Final Project

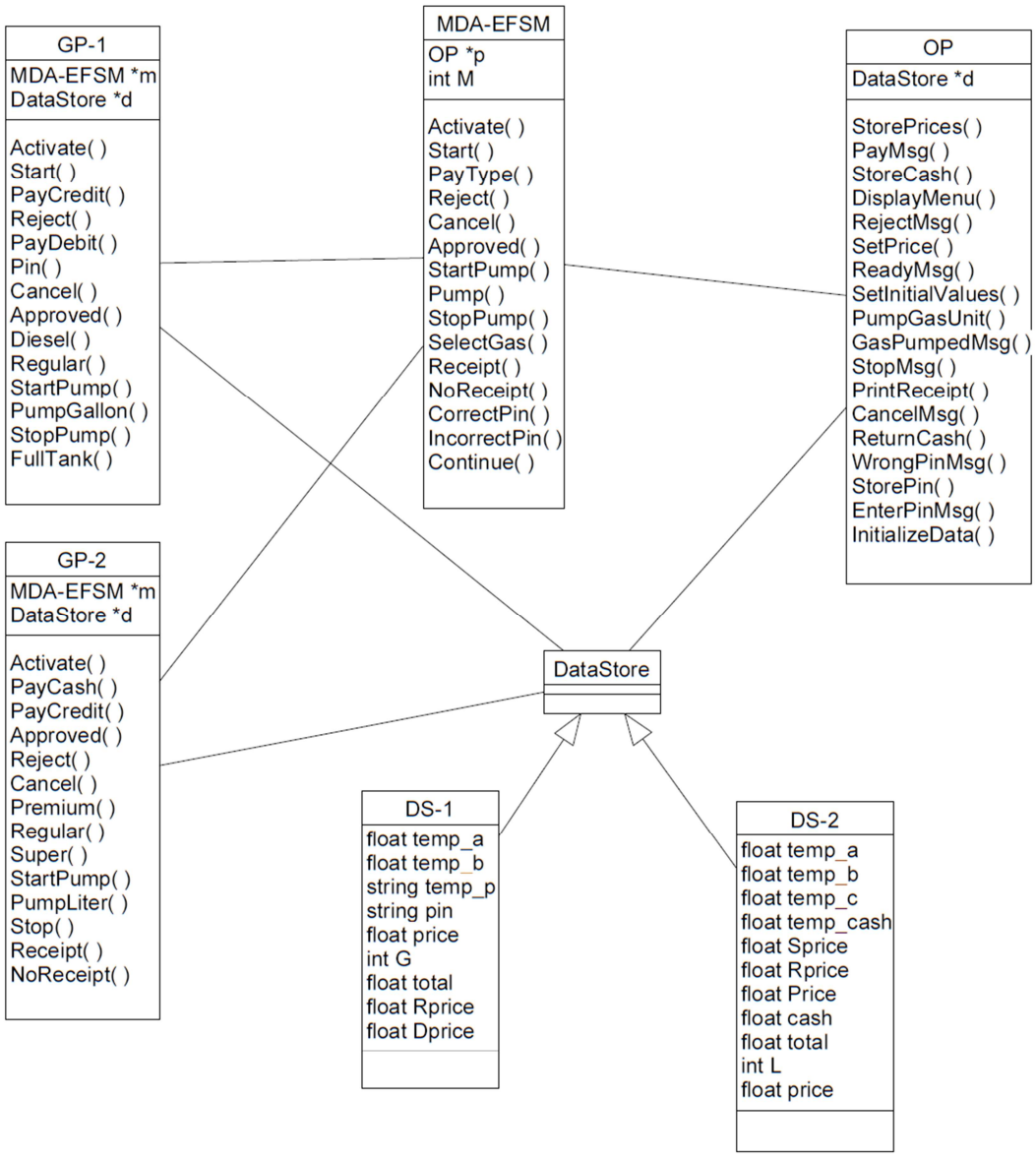
Gas Pump Simulation

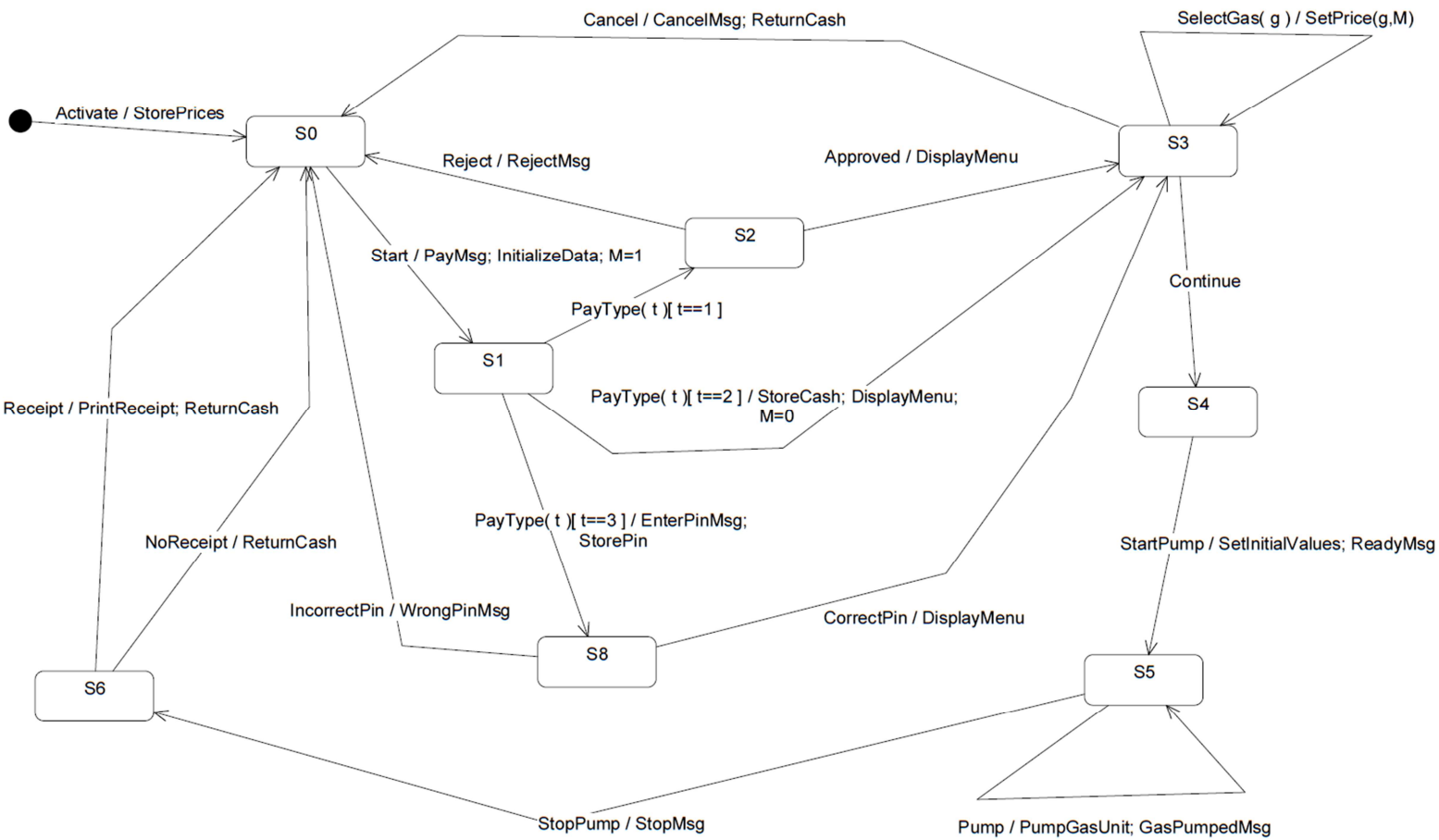
Daniel Moctezuma

Table of Contents

Note: The delivered program is in .jar format. To run the program from the command prompt, go to the directory where the jar is located and type: "java -jar Project.jar".

1. MDA_EFSM Model


GP-1
MDA-EFSM *m
DataStore *d
Activate( )
Start( )
PayCredit( )
Reject( )
PayDebit( )
Pin( )
Cancel( )
Approved( )
Diesel( )
Regular( )
StartPump( )
PumpGallon( )
StopPump( )
FullTank( )
MDA-EFSM
OP *p
int M
Activate( )
Start( )
PayType( )
Reject( )
Cancel( )
Approved( )
StartPump( )
Pump( )
StopPump( )
SelectGas( )
Receipt( )
NoReceipt( )
CorrectPin( )
IncorrectPin( )
Continue( )
OP
DataStore *d
StorePrices( )
PayMsg( )
StoreCash( )
DisplayMenu( )
RejectMsg( )
SetPrice( )
ReadyMsg( )
SetInitialValues( )
PumpGasUnit( )
GasPumpedMsg( )
StopMsg( )
PrintReceipt( )
CancelMsg( )
ReturnCash( )
WrongPinMsg( )
StorePin( )
EnterPinMsg( )
InitializeData( )
GP-2
MDA-EFSM *m
DataStore *d
Activate( )
PayCash( )
PayCredit( )
Approved( )
Reject( )
Cancel( )
Premium( )
Regular( )
Super( )
StartPump( )
PumpLiter( )
Stop( )
Receipt( )
NoReceipt( )
DataStore
DS-1
float temp_a
float temp_b
string temp_p
string pin
float price
int G
float total
float Rprice
float Dprice
DS-2
float temp_a
float temp_b
float temp_c
float temp_cash
float Sprice
float Rprice
float Price
float cash
float total
int L
float price



MDA-EFSM for Gas Pumps

MDA-EFSM Events:

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()

MDA-EFSM 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, 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

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

Operations of the Input Processor (GasPump-2)

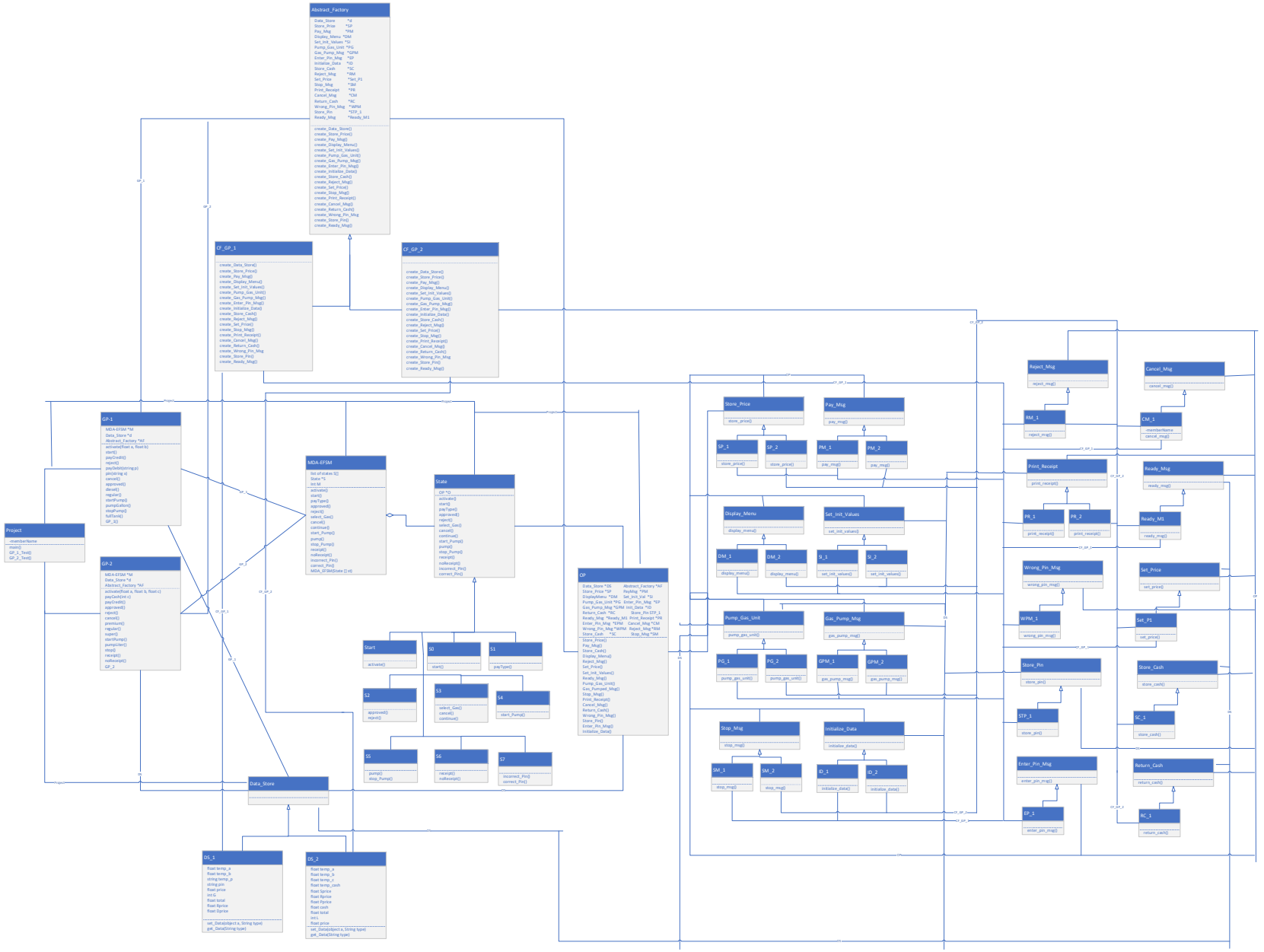
```
Activate(int a, int b, int c) {
        if ((a>0)&&(b>0)&&(c>0)) {
            d->temp_a=a;
            d->temp_b=b;
            d->temp_c=c
            m->Activate()
        }
}

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();
}
```

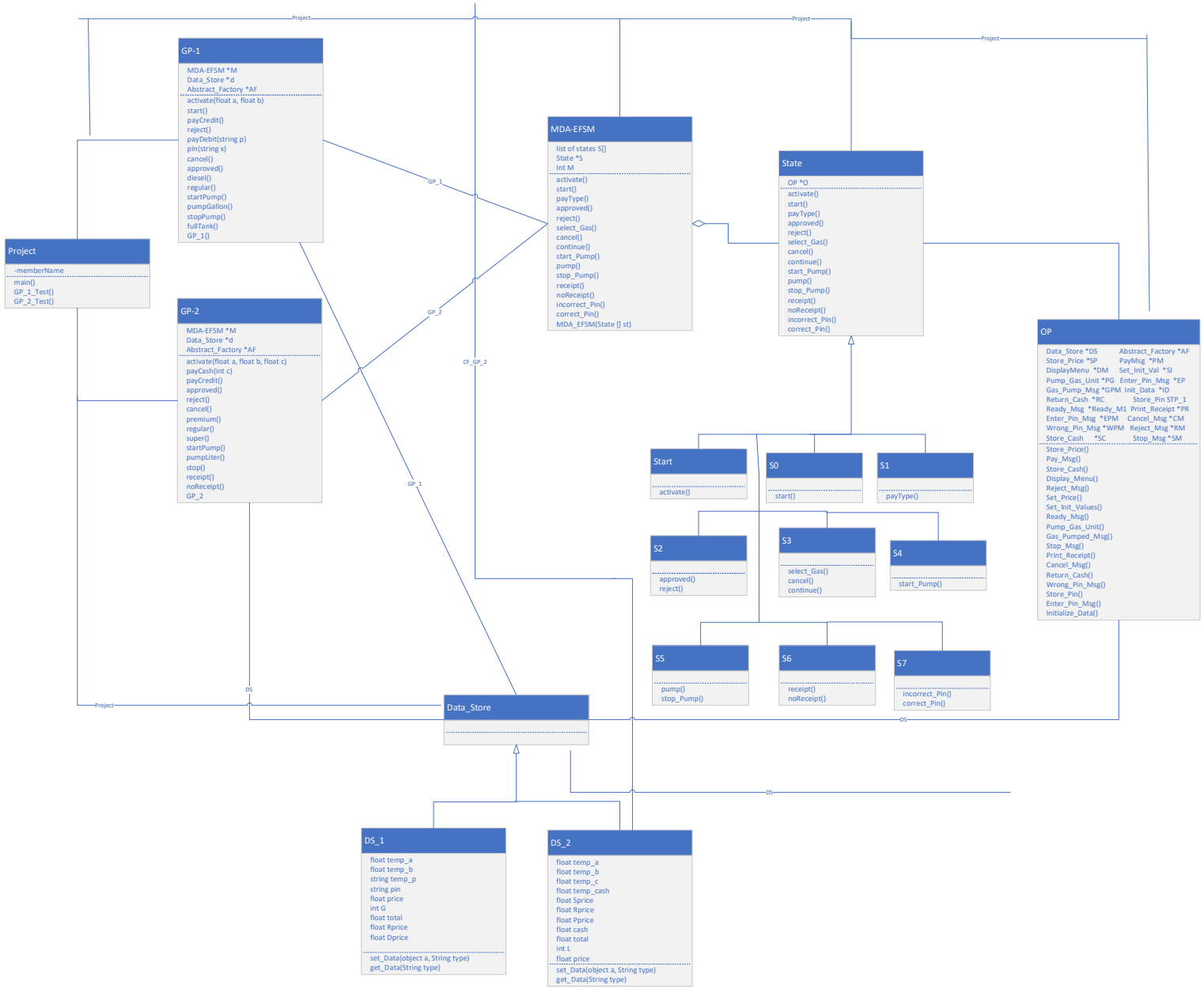
Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already pumped

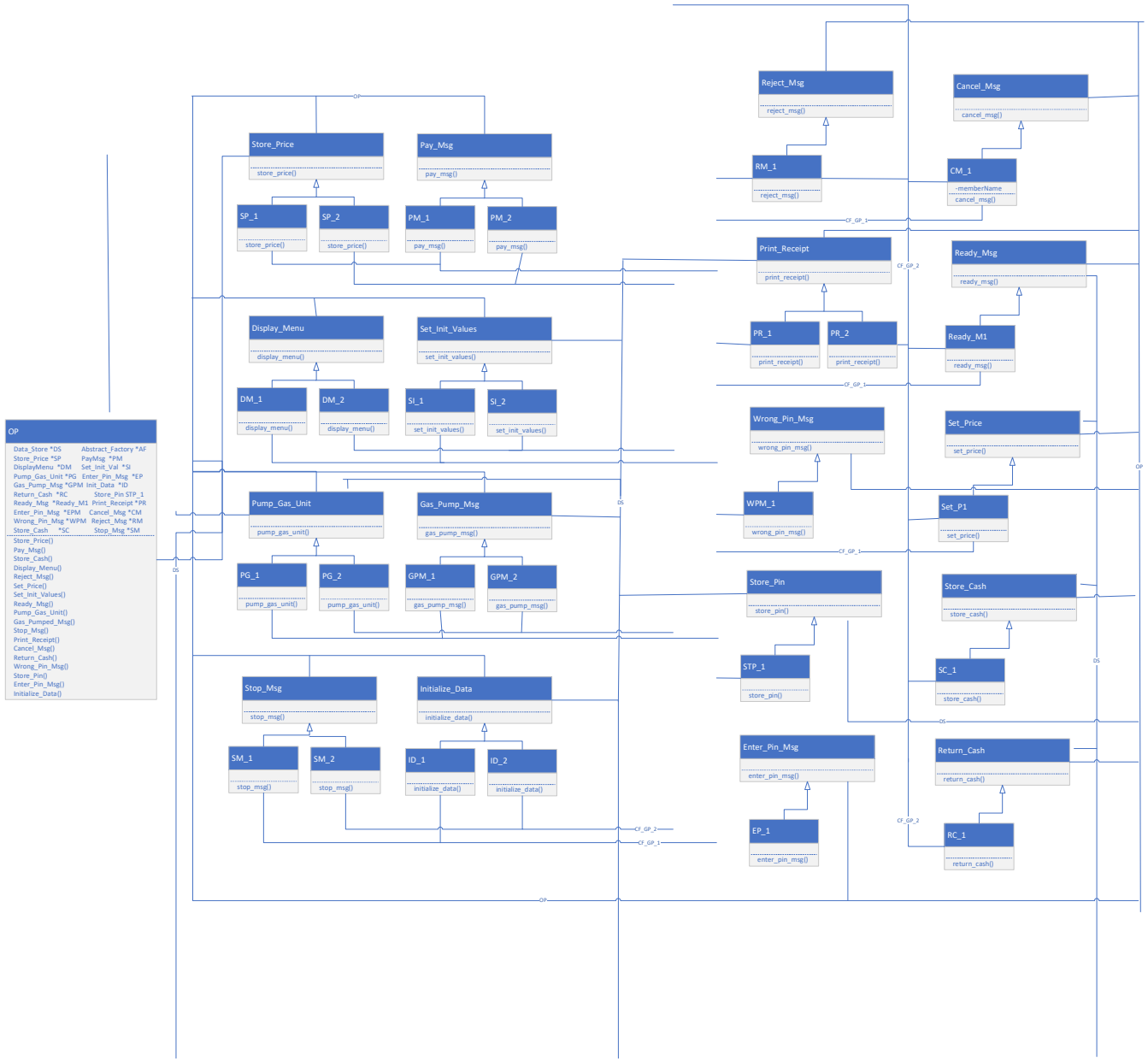
cash , *L*, *price* are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

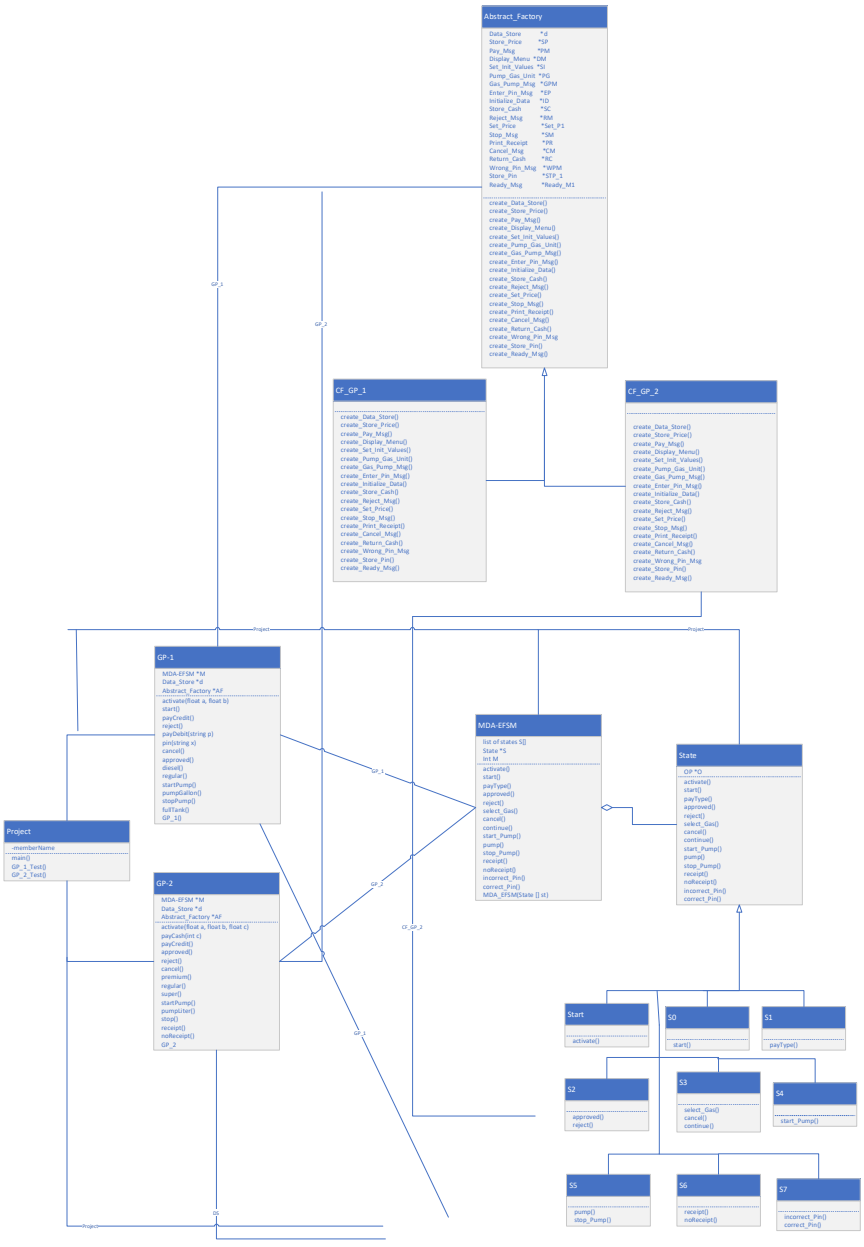
2. Class Diagrams

1st diagram: reference
2nd diagram: Driver, IP, MDA_EFSM, DS, States and OP
3Rd diagram: OP and Strategy Classes
4th diagram: Abstract Factory, Concrete Factory, Driver, IP, MDA_EFSM and States


Abstract_Factory
CF_GP_1
CF_GP_2
GP_1
GP_2
MDA-EFSM
State
OP
Project
Data_Store
DS_1
DS_2
Start
S0
S1
S2
S3
S4
S5
S6
S7
Store_Price
Pay_Msg
SP_1
SP_2
PM_1
PM_2
Display_Menu
Set_Init_Values
DM_1
DM_2
SI_1
SI_2
Pump_Gas_Unit
Gas_Pump_Msg
PG_1
PG_2
GPM_1
GPM_2
Stop_Msg
Initialize_Data
SM_1
SM_2
ID_1
ID_2
Reject_Msg
Cancel_Msg
RM_1
CM_1
Print_Receipt
Ready_Msg
PR_1
PR_2
Ready_M1
Wrong_Pin_Msg
Set_Price
WPM_1
Set_P1
Store_Pin
Store_Cash
STP_1
SC_1
Enter_Pin_Msg
Return_Cash
EP_1
RC_1


Project
-memberName
main()
GP_1_Test()
GP_2_Test()
GP-1
MDA-EFSM *M
Data_Store *d
Abstract_Factory *AF
activate(float a, float b)
start()
payCredit()
reject()
payDebit(string p)
pin(string x)
cancel()
approved()
diesel()
regular()
startPump()
pumpGallon()
stopPump()
fullTank()
GP_1()
GP-2
MDA-EFSM *M
Data_Store *d
Abstract_Factory *AF
activate(float a, float b, float c)
payCash(int c)
payCredit()
approved()
reject()
cancel()
premium()
regular()
super()
startPump()
pumpLiter()
stop()
receipt()
noReceipt()
GP_2
MDA-EFSM
list of states S[]
State *S
Int M
activate()
start()
payType()
approved()
reject()
select_Gas()
cancel()
continue()
start_Pump()
pump()
stop_Pump()
receipt()
noReceipt()
incorrect_Pin()
correct_Pin()
MDA_EFSM(State [] st)
State
OP *O
activate()
start()
payType()
approved()
reject()
select_Gas()
cancel()
continue()
start_Pump()
pump()
stop_Pump()
receipt()
noReceipt()
incorrect_Pin()
correct_Pin()
OP
Data_Store *DS
Store_Price *SP
DisplayMenu *DM
Pump_Gas_Unit *PG
Gas_Pump_Msg *GPM
Return_Cash *RC
Ready_Msg *Ready_M1
Enter_Pin_Msg *EPM
Wrong_Pin_Msg *WPM
Store_Cash *SC
Abstract_Factory *AF
PayMsg *PM
Set_Init_Val *SI
Enter_Pin_Msg *EP
Init_Data *ID
Store_Pin STP_1
Print_Receipt *PR
Cancel_Msg *CM
Reject_Msg *RM
Stop_Msg *SM
Store_Price()
Pay_Msg()
Store_Cash()
Display_Menu()
Reject_Msg()
Set_Price()
Set_Init_Values()
Ready_Msg()
Pump_Gas_Unit()
Gas_Pumped_Msg()
Stop_Msg()
Print_Receipt()
Cancel_Msg()
Return_Cash()
Wrong_Pin_Msg()
Store_Pin()
Enter_Pin_Msg()
Initialize_Data()
Start
activate()
S0
start()
S1
payType()
S2
approved()
reject()
S3
select_Gas()
cancel()
continue()
S4
start_Pump()
S5
pump()
stop_Pump()
S6
receipt()
noReceipt()
S7
incorrect_Pin()
correct_Pin()
Data_Store
DS_1
float temp_a
float temp_b
string temp_p
string pin
float price
int G
float total
float Rprice
float Dprice
set_Data(object a, String type)
get_Data(String type)
DS_2
float temp_a
float temp_b
float temp_c
float temp_cash
float Sprice
float Rprice
float Pprice
float cash
float total
int L
float price
set_Data(object a, String type)
get_Data(String type)
Project
GP_1
GP_2
CF_GP_2
DS


OP
Data_Store *DS
Store_Price *SP
DisplayMenu *DM
Pump_Gas_Unit *PG
Gas_Pump_Msg *GPM
Return_Cash *RC
Ready_Msg *Ready_M1
Enter_Pin_Msg *EPM
Wrong_Pin_Msg *WPM
Store_Cash *SC
Abstract_Factory *AF
PayMsg *PM
Set_Init_Val *SI
Enter_Pin_Msg *EP
Init_Data *ID
Store_Pin STP_1
Print_Receipt *PR
Cancel_Msg *CM
Reject_Msg *RM
Stop_Msg *SM
Store_Price()
Pay_Msg()
Store_Cash()
Display_Menu()
Reject_Msg()
Set_Price()
Set_Init_Values()
Ready_Msg()
Pump_Gas_Unit()
Gas_Pumped_Msg()
Stop_Msg()
Print_Receipt()
Cancel_Msg()
Return_Cash()
Wrong_Pin_Msg()
Store_Pin()
Enter_Pin_Msg()
Initialize_Data()
Store_Price
store_price()
Pay_Msg
pay_msg()
SP_1
SP_2
PM_1
PM_2
Display_Menu
display_menu()
Set_Init_Values
set_init_values()
DM_1
DM_2
SI_1
SI_2
Pump_Gas_Unit
pump_gas_unit()
Gas_Pump_Msg
gas_pump_msg()
PG_1
PG_2
GPM_1
GPM_2
Stop_Msg
stop_msg()
Initialize_Data
initialize_data()
SM_1
SM_2
ID_1
ID_2
Reject_Msg
reject_msg()
Cancel_Msg
cancel_msg()
RM_1
CM_1
-memberName
Print_Receipt
print_receipt()
Ready_Msg
ready_msg()
PR_1
PR_2
Ready_M1
Wrong_Pin_Msg
wrong_pin_msg()
Set_Price
set_price()
WPM_1
Set_P1
Store_Pin
store_pin()
Store_Cash
store_cash()
STP_1
SC_1
Enter_Pin_Msg
enter_pin_msg()
Return_Cash
return_cash()
EP_1
RC_1
CF_GP_1
CF_GP_2
DS
OP


Abstract_Factory
Data_Store *d
Store_Price *SP
Pay_Msg *PM
Display_Menu *DM
Set_Init_Values *SI
Pump_Gas_Unit *PG
Gas_Pump_Msg *GPM
Enter_Pin_Msg *EP
Initialize_Data *ID
Store_Cash *SC
Reject_Msg *RM
Set_Price *Set_P1
Stop_Msg *SM
Print_Receipt *PR
Cancel_Msg *CM
Return_Cash *RC
Wrong_Pin_Msg *WPM
Store_Pin *STP_1
Ready_Msg *Ready_M1
create_Data_Store()
create_Store_Price()
create_Pay_Msg()
create_Display_Menu()
create_Set_Init_Values()
create_Pump_Gas_Unit()
create_Gas_Pump_Msg()
create_Enter_Pin_Msg()
create_Initialize_Data()
create_Store_Cash()
create_Reject_Msg()
create_Set_Price()
create_Stop_Msg()
create_Print_Receipt()
create_Cancel_Msg()
create_Return_Cash()
create_Wrong_Pin_Msg
create_Store_Pin()
create_Ready_Msg()
CF_GP_1
create_Data_Store()
create_Store_Price()
create_Pay_Msg()
create_Display_Menu()
create_Set_Init_Values()
create_Pump_Gas_Unit()
create_Gas_Pump_Msg()
create_Enter_Pin_Msg()
create_Initialize_Data()
create_Store_Cash()
create_Reject_Msg()
create_Set_Price()
create_Stop_Msg()
create_Print_Receipt()
create_Cancel_Msg()
create_Return_Cash()
create_Wrong_Pin_Msg
create_Store_Pin()
create_Ready_Msg()
CF_GP_2
create_Data_Store()
create_Store_Price()
create_Pay_Msg()
create_Display_Menu()
create_Set_Init_Values()
create_Pump_Gas_Unit()
create_Gas_Pump_Msg()
create_Enter_Pin_Msg()
create_Initialize_Data()
create_Store_Cash()
create_Reject_Msg()
create_Set_Price()
create_Stop_Msg()
create_Print_Receipt()
create_Cancel_Msg()
create_Return_Cash()
create_Wrong_Pin_Msg
create_Store_Pin()
create_Ready_Msg()
GP_1
GP_2
Project
Project
GP-1
MDA-EFSM *M
Data_Store *d
Abstract_Factory *AF
activate(float a, float b)
start()
payCredit()
reject()
payDebit(string p)
pin(string x)
cancel()
approved()
diesel()
regular()
startPump()
pumpGallon()
stopPump()
fullTank()
GP_1()
MDA-EFSM
list of states S[]
State *S
int N
activate()
start()
payType()
approved()
reject()
select_Gas()
cancel()
continue()
start_Pump()
pump()
stop_Pump()
receipt()
noReceipt()
incorrect_Pin()
correct_Pin()
MDA_EFSM(State [] st)
State
OP *O
activate()
start()
payType()
approved()
reject()
select_Gas()
cancel()
continue()
start_Pump()
pump()
stop_Pump()
receipt()
noReceipt()
incorrect_Pin()
correct_Pin()
Project
-memberName
main()
GP_1_Test()
GP_2_Test()
GP-2
MDA-EFSM *M
Data_Store *d
Abstract_Factory *AF
activate(float a, float b, float c)
payCash(int c)
payCredit()
approved()
reject()
cancel()
premium()
regular()
super()
startPump()
pumpDiet()
stop()
receipt()
noReceipt()
GP_2
GP_1
GP_2
CF_GP_2
GP_1
Project
D1
Start
activate()
S0
start()
S1
payType()
S2
approved()
reject()
S3
select_Gas()
cancel()
continue()
S4
start_Pump()
S5
pump()
stop_Pump()
S6
receipt()
noReceipt()
S7
incorrect_Pin()
correct_Pin()

3. Class Descriptions

1. Project Class (found in project Package, contains main driver)

The project class tests contains the main driver of the build. It is used to test the Gas Pumps.

A. Operations

1. Main

This is the driver that tests the two input processors. GP_1 and GP_2. The user is given the option between choosing one of the two input processors and testing them. Based on user's choice the driver will either run GP_1_Test or GP_2_Test.

2. GP_1_Test

This operation prompts the user to test the various functionalities of the Input Processor GP_1. The program runs until the user terminates the program. This operation creates the State [] list, MDA_EFSM, CF_GP_1 (concrete factory), DS_1 (data store), IP and OP needed to simulate the Gas Pump 1 functionalities.

3. GP_2_Test

This operation prompts the user to test the various functionalities of the Input Processor GP_2. The program runs until the user terminates the program. This operation creates the State[]list, MDA_EFSM, CF_GP_2 (concrete factory), DS_2 (data store), IP and OP needed to simulate the Gas Pump 2 functionalities.

2. Input Processor Classes (Package)

A. GP_1

The input processor for Gas Pump 1 design.

1. Attributes:

    a. Data_Store *d

    b. Absract_Factory *gp1

    c. MDA_EFSM *m

2. Operations:

    a. public void GP_1(MDA_EFSM md, Abstract Factory AF, Data_Store ds)

    This is the constructor for GP_1. Sets the pointers of the attributes.

    b. void activate (float a, float b)

    Calls activate on the MDA_EFSM and stores temp_a and temp_b in the data store.

    c. void start()

    Calls start on the MDA_EFSM when user chooses start.

    d. void payCredit()

    Calls payType(1) on the MDA_EFSM if payCredit is chosen.

    e. void reject()

    Calls reject on the MDA_EFSM if the card is rejected.

    f. void payDebit(String p)

    Calls payDebit on the MDA_EFSM and stores a pin in the datastore.

g. void pin(String x)

Checks if the entered pin is equal to the stored pin. Calls incorrect pin on MDA_EFSM if incorrect and correctPin if correct.

h. void cancel()

Calls cancel on the MDA_EFSM if the user cancels the transaction.

i. void approved()

Calls approved on the MDA_EFSM if the card is approved.

j. void diesel()

Calls selectGas(4) on the MDA_EFSM if user selects diesel option.

k. void regular()

Calls selectGas(1) on the MDA_EFSM if user selects regular option.

l. void startPump()

Calls continue() and startpump() on the MDA_EFSM if this option is selected and price has been set.

m. void pumpGallon()

Calls pump on the MDA_EFSM and pumps a gallon of gas.

n. void stopPump()

Calls stopPump and receipt on the MDA_EFSM.

o. void fullTank()

Calls stopPump and receipt on the MDA_EFSM if the "tank is full."

B. GP_2

The input processor for Gas Pump 2.

1. Attributes:

a. Data_Store *d

b. Abstract_Factory *gp2

c. MDA_EFSM *m

2. Classes:

a. public void GP_2(MDA_EFSM md, Abstract Factory AF, Data_Store ds)

This is the constructor for GP_2. It sets the pointers for the attributes.

b. void activate(float a, float b, float c)

Calls activate on the MDA_EFSM and temporarily stores the three float values in the Data_Store.

c. void payCash(int c)

Temporarily stores the cash value, calls start on the MDA_EFSM and selects payType(2) on the MDA_EFSM.

d. void payCredit()

Calls start and payType(1) on the MDA_EFSM

e. void approved()

Calls approved on the MDA_EFSM if the card is approved.

f. void cancel()

Calls cancel on the MDA_EFSM if the user cancels their transaction.

g. void reject()

Calls reject on the MDA_EFSM if the card is rejected.

h. void premium()

Calls selectGas(3) and continue on the MDA_EFSM if the user selects premium.

i. void regular()

Calls selectGas(1) and continue on the MDA_EFSM if the user selects regular.

j. void super()

Calls selectGas(2) and continue on the MDA_EFSM if the user selects super.

k. void startPump()

Calls startPump on the MDA_EFSM

l. void pumpLiter()

Calls pump on the MDA_EFSM. Pumps a liter of gas.

m. void stop()

Calls stopPump on the MDA_EFSM to stop pumping.

n. void receipt()

Calls receipt on the MDA_EFSM

o. void noReceipt()

Calls noReceipt on the MDA_EFSM.

3. MDA-EFSM and State Classes (MDA_EFSM Package)

A. MDA-EFSM

The MDA_EFSM class contains the platform independent implementation of the Gas Pump designs. It is also the context class for the state pattern. I have chosen a centralized state pattern, so the MDA_EFSM changes the states in addition to calling operations on the state classes.

1. Attributes:

a. List of states s []

b. State *cs //used to track current state

c. int M // 0 represents cash, 1 is debit/credit

2. Operations

a. public MDA_EFSM(State [] st)

Constructor for MDA_EFSM. Passes a list of states and sets it to the appropriate attribute. Sets current state to s[0] and M to 1.

b. public void activate()

If the current state is start (S[0] in state list), calls activate on S[0] and changes cs to to S[1] (S0 on state chart)

c. public void start()

If the current state is s[1], calls start on s[1] and changes cs to s[2] (s1 on state chart).

d. public void payType(int t)

If current state is s[2], calls payType(t) on current state and sets cs to: s[3] (s2 in diagram) if t==1
s[4] (s3 in diagram) and sets m to 0 if t==2
s[8] (s8 in state diagram) if t==3.

e. public void approved()

If current state is s[3], calls approved on current state and sets cs to s[4] (s3 in state diagram)

f. public void reject()

If current state is s[3], calls reject on current state and sets cs to s[1] (s0 in state diagram)

g. public void select_Gas(int g)

If current state is s[4], calls select gas(g,m) on current state. Does not change states.

h. public void cancel()

If current state is s[4], calls cancel on current state and sets cs to s[1] (s0 in state diagram).

i.public void Continue()

If the current state is s[4], calls continue on the current state and sets cs to s[5] (s4 in state diagram)

j. public void start_Pump()

If current state is s[5], calls start pump on current state and sets cs to s[6] (s5 in diagram).

k. public void pump()

If current state is s[6], calls pump on current state. Does not change state.

l. public void stop_Pump()

If current state is s[6], calls stopPump on current state and sets cs to s[7] (s6 in state diagram)

m. public void receipt()

If current state is s[7], calls receipt on current state and sets cs to s[1] (s0 in state diagram)

n. public void noReceipt()

If current state is s[7], calls noreceipt on current state and sets cs to s[1] (s0 in state diagram).

p. public void incorrect_Pin()

If current state is s[8], call incorrect pin on current state and sets cs to s[1] (s0 in state diagram)

q. public void correct_Pin

If current state is s[8], calls correct pin on current state and sets cs to s[4] (s3 in state diagram).

B. State Class // super class

This is the interface on which the states are built on. State classes call on the Output Processor (OP) to perform certain actions.

1. Attributes:

OP *o // Pointer to output processor OP

MDA_EFSM

2. Operations: // these are overridden

a. public void activate()

b. public void start()

c. public void payType(int a)

d. public void approved()

e. public void reject()

f. public void select_Gas(int g, int m)

g. public void cancel()

h. public void Continue()

i. public void start_Pump()

j. public void pump()

k. public void stop_Pump()

l. public void receipt()

m. public void noReceipt()

n. public void incorrect_Pin()

o. public void correct_Pin

C. Start Class // inherits from State

The Start class is the representation of “Start” in the state diagram. It overrides the activate operation.

a. Start (OP o)

Constructor for start class. Initializes OP pointer to parameter.

b. activate()

Calls store_Price on outpur processor O.

D. S0 Class // inherits from State

The S0 class is the representation of “SO” in the state diagram. It overrides the start operation.

a. S0 (OP o)

Constructor for S1 class. Initializes OP pointer.

b. start()

Calls pay_Msg and initialize_Data on the output processor O.

E. S1 Class //inherits from State

The S1 class is the representation of “S1” in the state diagram. It overrides payType(int).

a. S1 (OP o)

Constructor. Initializes OP pointer.

b. payType(int t)

If t is 1, displays message that card is being checked. If t==2, calls store_cash and display_menu on the output processor O. If t==3, calls enter_pin and store_pin on the output processor O.

F. S2 Class //inherits from State

The S2 class is the representation of “S2” in the state diagram. It overrides approved and reject operations.

a. S2 (OP o)

Constructor. Initializes OP pointer.

b. approved()

Notifies card is approved and calls display menu on output processor

c. reject()

Calls reject_msg on Output processor

G. S3 Class //inherits from State

Representation of “S3” in the state diagram. Overrides select_gas(int, int), cancel and continue operations.

a. S3 (OP o)

Constructor. Initializes OP pointer.

b. select_gas(int g, int m)

Calls set_price(g,m) on output processor (OP)

c. cancel()

Calls cancel_msg on output processor

d. Continue()

Continues

H. S4 Class //inherits from State

Representation of “S4” in the state diagram. Overrides start pump operation.

a. S4 (OP o)

Constructor. Initializes OP pointer.

b. start_Pump

Calls set_init_values and ready_msg on the output processor (OP).

I. S5 Class //inherits from State

Representation of “S5” in the state diagram. Overrides pump and stop pump operations.

a. S5 (OP o)

Constructor. Initializes OP pointer.

b. pump()

Calls pump_gas_unit and gas_pumped_msg on the output processor.

c. stop_Pump()

Calls stop_msg on the output processor.

J. S6 Class //inherits from State

Representation of “S6” in state diagram. Overrides receipt and noReceipt operations.

a. S6 (OP o)

Constructor. Initializes OP pointer.

b. receipt()

Calls return_cash and print_receipt on the output processor.

c. noReceipt()

Displays message that no receipt was selected and calls return_cash on the output processor.

K. S7 Class //inherits from State

Representation of “S7” in state diagram. Overrides incorrect pin and correct pin operations.

a. S7 (OP o)

Constructor. Initializes OP pointer.

b. incorrect_Pin()

Calls wrong_pin_msg on the output processor.

c. correct_Pin()

Calls display_menu on the output processor.

4. OP Class (Package)

The OP contains the output processor information and operations. It is in charge of processing the appropriate actions that need to be done. It calls on the appropriate strategy classes to complete the necessary actions.

1. Attributes

a. Data_Store *d

b. Abstract_Factory *AF

c. Store_Price *SP

d. Display_Menu *DM

e. Pump_Gas_Unit *PG

f. Gas_Pump_Msg *GPM

g. Pay_Msg *PM

h. Set_Init_Values *SI

i. Enter_Pin_Msg *EPM

j. Initialize_Data *ID

k. Store_Cash *SC

l. Store_Pin *STP_1

m. Wrong_Pin_Msg *WPM

n. Reject_Msg *RM

o. Return_Cash *RC

p. Ready_Msg *Ready

q. Print_Receipt *PR

r. Cancel_Msg *CM

s. Set_Price *Set_P

t. Stop_Msg *SM

2. Operations

a. public OP(Abstract_Factory AF1, Data_Sotre ds)

Constructor for OP class. Initializes AF and d pointers to paramters.

b. public void store_Price()

Calls AF for a reference to SP. Calls for temp_a, temp_b and temp_c from data store and calls store_price on SP.

c. public void pay_Msg()

Calls AF for a reference to PM and calls pay_msg on PM.

d. public void store_cash()

Calls AF for a reference to SC and calls store_cash on SC.

e. public void display_menu()

Calls AF for a reference to DM and calls display_menu on DM.

f. public void reject_msg()

Calls AF for a reference to RM and calls reject_msg on RM.

g. public void set_price(int g, int M)

Calls AF for a reference to Set_P and calls set_Price(g,M,d)

On Set_P.

h. public void set_init_values()

Calls AF for a reference to SI and calls set_init_values(d) on SI.

i. public void ready_msg()

Calls AF for a reference to pointer “Ready” and calls ready_msg on Ready.

j. public void pump_gas_unit()

Calls AF for a reference to PG and calls pump_gas_unit(d) on PG.

k. public void gas_pumped_msg()

Calls AF for a reference to GPM and calls gas_pump_msg on GPM.

l. public void stop_msg()

Calls AF for a reference to SM and calls stop_msg on SM.

m. public void print_receipt()

Calls AF for a reference to PR and calls print_receipt on PR.

n. public void cancel_msg()

Calls AF for a reference to CM and calls cancel_msg on CM.

o. public void return_cash()

Calls AF for a reference to RC. If RC is not null, checks to see if there is any cash stored in the data_store. If there is, it calls return_cash on RC.

p. public void wrong_pin_msg()

Calls AF for a reference to WPM. Calls wrong_pin_msg on WPM.

q. public void store_pin()

Calls AF for a reference to STP_1. Calls store_pin(d) on STP_1.

r. public void enter_pin()

Calls AF for a reference to EPM. Calls enter_pin_msg on EPM.

s. public void initialize_data()

Calls AF for a reference to ID. Calls Initialize_data on ID.

5. Abstract Factory Classes (Package)

A. Abstract_Factory // super class, is abstract

Abstract_Factory is the interface that the Concrete_Factories implement.

1. Classes // These class are all abstract
   a. public abstract Data_Store create_Data_Store()
   b. public abstract Store_Price create_Store_Price()
   c. public abstract Pay_Msg create_Pay_Msg()
   d. public abstract Display_Menu create_Display_Menu()
   e. public abstract Set_Init_Values create_Set_Init_Values()
   f. public abstract Pump_Gas_Unit create_Pump_Gas_Unit()
   g. public abstract Gas_Pump_Msg create_Gas_Pump_Msg()
   h. public abstract Enter_Pin_Msg create_Enter_Pin_Msg()
   i. public abstract Initialize_Data create_Initialize_Data()
   j. public abstract Store_Cash create_Store_Cash()
   k. public abstract Reject_Msg create_Reject_Msg()
   l. public abstract Set_Price create_Set_Price()
   m. public abstract Stop_Msg create_Stop_Msg()
   n. public abstract Print_Receipt create_Print_Receipt()
   o. public abstract Cancel_Msg create_Cancel_Msg()
   p. public abstract Return_Cash create_Return_Cash()
   q. public abstract Wrong_Pin_Msg create_Wrong_Pin_Msg()
   r. public abstract Store_Pin create_Store_Pin()
   s. public abstract Ready_Msg create_Ready_Msg()
   t.

B. CF_GP_1 //extends Abstract_Factory

CF_GP_1 is in charge of delivering the appropriate objects needed by other objects in the Gas Pump 1 model. Both the IP and the OP get references to needed objects via the operations of CF_GP_1.

1. Attributes:

DS_1 *d1 // a pointer to DS_1

2. Operations

a. public Data_Store create_Data_Store()
   If d1 does not exist, it creates data_Store and returns it. If it exists, it returns it.
b. public Store_Price create_Store_Price()
   Creates and returns a Store_Price object.
c. public Pay_Msg create_Pay_Msg()
   Creates and returns a Pay_Msg object.
d. public Display_Menu create_Display_Menu()
   Creates and returns a Display_Menu object.
e. public Set_Init_Values create_Set_Init_Values()
   Creates and returns a Set_Init_Values object.
f. public Pump_Gas_Unit create_Pump_Gas_Unit()
   Creates and returns a Pump_Gas_Unit object.
g. public Gas_Pump_Msg create_Gas_Pump_Msg()
   Creates and returns a Gas_Pump_Msg object.
h. public Enter_Pin_Msg create_Enter_Pin_Msg()
   Creates and returns an Enter_Pin_Msg object.
i. public Initialize_Data create_Initialize_Data()

Creates and returns an Initialize_Data object.

j. public Store_Cash create_Store_Cash()

returns null

k. public Reject_Msg create_Reject_Msg()
Creates and returns a Reject_Msg object.

l. public Set_Price create_Set_Price()
Creates and returns a Set_Price object.

m. public Stop_Msg create_Stop_Msg()
Creates and returns a Stop_Msg() object

n. public Print_Receipt create_Print_Receipt()
Creates and returns a Print_Receipt object.

o. public Cancel_Msg create_Cancel_Msg()
Creates and returns a Cancel_Msg object

p. public Return_Cash create_Return_Cash()
returns null

q. public Wrong_Pin_Msg create_Wrong_Pin_Msg()
Creates and returns a Wrong_pin_msg object.

r. public Store_Pin create_Store_Pin()
Creates and returns a Store_Pin object.

s. public Ready_Msg create_Ready_Msg()
Creates and returns a Ready_Msg object.

C. CF_GP_2 Class

CF_GP_2 is in charge of delivering the appropriate objects needed by other objects in the Gas Pump 2 model. Both the IP and the OP get references to needed objects via the operations of CF_GP_2.

1. Operations
   a. public Data_Store create_Data_Store()
      Creates and returns a Data_Store object.
   b. public Store_Price create_Store_Price()
      Creates and returns a Store_Price object.
   c. public Pay_Msg create_Pay_Msg()
      returns null
   d. public Display_Menu create_Display_Menu()
      Creates and returns a Display_Menu object.
   e. public Set_Init_Values create_Set_Init_Values()
      Creates and returns a Set_Init_Values object.
   f. public Pump_Gas_Unit create_Pump_Gas_Unit()
      Creates and returns a Gas_Pump_Unit object.
   g. public Gas_Pump_Msg create_Gas_Pump_Msg()
      Creates and returns a Gas_Pump_Msg object.
   h. public Enter_Pin_Msg create_Enter_Pin_Msg()
      Creates and returns an Enter_Pin_Msg object.
   i. public Initialize_Data create_Initialize_Data()
      Creates and returns an Initialize_Data object.
   k. public Store_Cash create_Store_Cash()
      Creates and returns a Store_Cash object.
   l. public Reject_Msg create_Reject_Msg()
      Creates and returns a Reject_Msg object.
   m. public Set_Price create_Set_Price()

Creates and returns a Set_Price object.

n. public Stop_Msg create_Stop_Msg()
   Creates and returns a Stop_Msg() object
o. public Print_Receipt create_Print_Receipt()
   Creates and returns a Print_Receipt object.
p. public Cancel_Msg create_Cancel_Msg()
   Creates and returns a Cancel_Msg object
q. public Return_Cash create_Return_Cash()
   Creates and returns a Return_Cash object
r. public Wrong_Pin_Msg create_Wrong_Pin_Msg()
   returns null
s. public Store_Pin create_Store_Pin()
   returns null
t. public Ready_Msg create_Ready_Msg()
   Creates and returns a Ready_Msg object.

6. Strategy Classes (Package)

All of the classes in the strategy package are self-described. They are in charge of carrying out the appropriate actions in the name (e.g. Store_Price is in charge of storing price, Display_Menu in charge of displaying menu). The super class methods are overridden by their corresponding classes.

A. Store_Price //abstract super class
   a. SP_1 //extends Store_Price
      Overrides store_price() and sets data for Regular and Diesel gas in data store
   b. SP_2 // extends Store_Price
      Overrides store_price() and sets data for Super, Regular and Premium
B. Pay_Msg //abstract super class
   a. PM_1 //extends Pay_Msg
      Overrides pay_msg(). Displays Debit or Credit payment method options.
C. Display_Menu //abstract super class
   a. DM_1 //extends Display_Menu
      Overrides display_menu() and displays gas options of Regular and Diesel
   b. DM_2 //extends Display_Menu
      Overrides display_menu() and displays gas types of Super, Regular and Premium.

D. Set_Init_Values //abstract super class

a. SI_1 //extends Set_Init_Values
   Overrides set_init_values(Data_Store d) and sets G to 0 and total to 0 in data store.
b. SI_2 //extends Set_Init_Values
   Overrides set_init_values(Data_Store d) and sets L to 0 and total to 0 in the data store

E. Pump_Gas_Unit //abstract super class

a. PG_1 //extends Pump_Gas_Unit
   Overrides pump_gas_unit() and increases G by 1 in the data store; updates the total price of gas purchased in Data store.
b. PG_2 //extends Pump_Gas_Unit
   Overrides pump_gas_unit() and increases L by 1 in the data store; updates the total price of gas purchased in data store.

F. Gas_Pump_Msg //abstract super class

a. GPM_1 //extends Gas_Pump_Msg
   Overrides gas_pump_msg() and displays gallons pumped and current total.
b. GPM_2 //extends Gas_Pump_Msg
   Overrides gas_pump_msg() and displays liters pumped and current total.

G. Stop_Msg //abstract super class

a. SM_1 //extends Stop_Msg
   Overrides stop_msg() and notifies printing receipt
b. SM_2 //extends Stop_Msg
   Overrides stop_msg() and displays option for a receipt or no receipt

H. Initialize_Data //abstract super class
    a. ID_1 //extends Initialize_Data
       Overrides initialize_data() and sets price to 0.0 in data store.
    b. ID_2 //extends Initialize_Data
       Overrides initialize_data() and sets price and cash fields to 0.0 in data store.

I. Reject_Msg //abstract super class
    a. RM_1 //extends Reject_Msg
       Overrides reject_msg() and displays message saying card is rejected and transaction canceled

J. Cancel_Msg //abstract super class
    a. CM_1 //extends Cancel_msg
       Overrides cancel_msg() and displays message saying transaction is canceled.

K. Print_Receipt //abstract super class
    a. PR_1 //extends print_receipt
       Overrides print_receipt(), displaying number of gallons sold and total cost of transaction.
    b. PR_2 //extends print_receipt
       Overrides print_receipt(), displaying number of liters sold and total cost of transaction. If cash needs to be returned, displays amount of cash to be refunded

L. Ready_Msg //abstract super class
    a. Ready_M1 //extends Ready_msg
       Overrides ready_msg() and displays message saying pump Is ready.

M. Wrong_Pin_Msg //abstract super class

a. WPM_1 //extends wrong_pin_msg

Overrides wrong_pin_msg(), displays message saying wrong pin entered and transaction is canceled.

N. Store_Pin //abstract super class

a. STP_1 //extends store_pin

Overrides store_pin(). Gets temporary pin and stores it to pin field in data store.

O. Enter_Pin_Msg //abstract super class

a. EP_1 //extends enter_pin_msg

Overrides enter_pin_msg and displays message to enter pin.

P. Set_Price //abstract super class

a. Set_P1 //extends set_price

Overrides set_price(int g, int M, Data_Store d). If M ==1, all prices are increased by multiplier of 1.1.
If g==1, Regular gas price is selected as price
If g==2, Super gas price is selected as price
If g==3, Premium gas price is selected as price
If g==4, Diesel gas price is selected as price.

Q. Store_Cash //abstract super class

a. SC_1 //extends store_cash

Overrides store_cash(). Gets temp_cash value from data store and store it as value for cash field.

R. Return_Cash //abstract super class

a. RC_1 //extends return_cash

Overrides return_cash(), displaying amount of cash needed to be returned. Subtracts total from cash stored in data store.

7. Data Store Classes (Package)

A. Data_Store //interface

The Data_Store interface is in charge of storing the appropriate data for the corresponding Gas Pump models. It "gets" and "sets" data.

1. Operations

a. public Object get_Data(String type)

b. void set_Data(Object a, String type)

B. DS_1 //implements Data_Store

DS_1 stores the necessary data for the Gas Pump 1 model. It returns and sets the necessary data needed by the IP and OP of Gas Pump 1.

1. Attributes

a. float temp_a
b. float temp_b
c. String temp_p
d. String pin
e. float price
f. int G
g. float total
h. float Rprice
i. float Dprice

2. Operations

a. public Object get_Data(String type)

If type matches any of the names of the attributes of DS_1, the value of that field is returned by calling get_<field name>.

b. void set_Data(Object a, String type)

If type matches any of the names of the attributes of DS_1, the value of a is stored in that field by calling set_<field name> of the corresponding field.

c. float get_temp_a()

Returns a float temp_a

d. void set_temp_a(double a)

Stores a value in temp_a field. Casts it as a float.

e. get_temp_b()

Returns a float temp_b

f. set_temp_b (double b)

Stores a value in temp_b field. Casts it as a float

g. String get_temp_p()

Returns String temp_p

h. set_temp_p(String tpin)

Sets String temp_p()

i. String get_pin

Returns String pin

j. set_pin(String pin2)

Sets String pin

k. float get_price()

Returns price

l. set_price(float aprice)

Sets a value in price field.

m. int get_G()

Returns G value

n. set_G(int g)

Sets a value in G field

o. float get_total()

Returns value of total field

p. set_total(double atotal)

Sets the value of the total field

q. float get_Rprice()

Returns the value of the Rprice field

r. set_Rprice(float R_price)

Sets the value of the Rprice field

s. float get_Dprice()

Returns the value of the Dprice field

t. set_Dprice(float D_price)

Sets the value of the Dprice field

C. DS_2 //implements Data_Store

DS_2 stores the necessary data for the Gas Pump 2 model. It returns and sets the necessary data needed by the IP and OP of Gas Pump 2.

1. Attributes

a. float temp_a
b. float temp_b
c. float temp_c
d. float temp_cash
e. float Sprice

f. float Rprice
g. float Pprice
h. float cash
i. float total
j. int L
k. float price

2. Operations

a. public Object get_Data(String type)

If type matches any of the names of the attributes of DS_1, the value of that field is returned by calling get_<field name> of the corresponding field

b. void set_Data(Object a, String type)

If type matches any of the names of the attributes of DS_1, the value of a is stored in that field by calling set_<field name> of the corresponding field.

c. float get_temp_a()

Returns the value of the temp_a field.

d. set_temp_a(double a)

Sets the value of the temp_a field. Casts to float.

e. float get_temp_b()

Returns the value of the temp_b field.

f. set_temp_b(float b)

Sets the value of the temp_b field. Casts to float

g. float get_temp_c()

Returns the value of the temp_c field.

h. set_temp_c(double c)

Sets the value of the temp_c field. Casts to float.

i. float get_temp_cash()

Returns the value of temp_cash field

j. set_temp_cash(double tcash)

Sets the value of the temp_cash field. Casts to float.

k. float get_Sprice()

Returns the value of Sprice field.

l. set_Sprice(double S_price)

Sets the value of Sprice. Casts to float.

m. float get_Rprice()

Returns the value of Rprice field

n. set_Rprice(double R_price)

Sets the value of Rprice. Casts to float.

o. float get_Pprice()

Returns the value of Pprice field.

p. set_Pprice(double P_price)

Sets the value of Pprice. Casts to float.

q. float get_cash()

Returns the value of cash field.

r. set_cash(double cash2)

Sets the value of the Cash field. Casts to float.

s. float get_total()

Returns the value of the total field.

t. set_total(double total2)

Sets the value of the total field. Casts to float.

u. int get_L()

Returns the value of the L field.

v. set_L(int liters)

Sets the value of the L field.

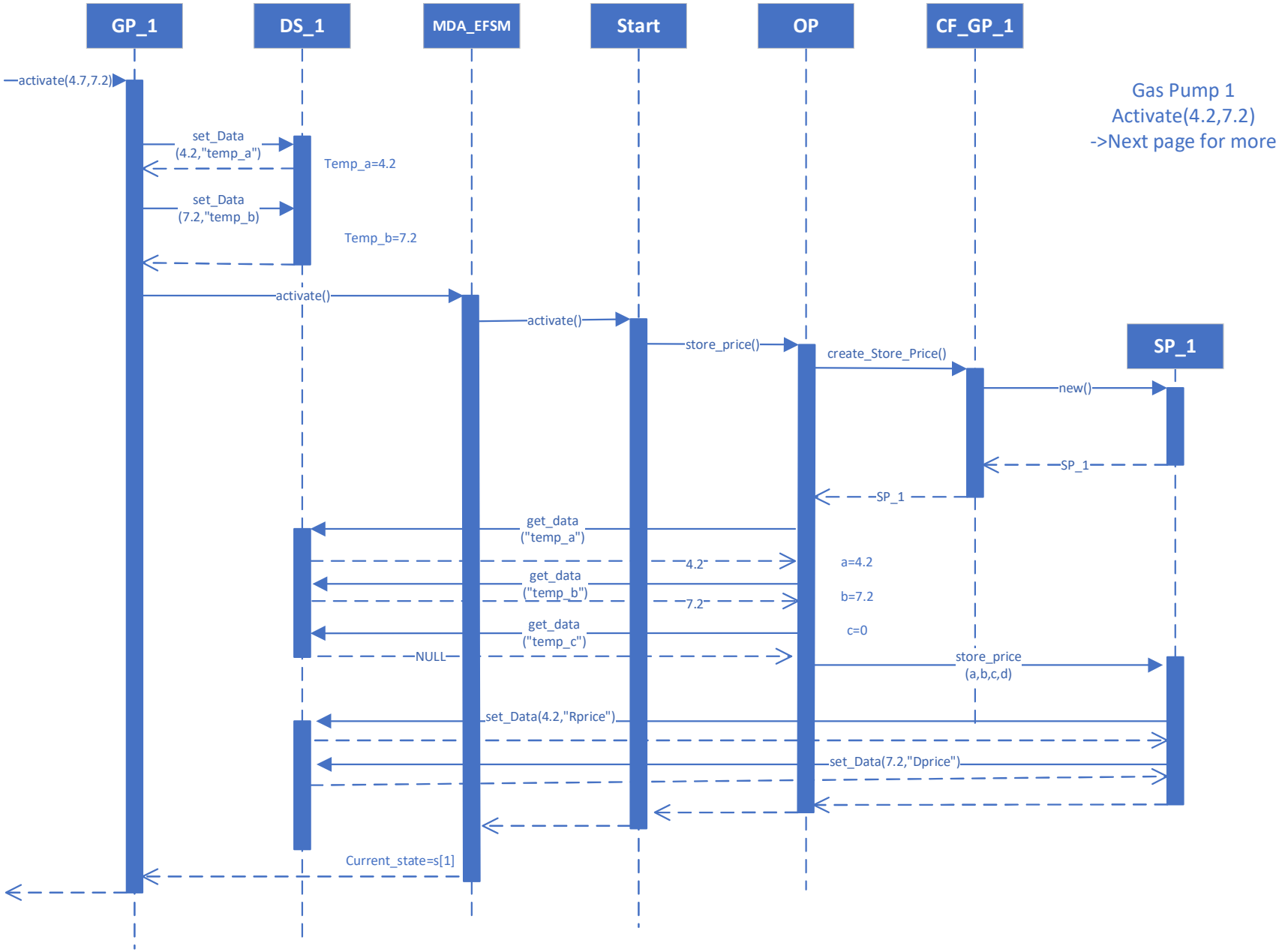
w. float get_price()

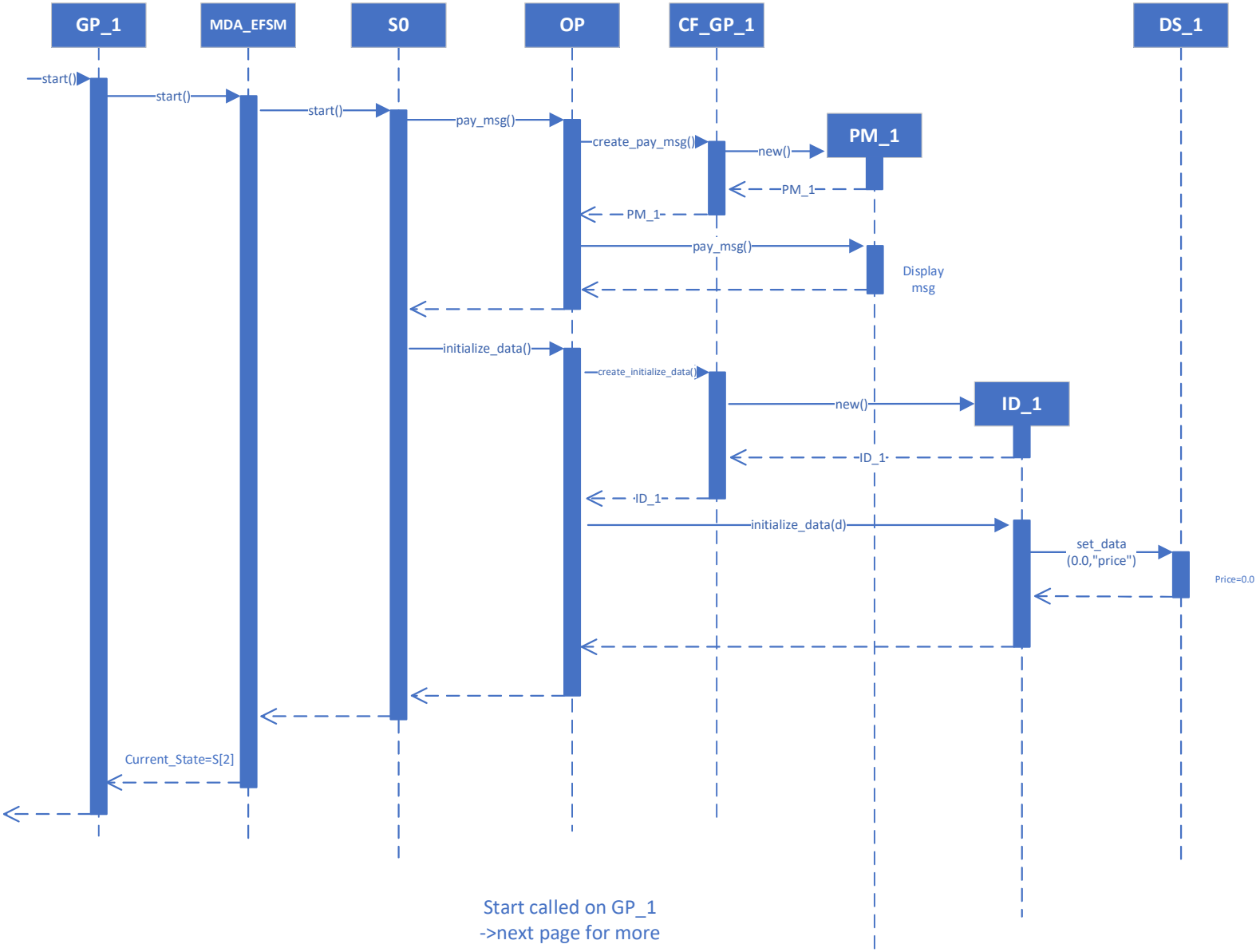
Returns the value of the price field.

x. set_price(double price2)

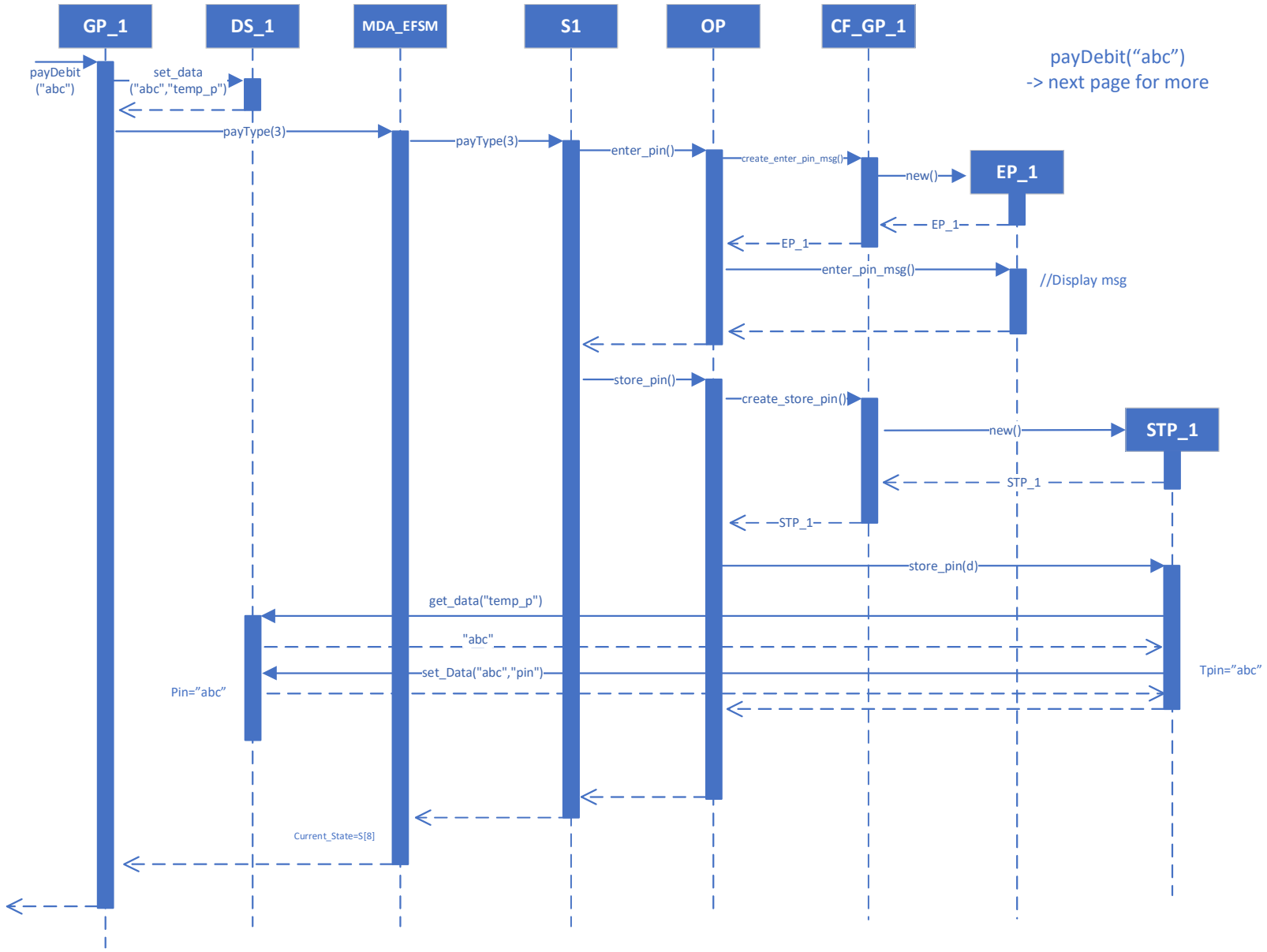
Sets the value of the price field. Cast to float.

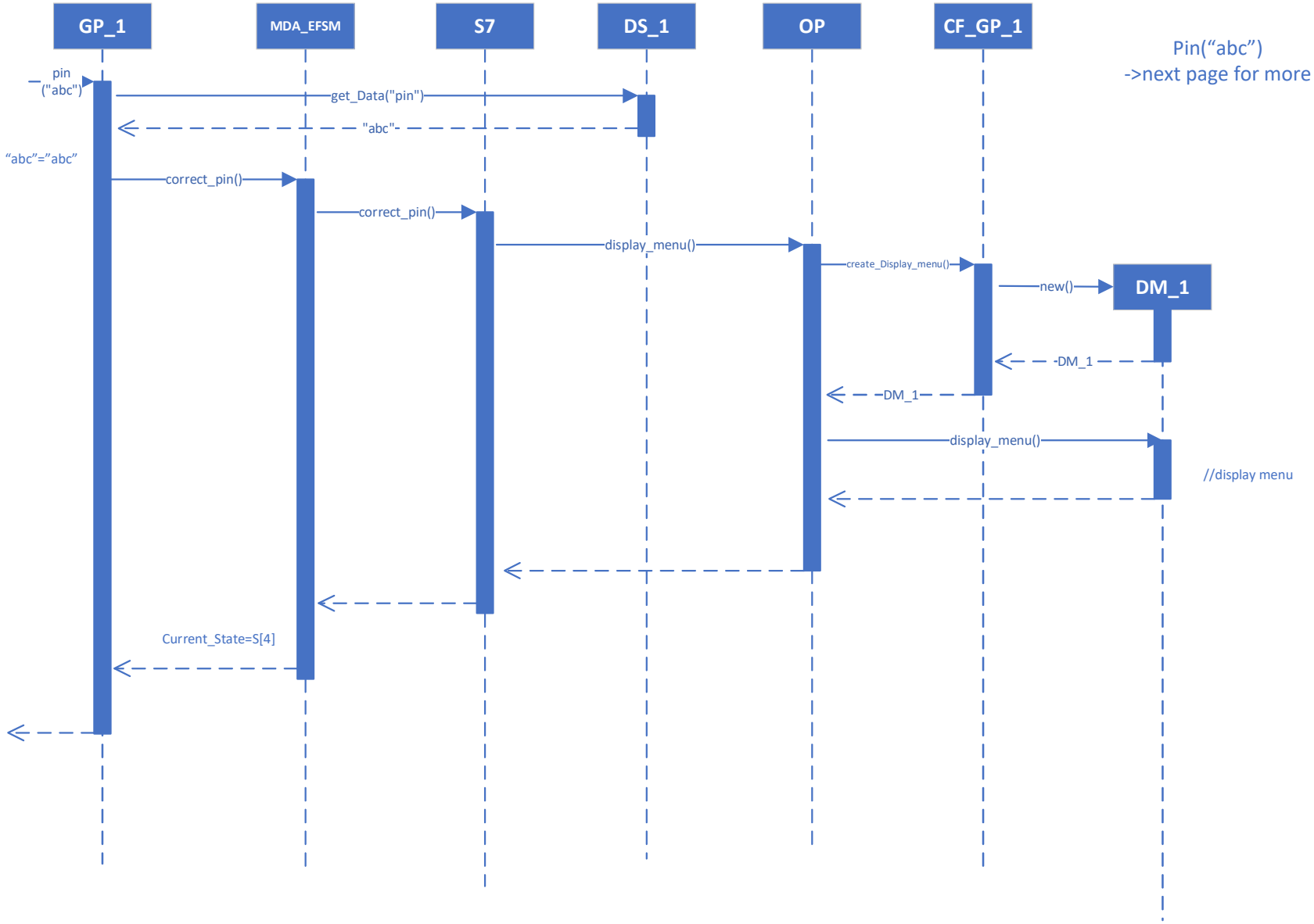
4. Sequence Diagrams

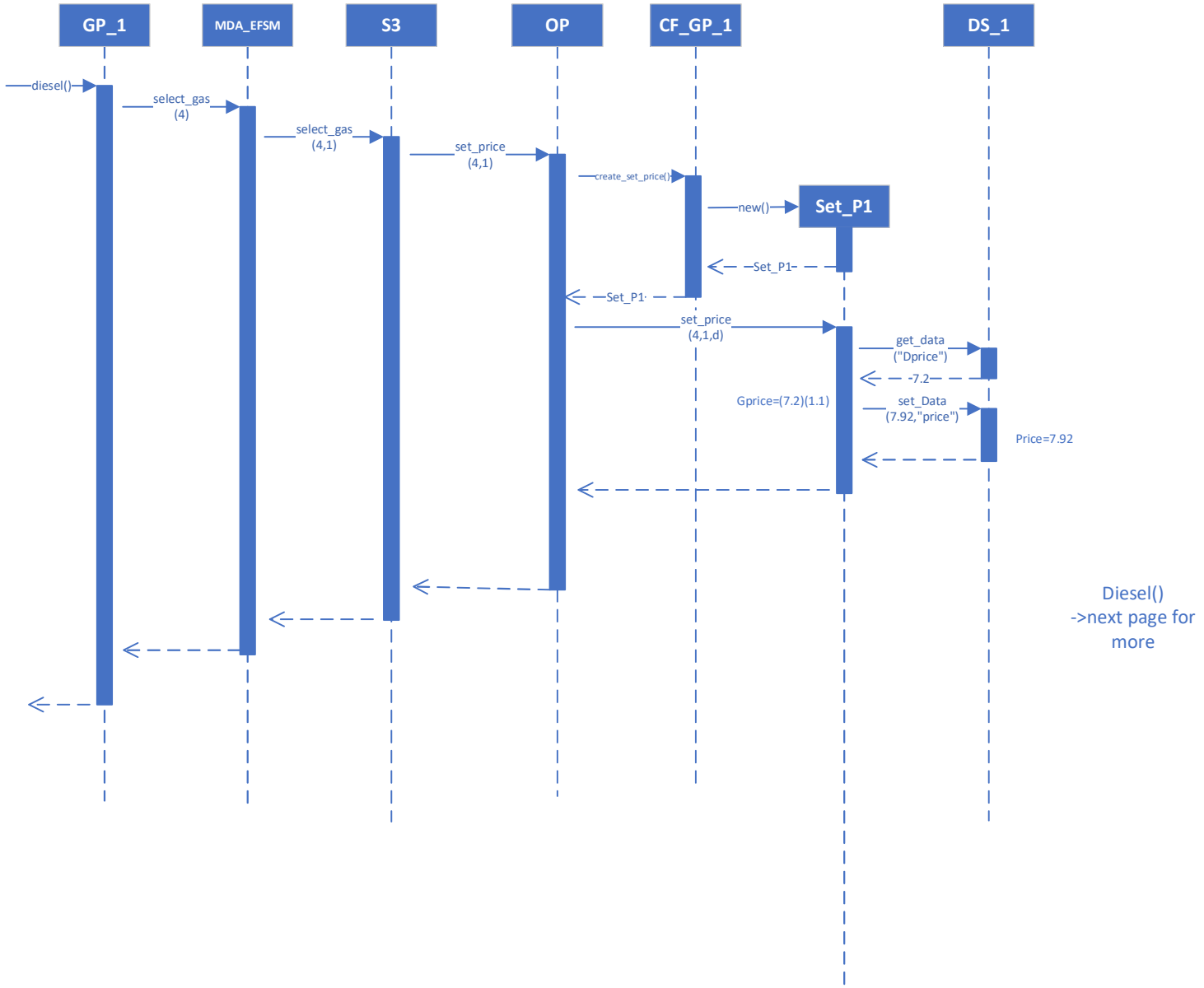

GP_1
DS_1
MDA_EFSM
Start
OP
CF_GP_1
SP_1
activate(4.7,7.2)
set_Data
(4.2,"temp_a")
Temp_a=4.2
set_Data
(7.2,"temp_b)
Temp_b=7.2
activate()
activate()
store_price()
create_Store_Price()
new()
SP_1
SP_1
get_data
("temp_a")
4.2
a=4.2
get_data
("temp_b")
7.2
b=7.2
get_data
("temp_c")
c=0
NULL
store_price
(a,b,c,d)
set_Data(4.2,"Rprice")
set_Data(7.2,"Dprice")
Current_state=s[1]
Gas Pump 1
Activate(4.2,7.2)
->Next page for more

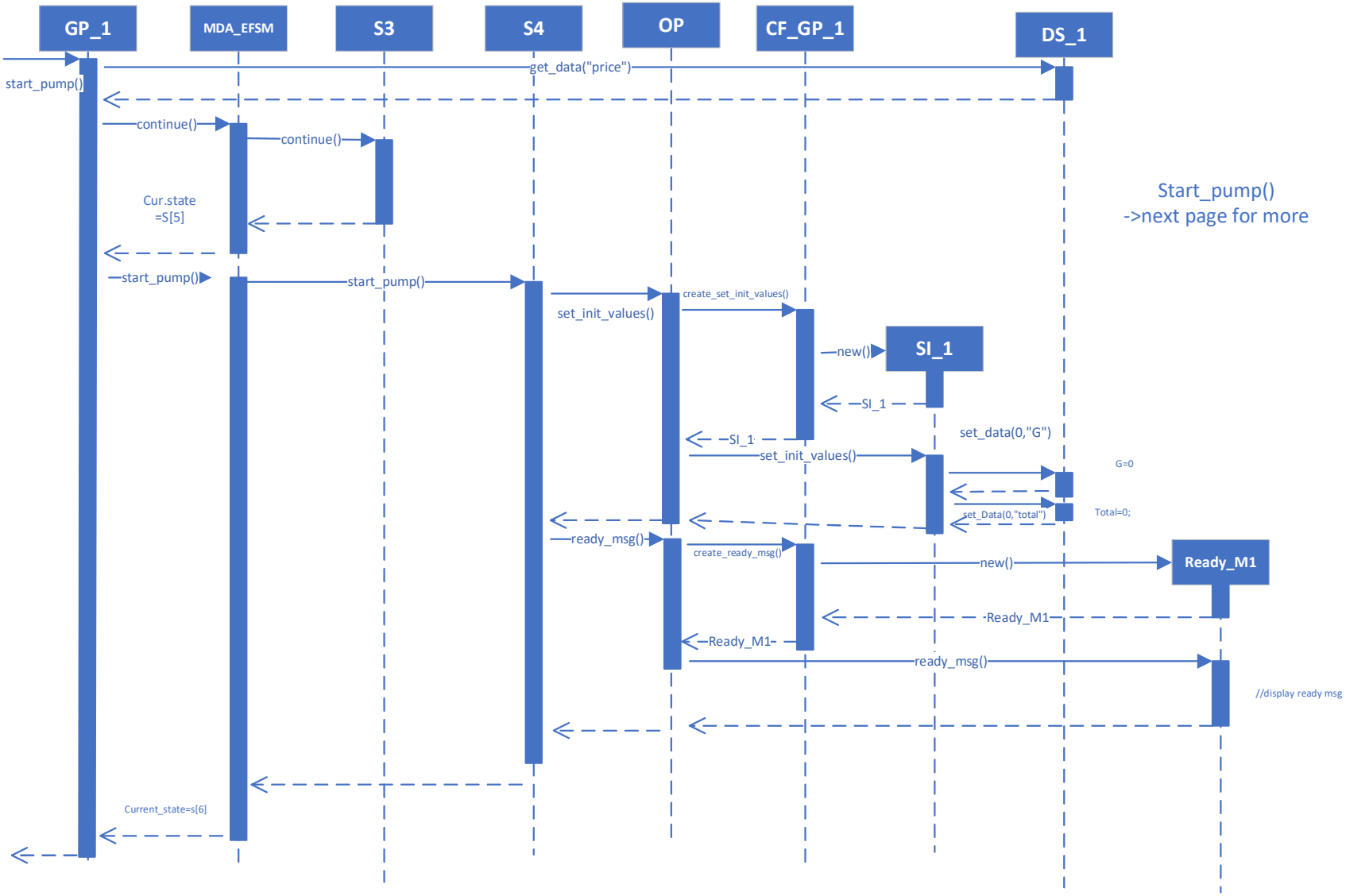


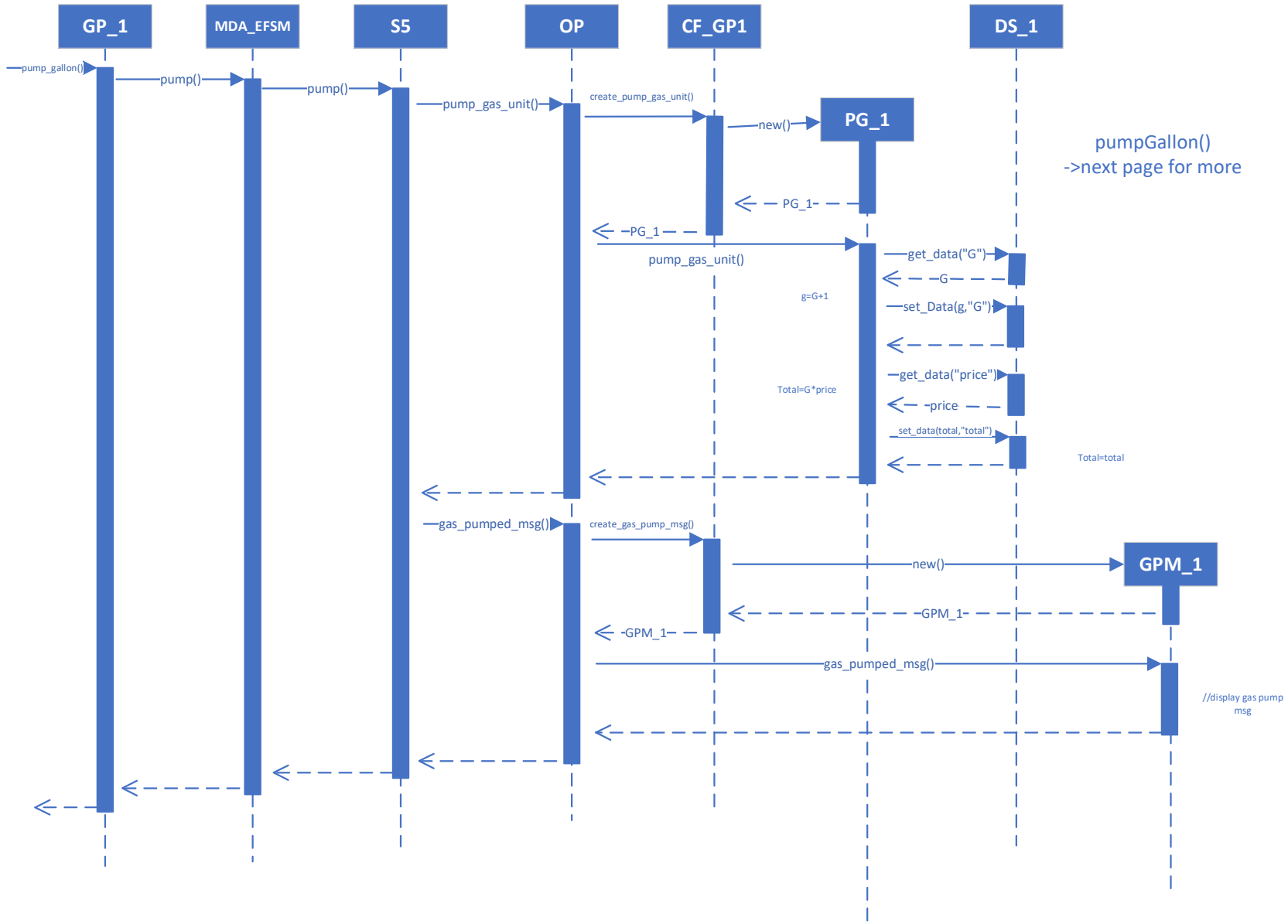
Start called on GP_1
->next page for more

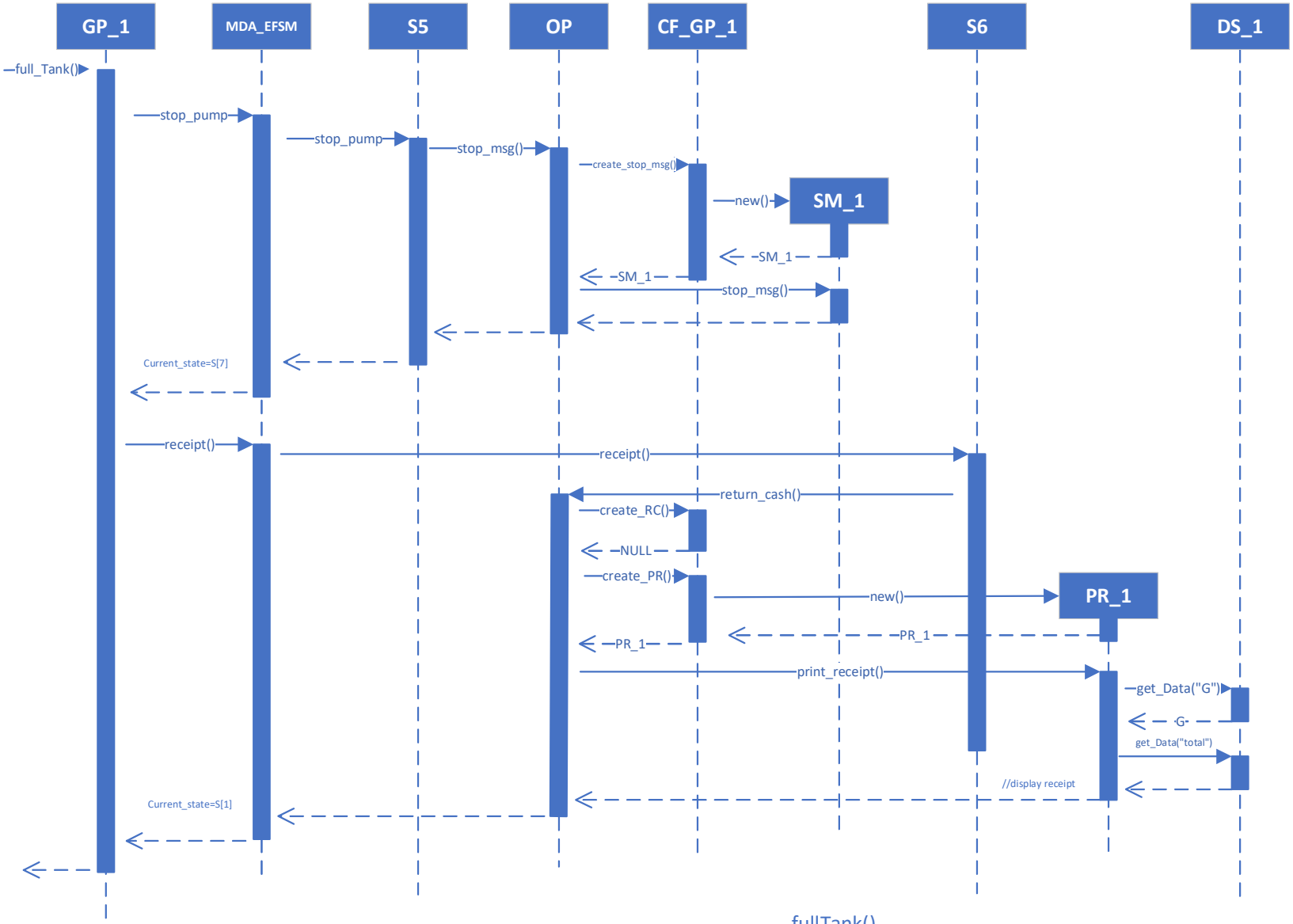

GP_1
DS_1
MDA_EFSM
S1
OP
CF_GP_1
payDebit("abc")
-> next page for more
payDebit ("abc")
set_data ("abc","temp_p")
payType(3)
payType(3)
enter_pin()
create_enter_pin_msg()
new()
EP_1
EP_1
EP_1
enter_pin_msg()
//Display msg
store_pin()
create_store_pin()
new()
STP_1
STP_1
STP_1
store_pin(d)
get_data("temp_p")
"abc"
set_Data("abc","pin")
Tpin=”abc”
Pin=”abc”
Current_State=S[8]


GP_1
MDA_EFSM
S7
DS_1
OP
CF_GP_1
DM_1
Pin("abc")
->next page for more
pin
("abc")
get_Data("pin")
"abc"
"abc"="abc"
correct_pin()
correct_pin()
display_menu()
create_Display_menu()
new()
DM_1
DM_1
DM_1
display_menu()
//display menu
Current_State=S[4]

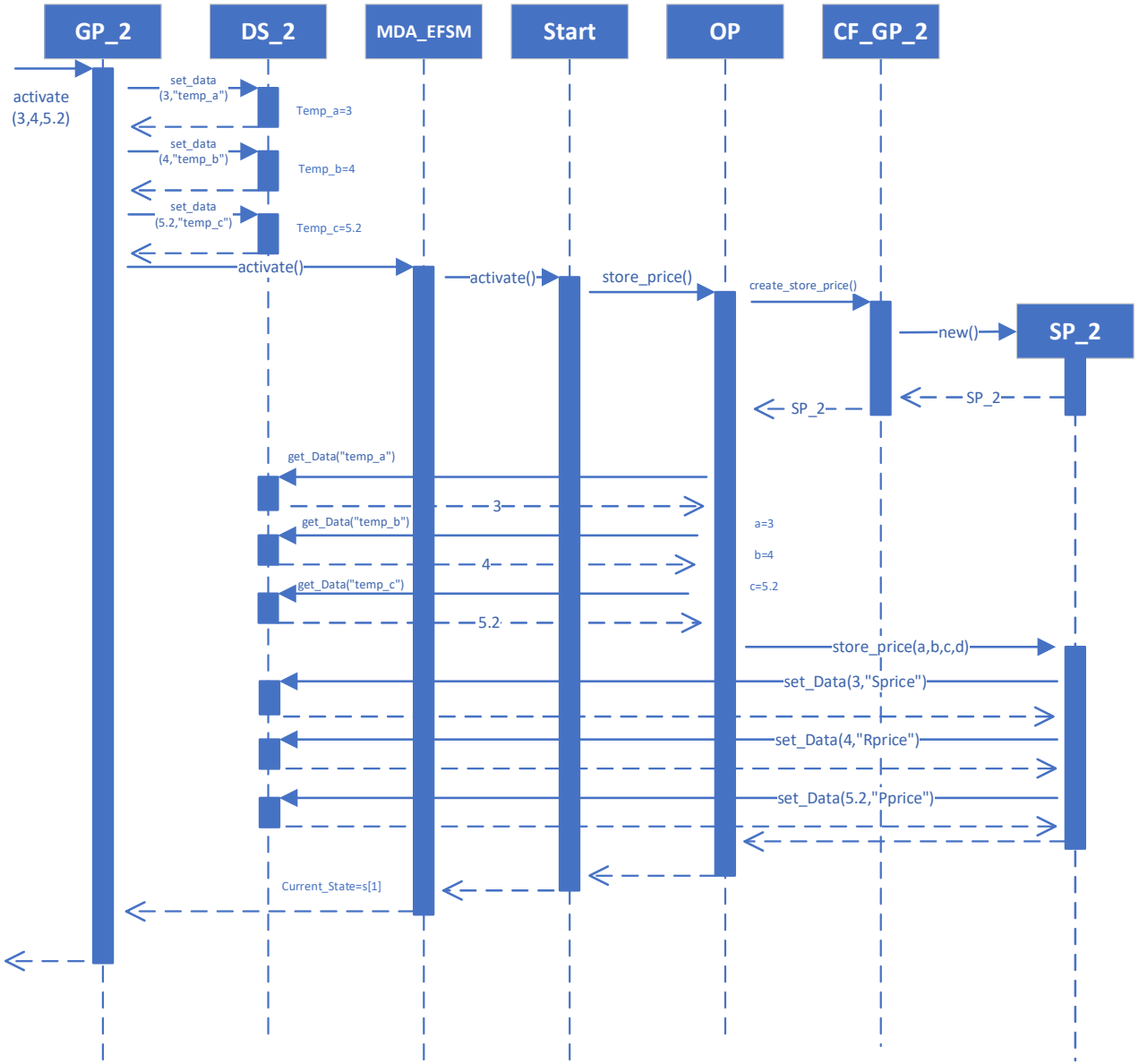

GP_1
MDA_EFSM
S3
OP
CF_GP_1
Set_P1
DS_1
diesel()
select_gas
(4)
select_gas
(4,1)
set_price
(4,1)
create_set_price()
new()
Set_P1
Set_P1
Set_P1
set_price
(4,1,d)
get_data
("Dprice")
-7.2
Gprice=(7.2)(1.1)
set_Data
(7.92,"price")
Price=7.92
Diesel()
->next page for
more


GP_1
MDA_EFSM
S3
S4
OP
CF_GP_1
DS_1
start_pump()
get_data("price")
continue()
continue()
Cur.state
=S[5]
start_pump()
start_pump()
create_set_init_values()
set_init_values()
new()
SI_1
SI_1
SI_1
set_data(0,"G")
set_init_values()
G=0
set_Data(0,"total")
Total=0;
ready_msg()
create_ready_msg()
new()
Ready_M1
Ready_M1
Ready_M1
ready_msg()
//display ready msg
Current_state=s[6]
Start_pump()
->next page for more

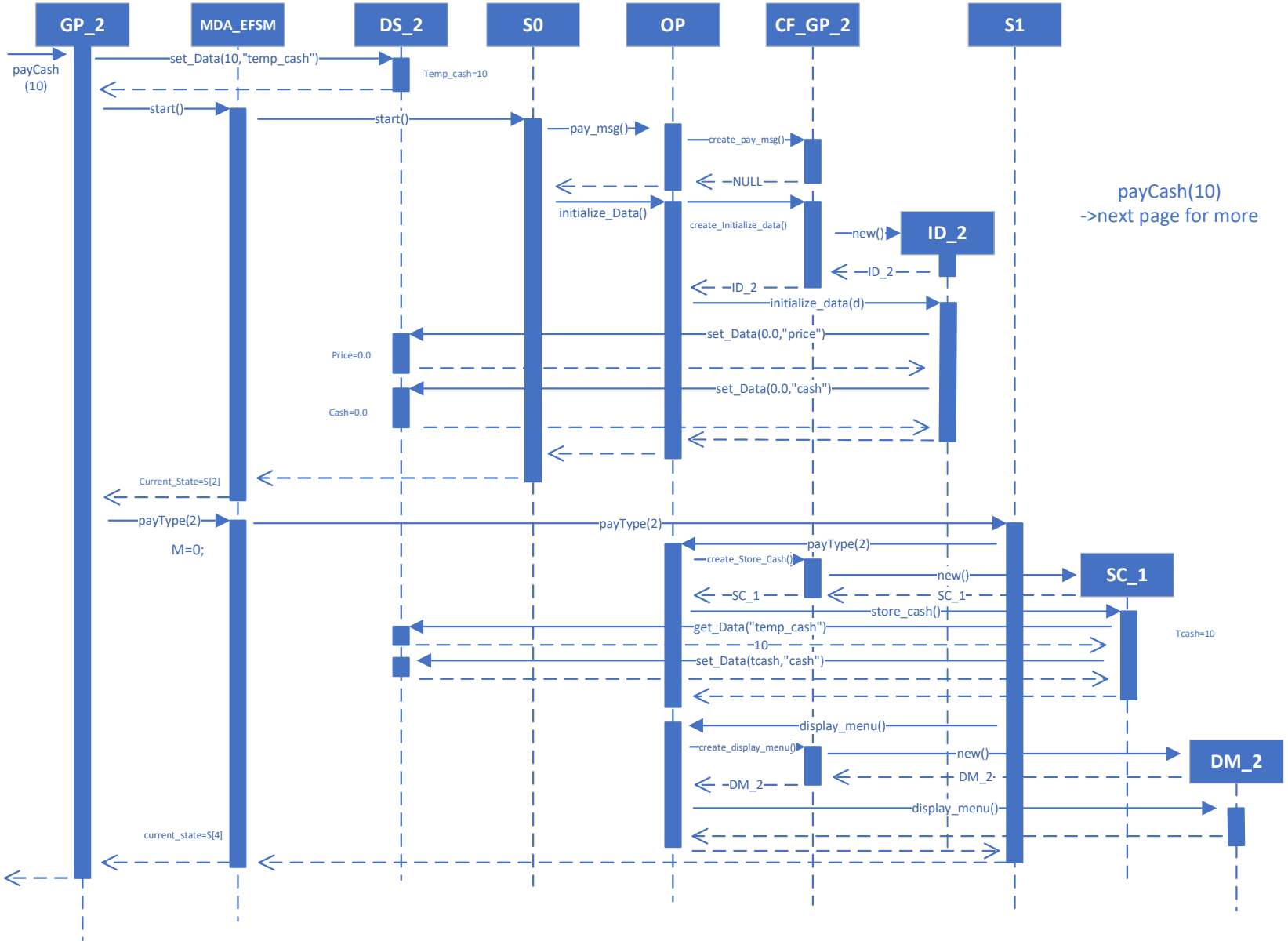

GP_1
MDA_EFSM
S5
OP
CF_GP1
DS_1
PG_1
GPM_1
pump_gallon()
pump()
pump()
pump_gas_unit()
create_pump_gas_unit()
new()
PG_1
PG_1
pump_gas_unit()
get_data("G")
G
g=G+1
set_Data(g,"G")
get_data("price")
Total=G*price
price
_set_data(total,"total")
Total=total
gas_pumped_msg()
create_gas_pump_msg()
new()
GPM_1
GPM_1
gas_pumped_msg()
//display gas pump msg
pumpGallon()
->next page for more

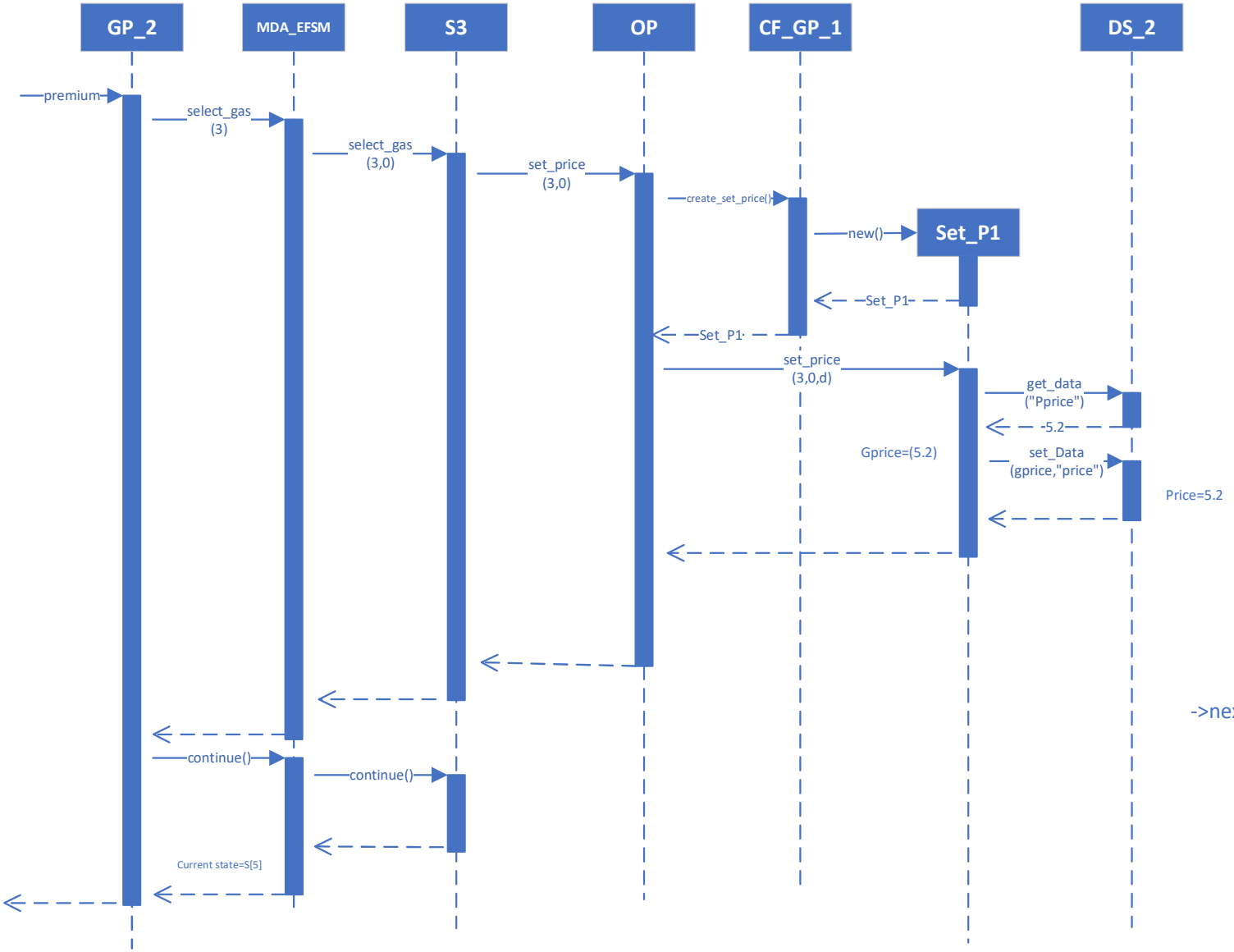


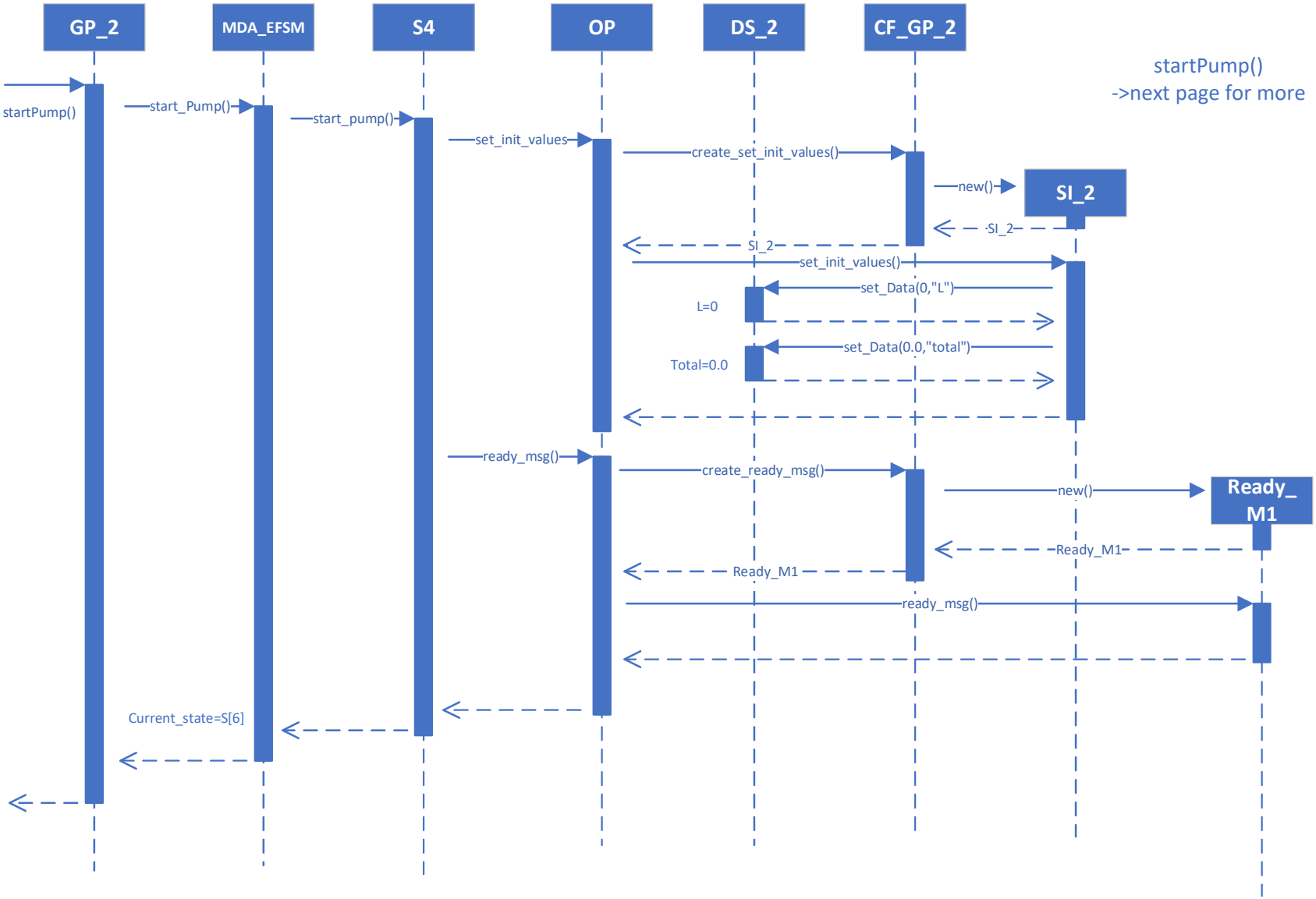
fullTank()
END

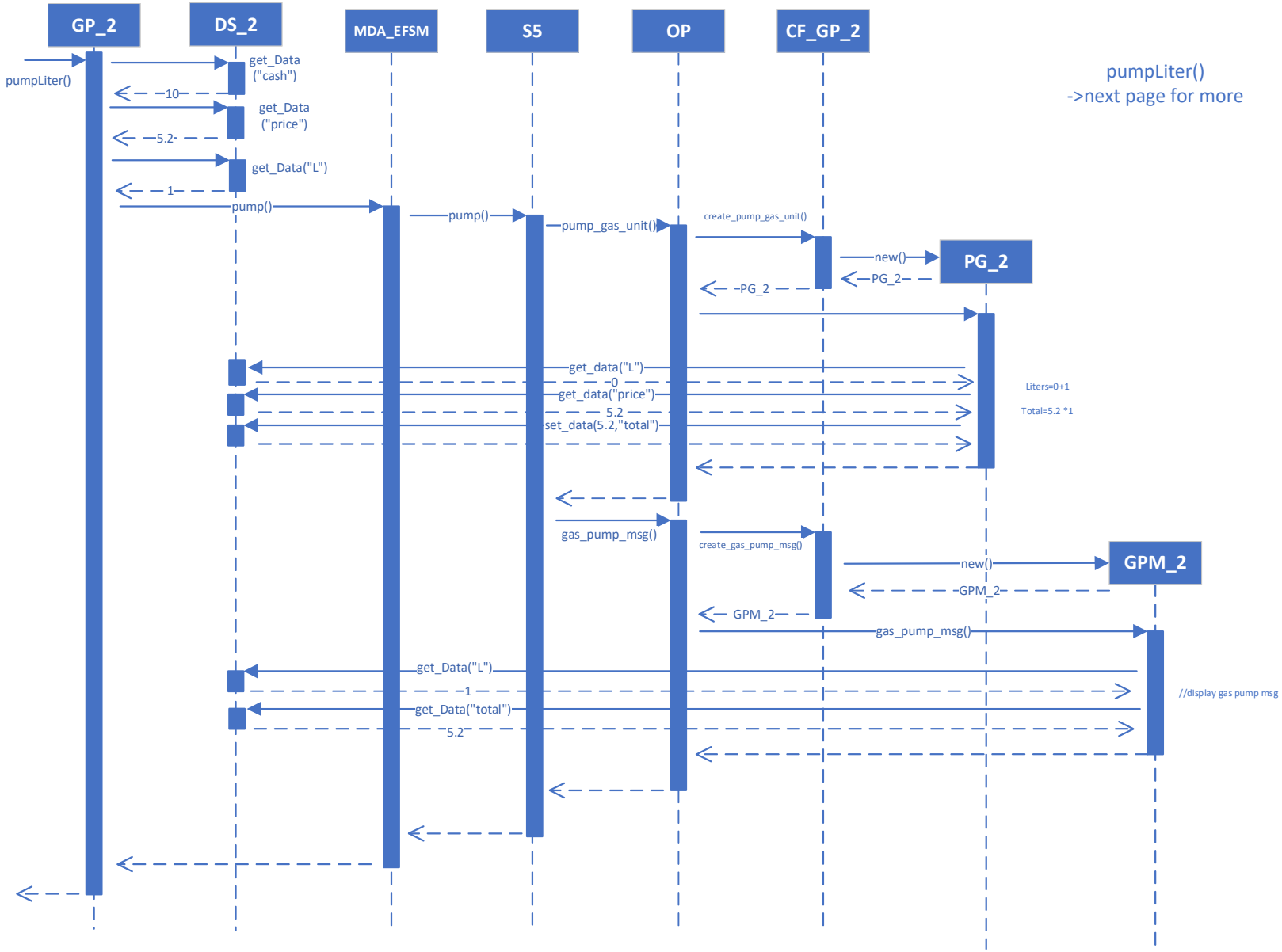


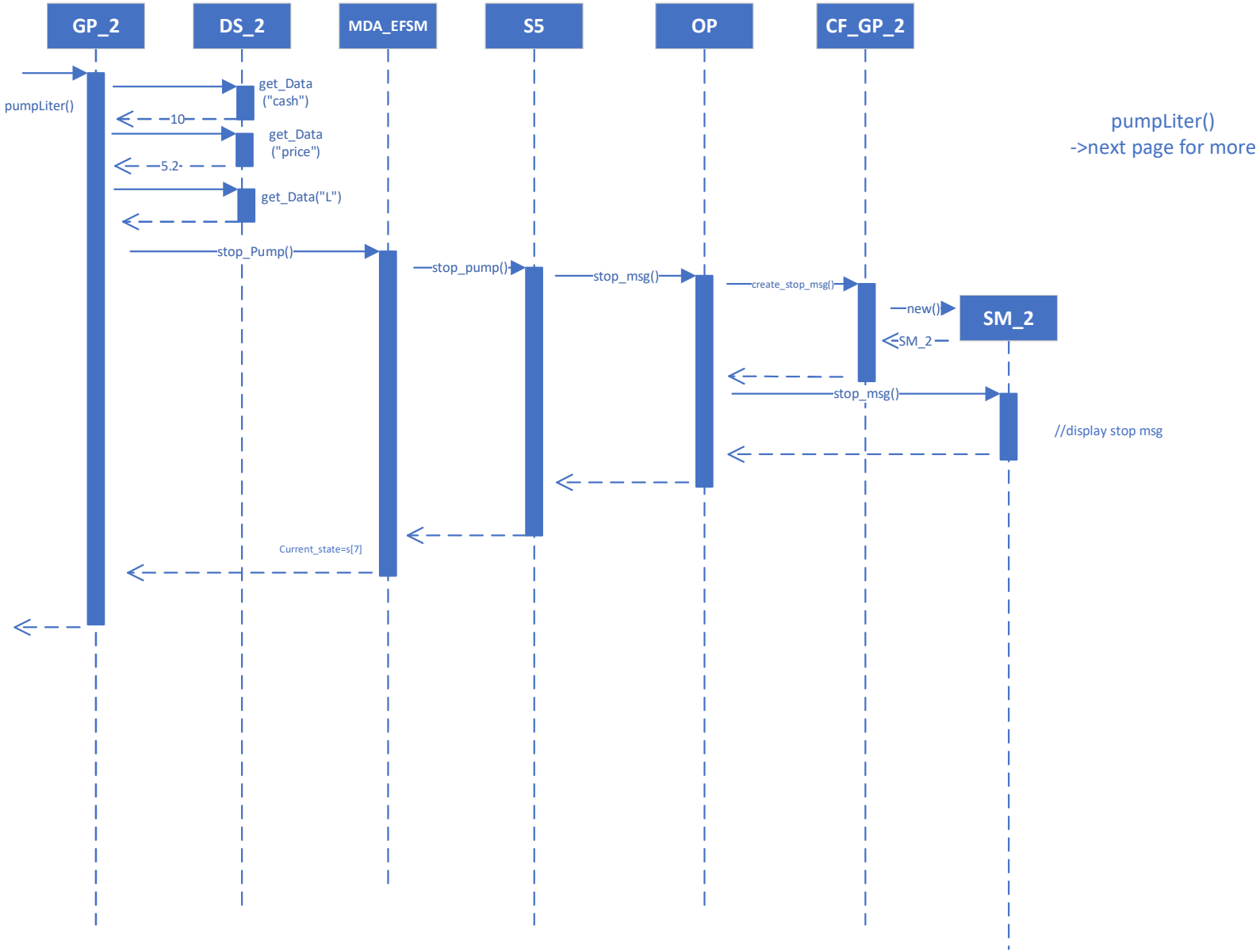
Gas Pump 2
Activate(3,4,5.2)
->next page for more

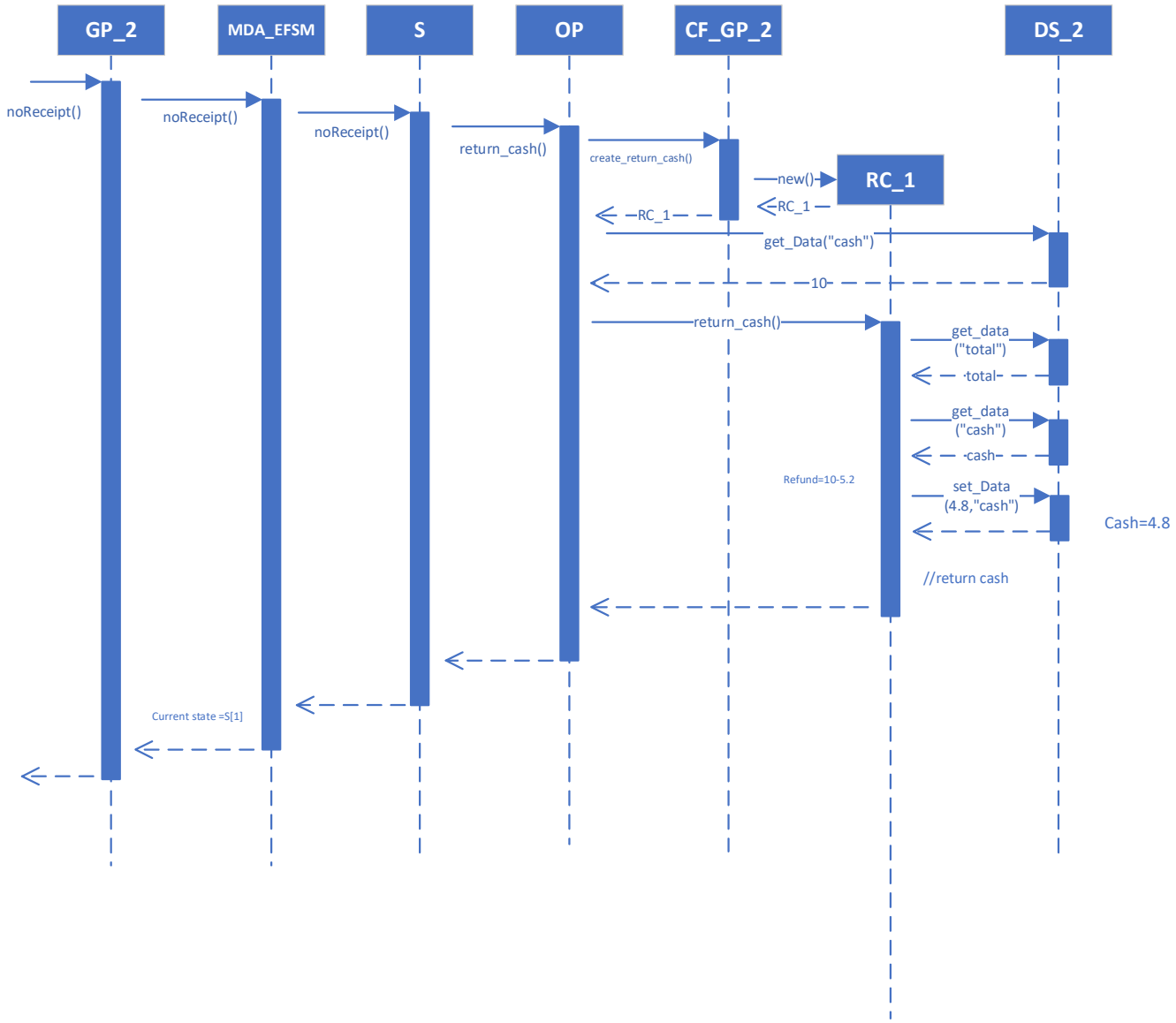

GP_2
MDA_EFSM
DS_2
S0
OP
CF_GP_2
S1
payCash
(10)
set_Data(10,"temp_cash")
Temp_cash=10
start()
start()
pay_msg()
create_pay_msg()
NULL
initialize_Data()
create_Initialize_data()
new()
ID_2
ID_2
ID_2
initialize_data(d)
set_Data(0.0,"price")
Price=0.0
set_Data(0.0,"cash")
Cash=0.0
Current_State=S[2]
payType(2)
M=0;
payType(2)
payType(2)
create_Store_Cash()
new()
SC_1
SC_1
store_cash()
get_Data("temp_cash")
10
Tcash=10
set_Data(tcash,"cash")
display_menu()
create_display_menu()
new()
DM_2
DM_2
DM_2
display_menu()
current_state=S[4]
payCash(10)
->next page for more


GP_2
MDA_EFSM
S3
OP
CF_GP_1
DS_2
Set_P1
premium
select_gas
(3)
select_gas
(3,0)
set_price
(3,0)
create_set_price()
new()
Set_P1
Set_P1
Set_P1
set_price
(3,0,d)
get_data
("Pprice")
-5.2
Gprice=(5.2)
set_Data
(gprice,"price")
Price=5.2
continue()
continue()
Current state=S[5]
Premium()
->next page for more


GP_2
MDA_EFSM
S4
OP
DS_2
CF_GP_2
startPump()
->next page for more
startPump()
start_Pump()
start_pump()
set_init_values
create_set_init_values()
new()
SI_2
SI_2
SI_2
set_init_values()
set_Data(0,"L")
L=0
set_Data(0.0,"total")
Total=0.0
ready_msg()
create_ready_msg()
new()
Ready_
M1
Ready_M1
Ready_M1
ready_msg()
Current_state=S[6]


GP_2
DS_2
MDA_EFSM
S5
OP
CF_GP_2
PG_2
GPM_2
pumpLiter()
get_Data ("cash")
10
get_Data ("price")
5.2
get_Data("L")
1
pump()
pump()
pump_gas_unit()
create_pump_gas_unit()
new()
PG_2
PG_2
get_data("L")
0
get_data("price")
5.2
set_data(5.2,"total")
Liters=0+1
Total=5.2 *1
gas_pump_msg()
create_gas_pump_msg()
new()
GPM_2
GPM_2
gas_pump_msg()
get_Data("L")
1
get_Data("total")
5.2
//display gas pump msg
pumpLiter()
->next page for more


GP_2
DS_2
MDA_EFSM
S5
OP
CF_GP_2
SM_2
pumpLiter()
get_Data ("cash")
10
get_Data ("price")
5.2
get_Data("L")
stop_Pump()
stop_pump()
stop_msg()
create_stop_msg()
new()
SM_2
stop_msg()
//display stop msg
Current_state=s[7]
pumpLiter()
->next page for more


GP_2
MDA_EFSM
S
OP
CF_GP_2
RC_1
DS_2
noReceipt()
noReceipt()
noReceipt()
return_cash()
create_return_cash()
new()
RC_1
RC_1
get_Data("cash")
10
return_cash()
get_data ("total")
total
get_data ("cash")
cash
Refund=10-5.2
set_Data (4.8,"cash")
Cash=4.8
//return cash
Current state =S[1]
noReceipt()
END

5. Program Instructions and Source Code:

From the Command prompt: From the folder where you have the .jar file, type: java -jar Project.jar.

Source code is included in .zip file. To view source code, unzip the file titled “Source.”