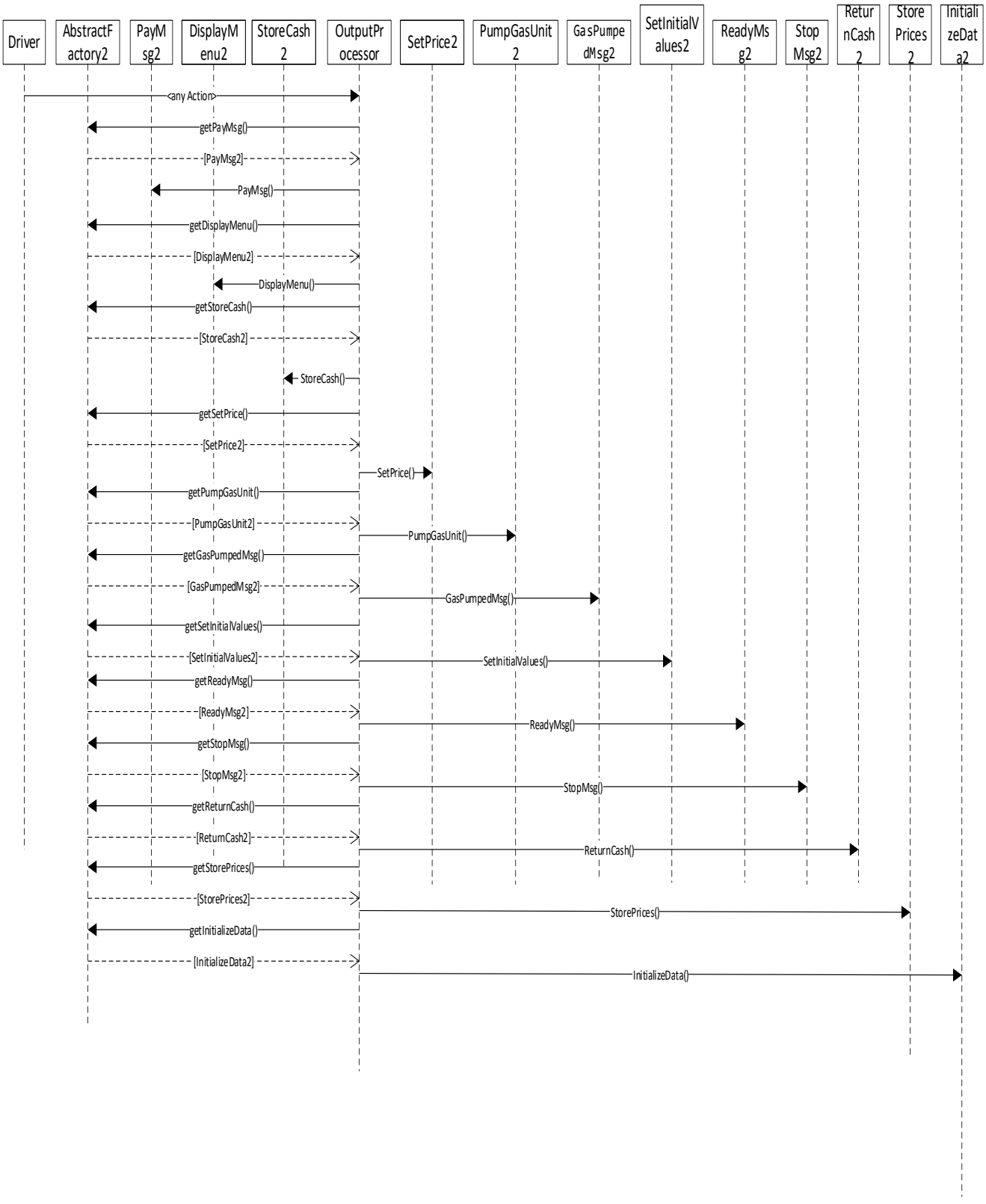
Operation FullTank():



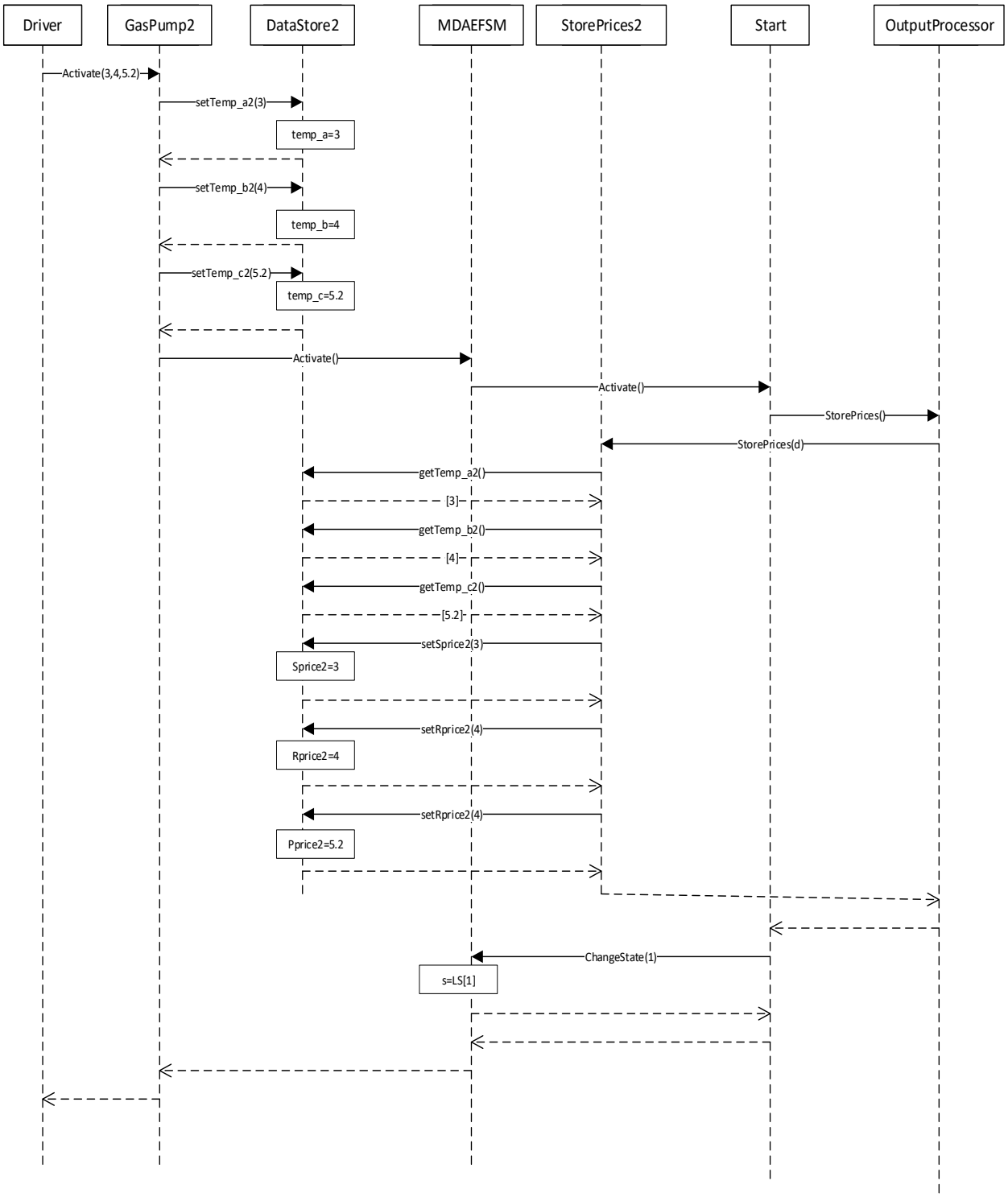
Scenario-II:

Initialization:

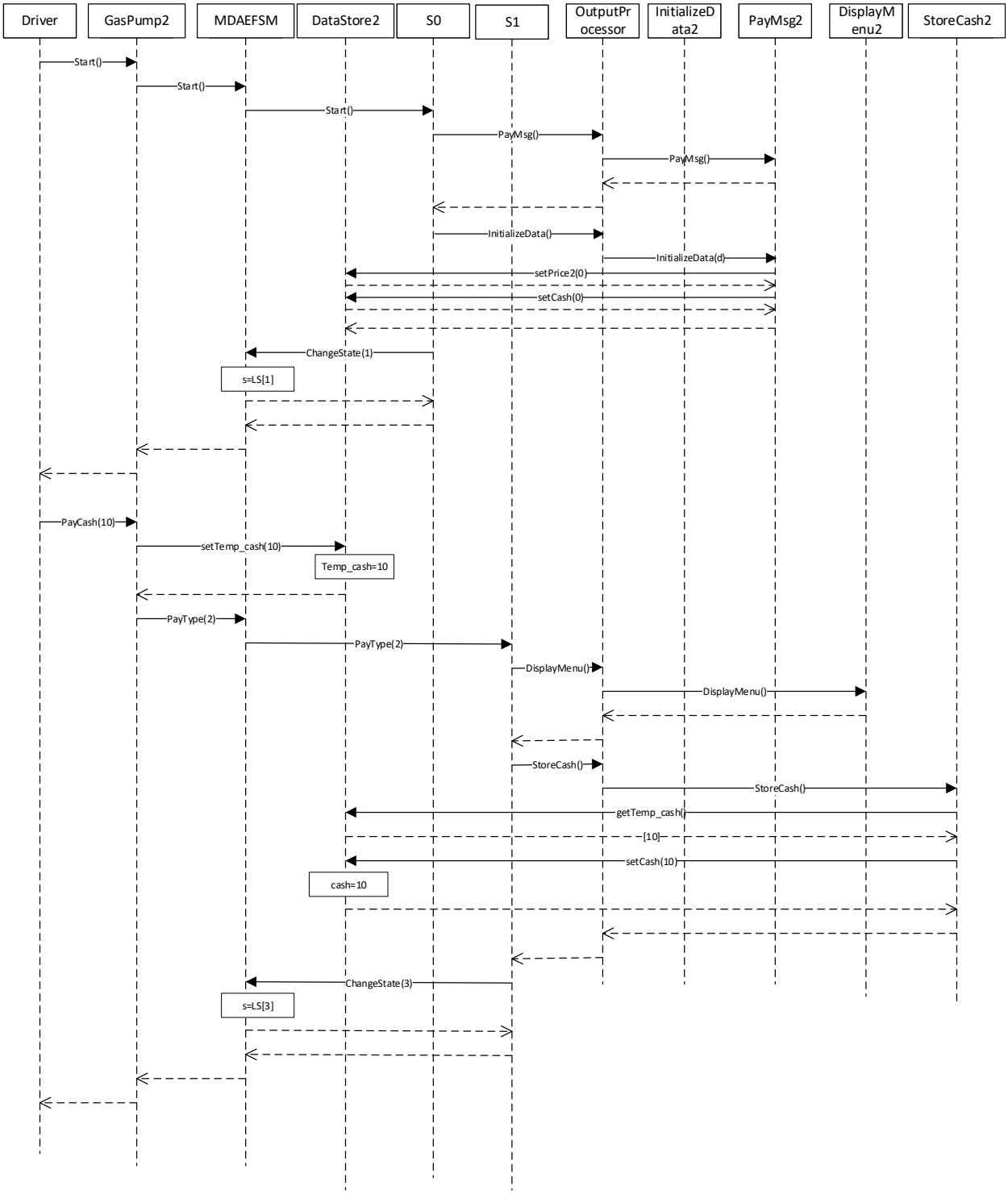
Following sequence diagram shows how Outputprocessor retrieves the objects from AbstractFactory. The diagram shows object creation for classes which are invoked in scenario2. Now, output processor has all the object it can invoke the methods directly. For demonstration only, method calls from outputprocessor are shown. Actual process will be shown in main sequence diagram.


Driver
AbstractFactory2
PayMsg2
DisplayMenu2
StoreCash2
OutputProcessor
SetPrice2
PumpGasUnit2
GasPumpedMsg2
SetInitialValues2
ReadyMsg2
StopMsg2
ReturnCash2
StorePrices2
InitializeData2
<any Action>
getPayMsg()
[PayMsg2]
PayMsg()
getDisplayMenu()
[DisplayMenu2]
DisplayMenu()
getStoreCash()
[StoreCash2]
StoreCash()
getSetPrice()
[SetPrice2]
SetPrice()
getPumpGasUnit()
[PumpGasUnit2]
PumpGasUnit()
getGasPumpedMsg()
[GasPumpedMsg2]
GasPumpedMsg()
getSetInitialValues()
[SetInitialValues2]
SetInitialValues()
getReadyMsg()
[ReadyMsg2]
ReadyMsg()
getStopMsg()
[StopMsg2]
StopMsg()
getReturnCash()
[ReturnCash2]
ReturnCash()
getStorePrices()
[StorePrices2]
StorePrices()
getInitializeData()
[InitializeData2]
InitializeData()

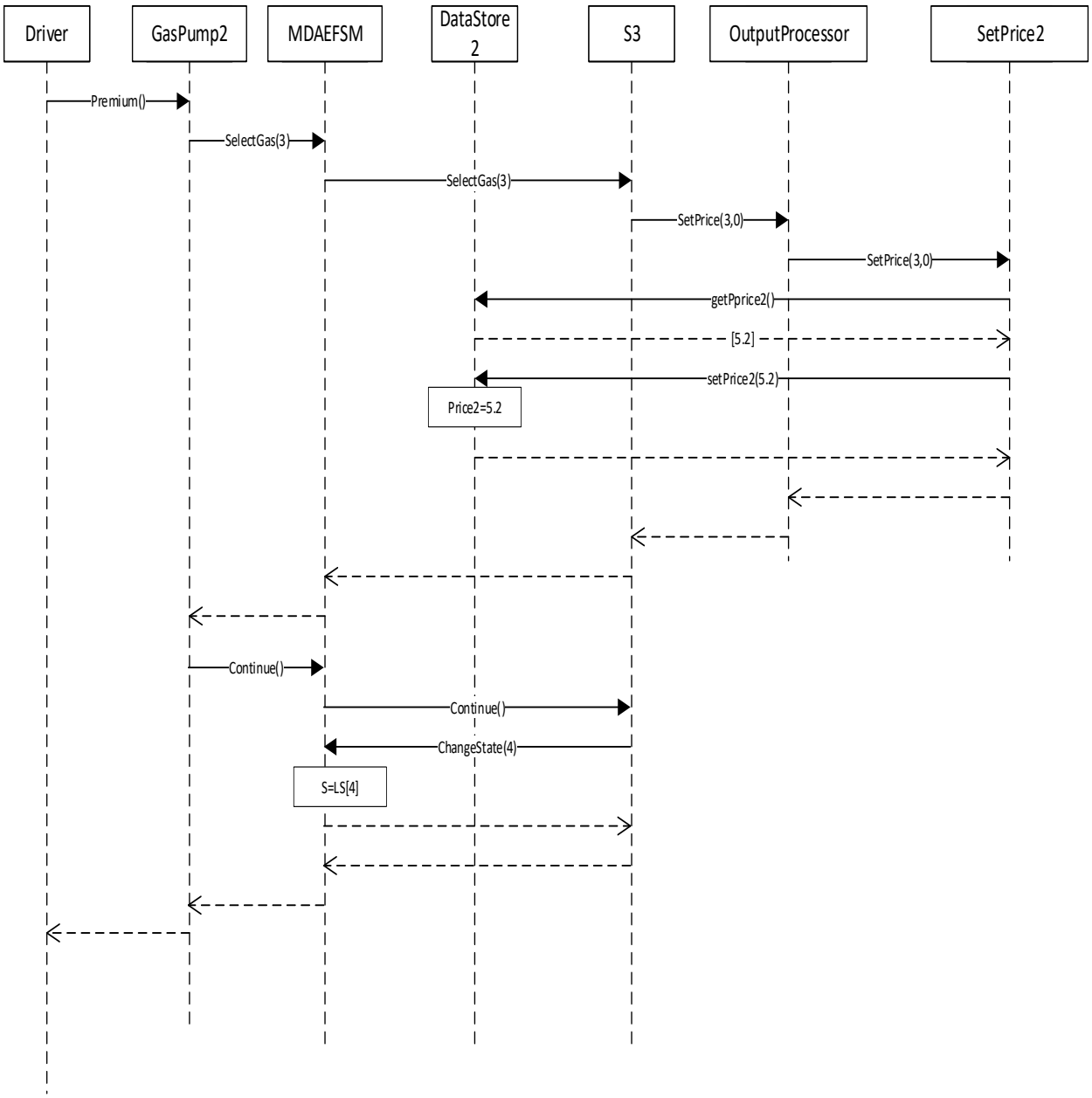
Operation Activate (3,4,5.2)



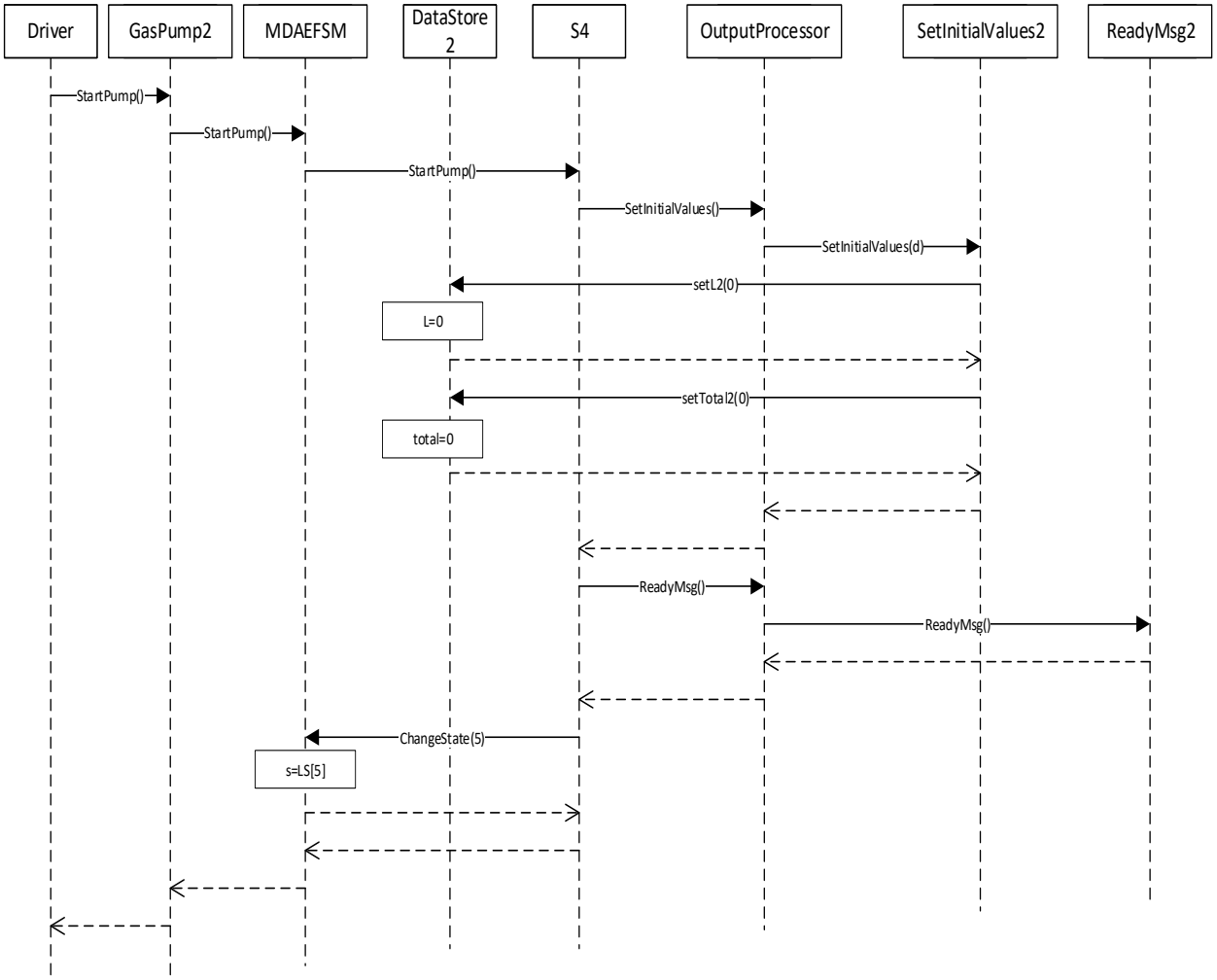
Operation Start() and PayCash(10):



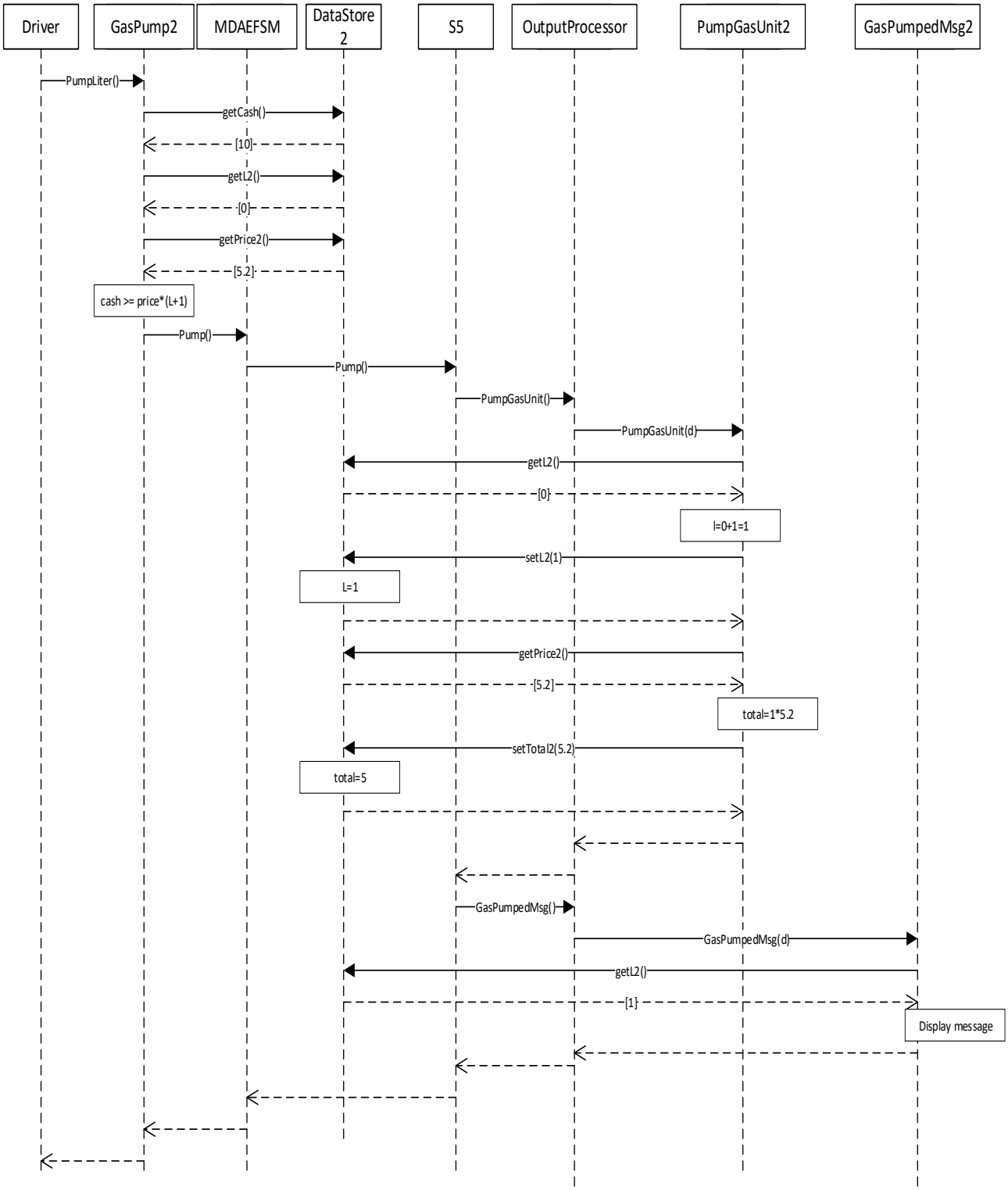
Operation Premium():



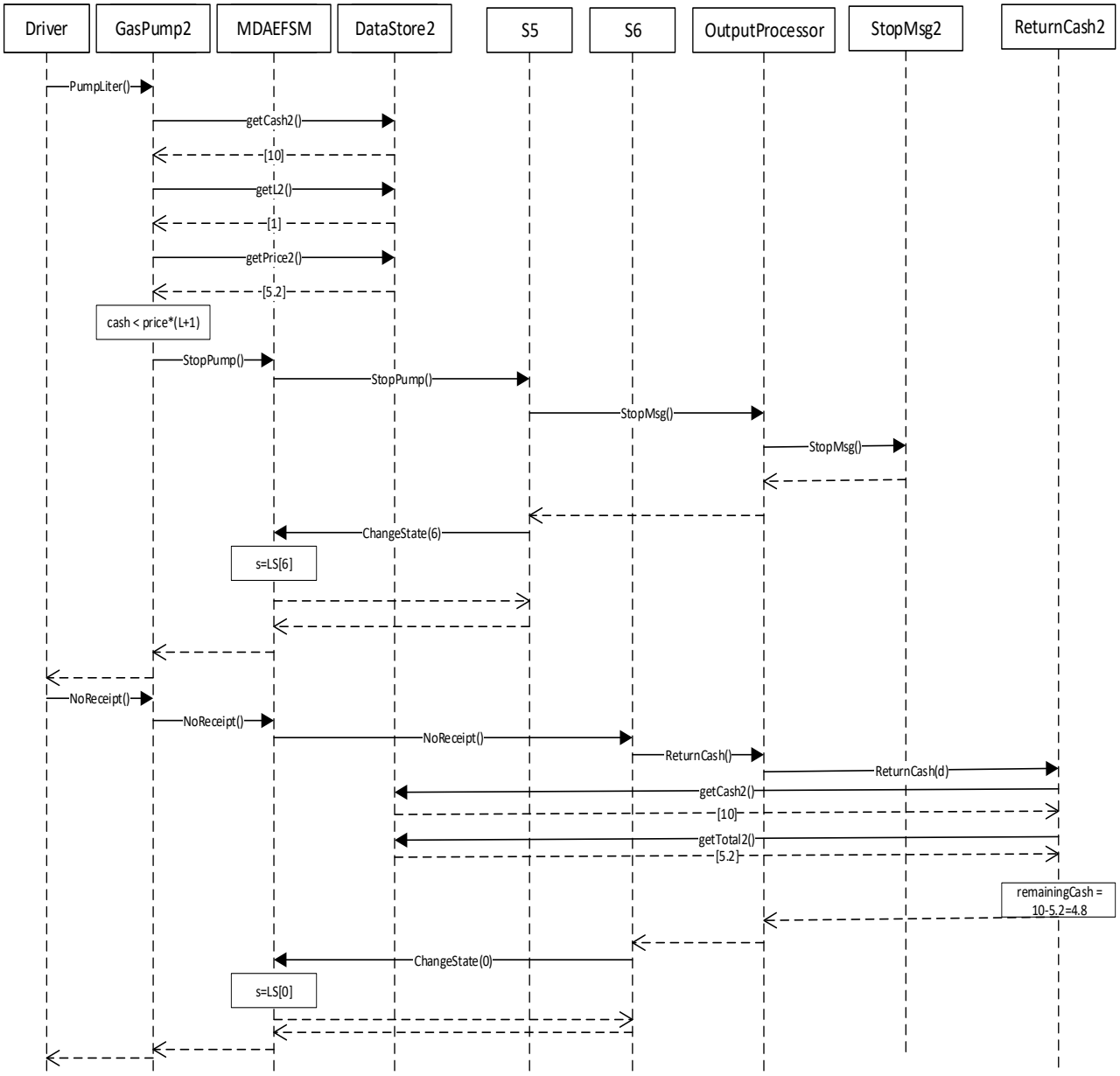
Operation StartPump():



Operation PumpLiter():



Operation PumpLiter() and NoReceipt():



Well documented (commented) source code:

The implementation of GasPump system is done in Java. There are total 6 packages created in entire implementation. Each package contains several Java files which are linked to each other. Out of 6 packages, three belongs to the 3 design patterns which were asked to implement. The description of packages is provided below:

MDA.EFSM: The classes created inside this package belongs to the **State Pattern**.

abstractFactoryPackage: The classes created inside this package belongs to the **AbstractFactory design Pattern**.

strategy and OutputProcessor: The classes created inside this package belongs to the **Strategy design pattern.**

data: The classes created inside this package belongs to the Data Store.

gasPump: The classes created inside this package belongs to Input Processor for the MDA Architecture.

MainDriver: The class created inside this package is the main function.