Exam #2

Monday, March 25 at 5:00pm
Closed books and notes.
The coverage is posted
sample exam is posted

Project description
is posted
Part #1: is posted

COVERAGE FOR EXAM #2

CS 586; Spring 2024

Exam #2 will be held on **Monday, March 25, 2024,** at **5:00 p.m.**
Location: 104 Stuart Building
The exam is a **CLOSED books and notes** exam.

Coverage for the exam:

- OO design patterns: proxy, adapter, strategy, and abstract factory patterns. [Textbook: Section 3.4 (pp. 263-275); Handout #1, class notes]
- Interactive systems. Model-View-Controller architectural pattern [Textbook: Section 2.4, pp. 123-143]
- Client-Server Architecture
    - Client-Dispatcher-Server [Textbook: Section 3.6: pp. 323-337]
    - Client-Broker-Server Architecture [Textbook: Section 2.3; pp. 99-122]

Sources:

- Textbook: F. Buschmann, et. al., Pattern-oriented software architecture, vol. I, John Wiley & Sons.
- Class notes
- Handout #1

Project

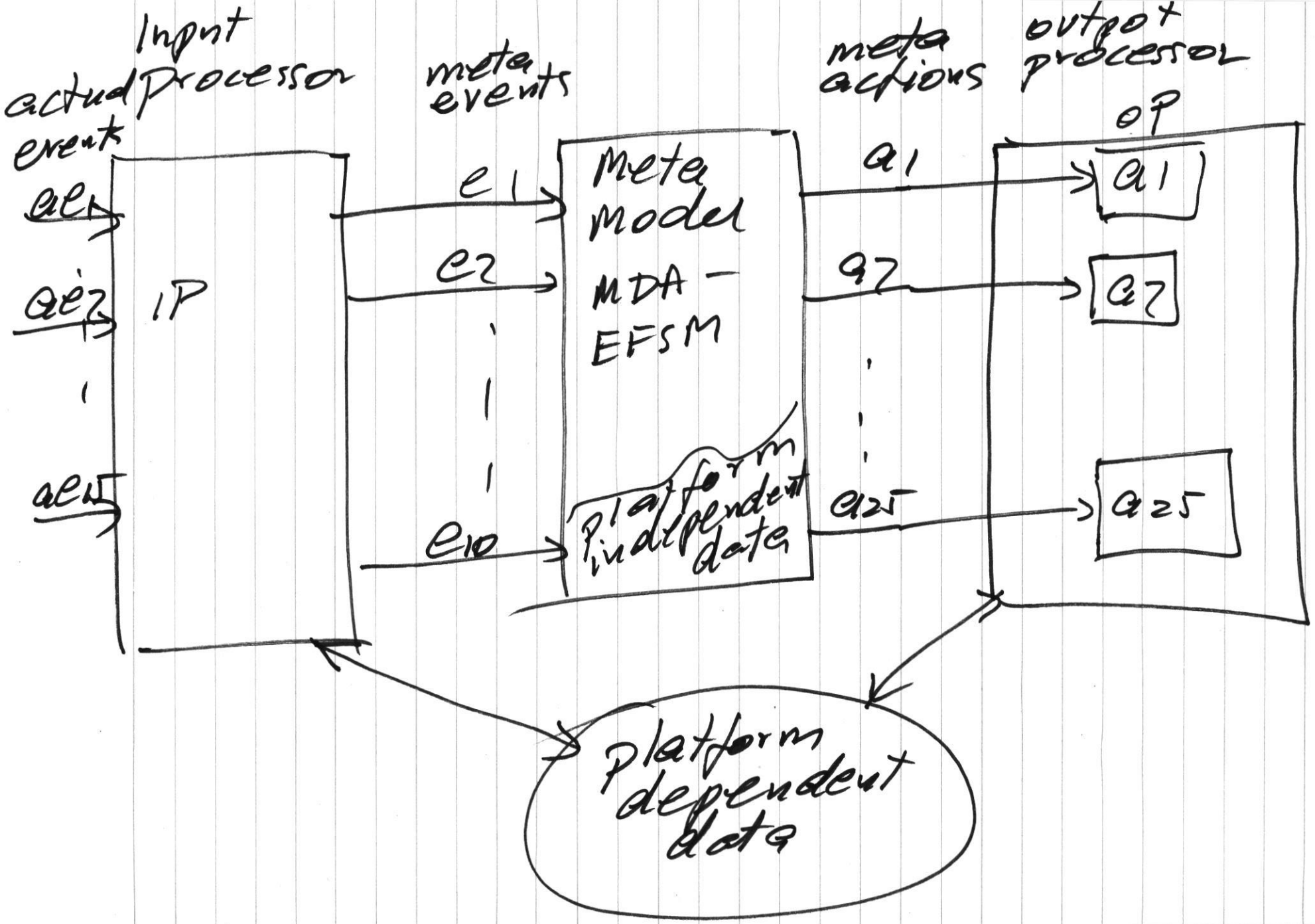
Model driven architecture MDA

seperate:

platform independent state behavior of a "family" of components

FROM

platform specific behavior/aspects of the family of components.


Input Processor
actual event
meta events
meta actions
output processor
ae1
ae2
ae25
IP
e1
e2
e10
Meta Model
MDA - EFSM
Platform independent data
a1
a2
a25
OP
a1
a2
a25
Platform dependent data

Meta State Behavior

MDA - EFSM

EFSM with restrictions.

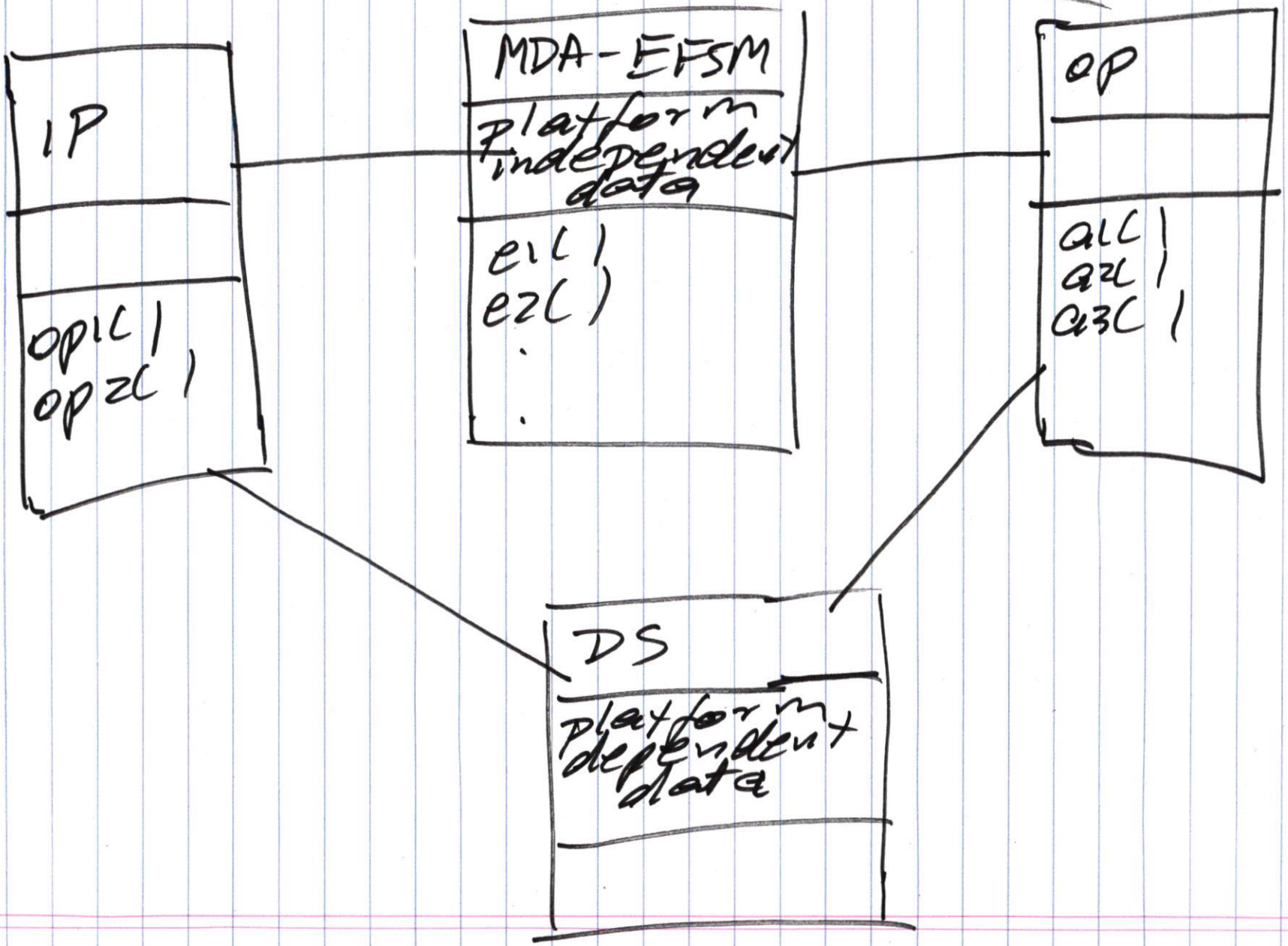
(1) events and actions must be platform independent

(2) no data

or

platform independent data.

MDA Architecture



Responsibilities

IP - operations

(1) check conditions

(2) store parameters in temp area in data store to be used (potentially) by actions of OP

(3) invoke meta event(s) on MDA-EFSM

DO NOT update data in data store!!!

IP operations do not perform any actions!!

MDA-EFSM

(1) accept meta events

(2) execute MDA-EFSM model

(3) issue/invoke actions on OP for execution.

(4) can have access to its own platform independent data.

(5) does not have access to data store!!

OP

* execute actions
  invoked by MDA-EFSM
* manipulates (read/write)
  data in data store DS.

Project

Use MDA architecture to design and implement a "family" of Gas Pump components.

Two Gas Pumps.

GP-1 GP-2.

State behavior is specified by EFSMs.

GP-1 Figure 1.

GP-2 Figure 2.

Differences

1. operation names/signature.
2. types of payment.
3. types of gas disposed.
4. disposed units of gas.
5. display menus
6. messages
7. . .

PROJECT-General Description
PART #1

CS 586; Spring 2024

Deadlines:

Part #1: MDA-EFSM (5 points): **Monday, April 1, 2024**

Late submissions: 50% off

After **April 4** the MDA-EFSM will not be accepted.

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

Submission: The MDA-EFSM assignment must be submitted on Blackboard. Your submission should be as a **single PDF-file** (otherwise, a 10% penalty will be applied). The hardcopy submissions will not be accepted.

The detailed description of Part #2 of the project will be posted later.

Goal:

The goal of this project is to design two different gas pump components using the Model-Driven Architecture (MDA) and then implement these gas pump components based on this design.

Description of the Project:

There are two gas pump components: *GP-1* and *GP-2*.

The gas pump **GP-1** component supports the following operations:

Activate (int a)	// the gas pump is activated where *a* is the price of the gas per liter
Start()	//start the transaction
PayCredit()	// pay for gas by a credit card
Reject()	// credit card is rejected
Cancel()	// cancel the transaction
Approved()	// credit card is approved
PayCash(int c)	// pay for gas by cash, where *c* represents prepaid cash
StartPump()	// start pumping gas
Pump()	// one liter of gas is disposed
StopPump()	// stop pumping gas

The **GasPump-2** component supports the following operations:

Activate (float a, float b, float c)	// the gas pump is activated where *a* is the price of Regular gas, *b* is //the price of Premium gas and *c* is the price of Diesel per Gallon
Start()	//start the transaction
PayCash(int c)	// pay for gas by cash, where *c* represents prepaid cash
Cancel()	// cancel the transaction
Premium()	// Premium gas is selected
Regular()	// Regular gas is selected
Diesel()	// Diesel gas is selected
StartPump()	// start pumping gas
PumpGallon()	// one Gallon of gas is disposed
Stop()	// stop pumping gas
Receipt()	// Receipt is requested
NoReceipt()	// No receipt

Both gas pumps are state-based components and are used to control simple gas pumps. Users can pay by cash or a credit card. The gas pump may dispose of different types of gasoline. The price of the gasoline is provided when the gas pump is activated. The detailed behavior of gas pump components is specified using EFSM. The EFSM of Figure 1 shows the detailed behavior of gas pump *GP-1* and the EFSM of Figure 2 shows the detailed behavior of gas pump *GP-2*. Notice that there are several differences between gas pump components.

Aspects that vary between two gas pump components:

a. Types of gasoline disposed
b. Types of payment
c. Display menu(s)
d. Messages
e. Receipts
f. Operation names and signatures
g. Data types
h. etc.

The goal of this project is to design two GP components using the Model-Driven Architecture (MDA) covered in the course. An executable meta-model referred to as MDA-EFSM of GP components should capture the "generic behavior" of two GP components and should be de-coupled from data and implementation details. Notice that in your design there should be **ONLY** one MDA-EFSM for two GP components. The meta-model (MDA-EFSM) used in the Model-Driven architecture should be expressed as an EFSM (Extended Finite State Machine) model. Notice that the EFSMs shown in Figure 1 and Figure 2 are **not acceptable** as a meta-model (MDA-EFSM) for this model-driven architecture.

MDA-ESM REPORT SUBMISSION

The MDA-EFSM model report for the *GP* components should contain:

- A class diagram
- A list of meta events for the MDA-EFSM
- A list of meta actions for the MDA-EFSM, where the responsibility of each action must be described
- A state diagram/model of the MDA-EFSM
- Pseudo-code of all operations of Input Processors of *GP-1* and *GP-2*

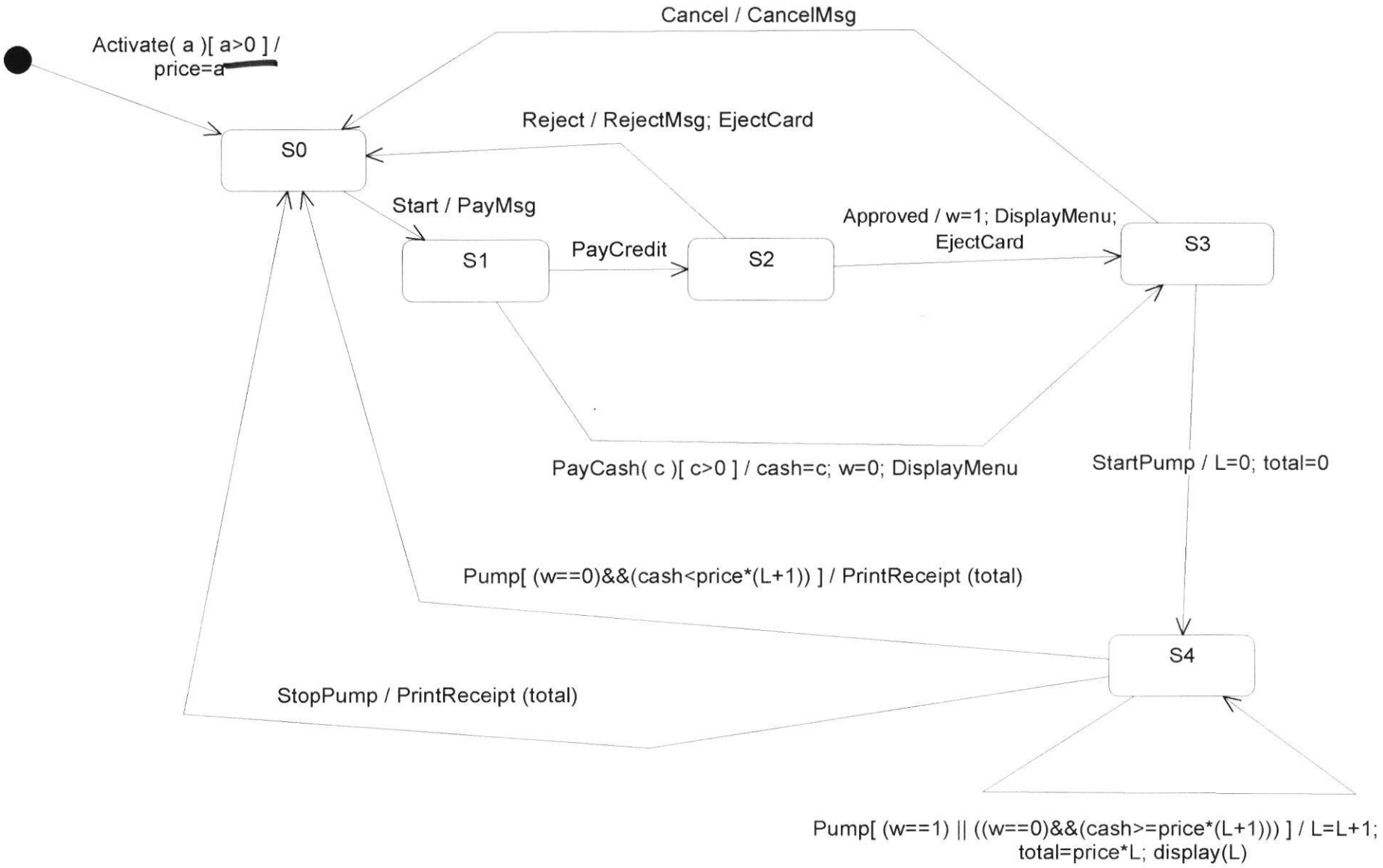


Figure 1: EFSM for gas pump GP-1

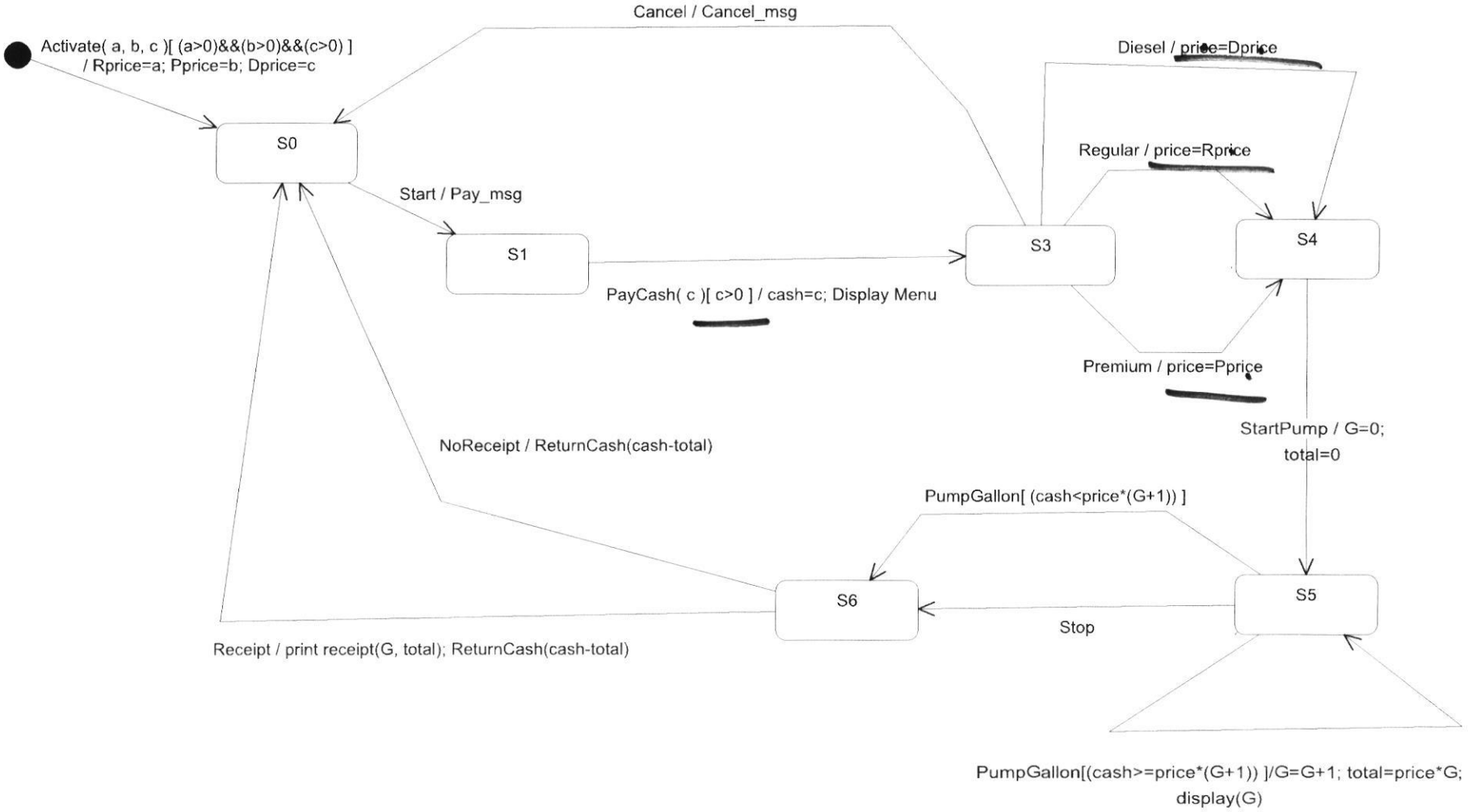


Figure 2: EFSM for GasPump-2

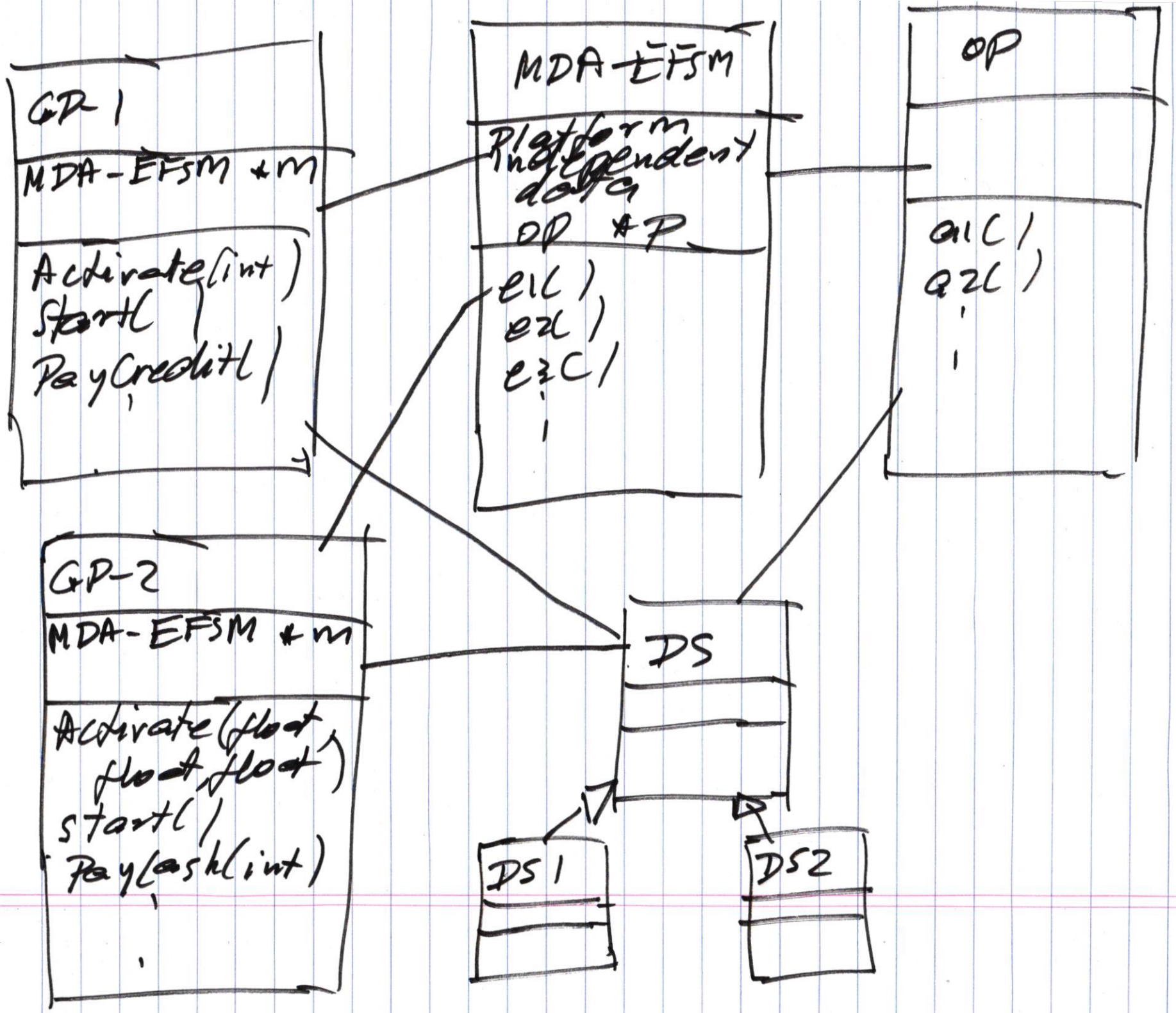
First Part

create MDA-EFSM that captures "meta/generic" behavior of both Gas Pumps.

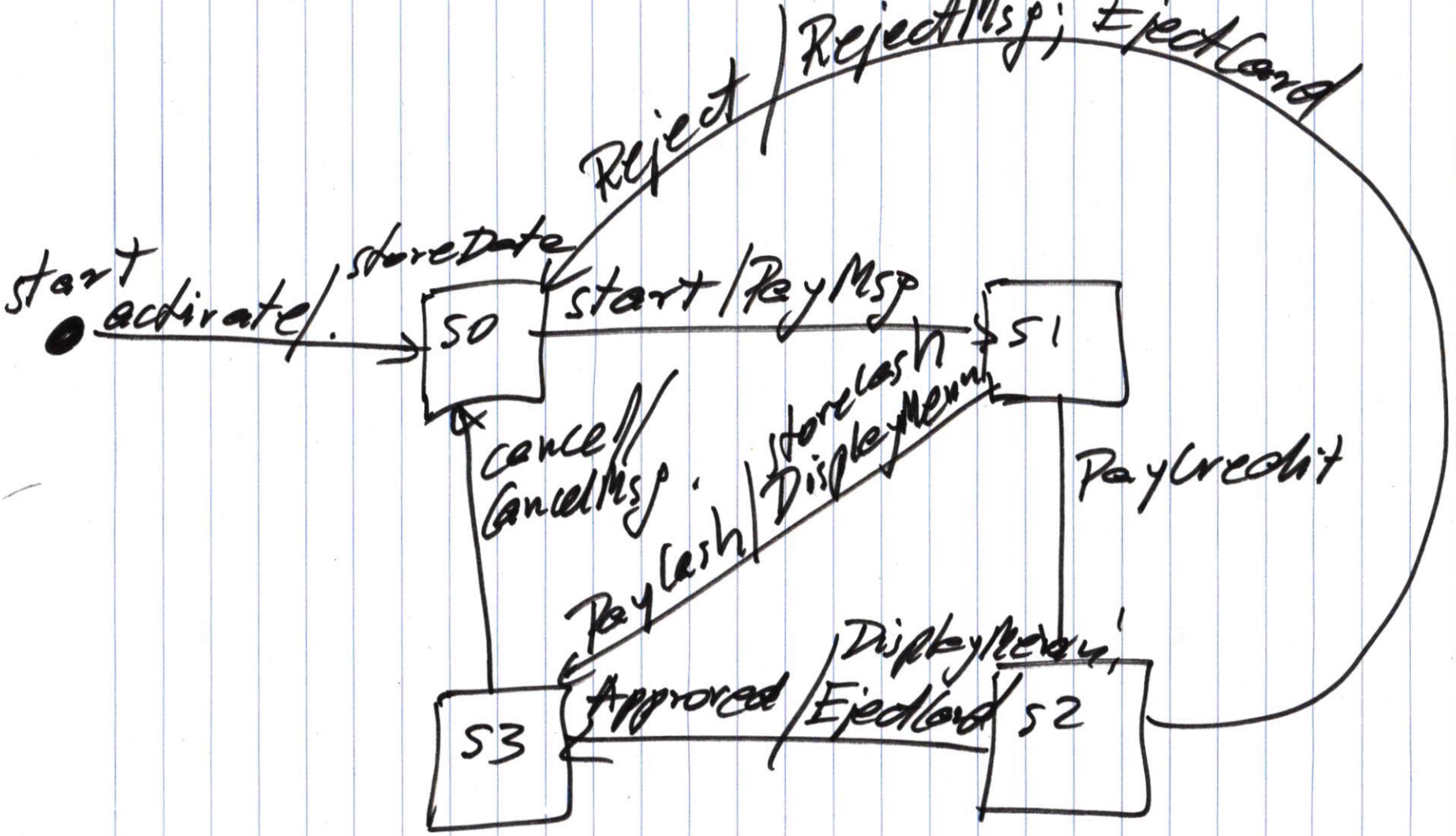
Deadline: April 1

---

1. Identify meta events,
2. Identify meta actions.
   (provide description of each meta action)
3. State Diagram MDA-EFSM
   - states
   - transitions.
4. Pseudo code for all operations of GP-1 and GP-2
5. Data in DS-1 and DS2


GP-1
MDA-EFSM *m
Activate(int)
Start()
PayCredit()
MDA-EFSM
Platform Independent data
OP *P
e1()
e2()
e3()
OP
q1()
q2()
GP-2
MDA-EFSM *m
Activate(float, float, float)
Start()
PayCash(int)
DS
DS1
DS2

meta events	meta actions
activate()	StoreData()
start()	PayMsg()
PayCredit()	RejectMsg()
Approved()	EjectCard()
Reject()	DisplayMenu()
PayCash()	StoreCash()
Cancel()	CancelMsg()
SelectGas(int)	StorePrice(int)


start
activate/
storeData
S0
start/KeyMsg
S1
Reject/RejectMsg; EjectCard
cancel/
CancelMsg
PayCash/
storeCash
DisplayMenu
PayCredit
S3
Approved/
EjectCard
DisplayMenu
S2

d → DS 1

GP-1

```
Activate(int a){
  if(a>0){
    d->temp_a = a;
    m->activate();
  }
}
```

```
start()
{ m -> start()
}
```

```
PayCredit()
{ m -> ~~Credi~~PayCredit()
}
```

d → DS-2

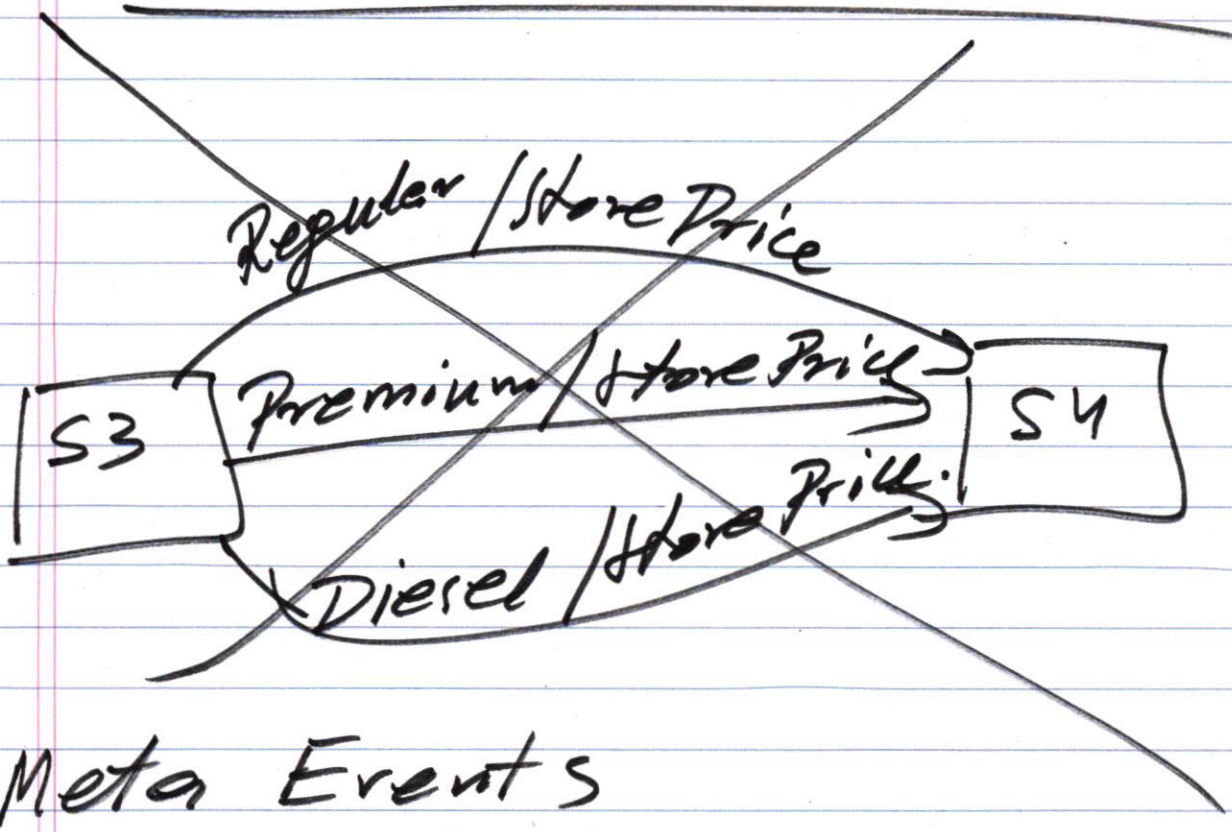
GP-2

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

```
start()
{ m -> start()
}
```

```
PayCash(int c)
{ if(c>0){
  d->tempcc = c;
  m->PayCash();
}
}
```

Selecting Gas Type



Meta Events

Regular
Premium
Diesel
} platforms dependent.

Incorrect solutions!!

Correct

~~Better~~ Solution

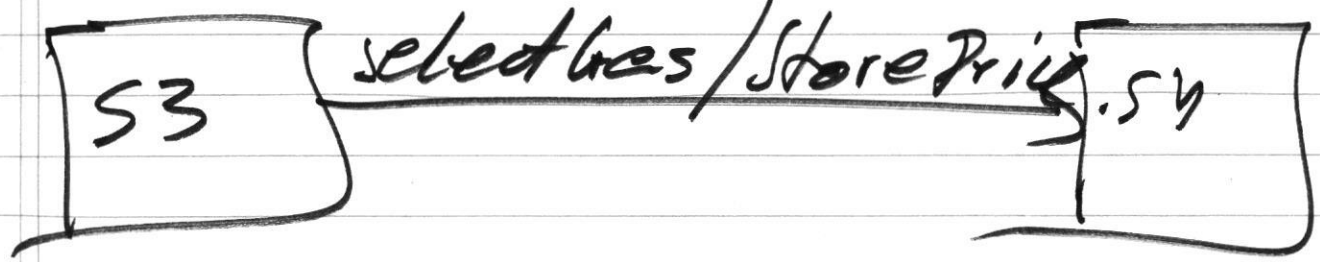
version #1

meta event

selectGas()

meta Action

storePrice()

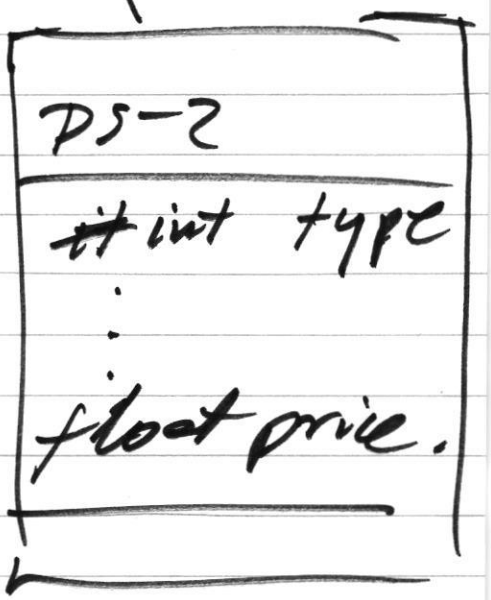


1: Regular
2: Diesel
3: Premium

IP
GP-2

Regular()
{
d->type = 1
m->selectGas()
}

Diesel()
{
d->type = 2
m->selectGas()
}



version #2

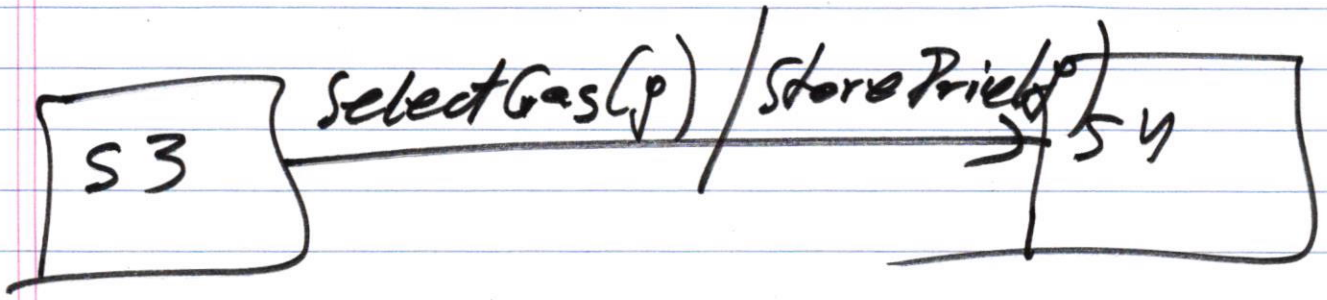
meta event

selectGas(int g)

meta action

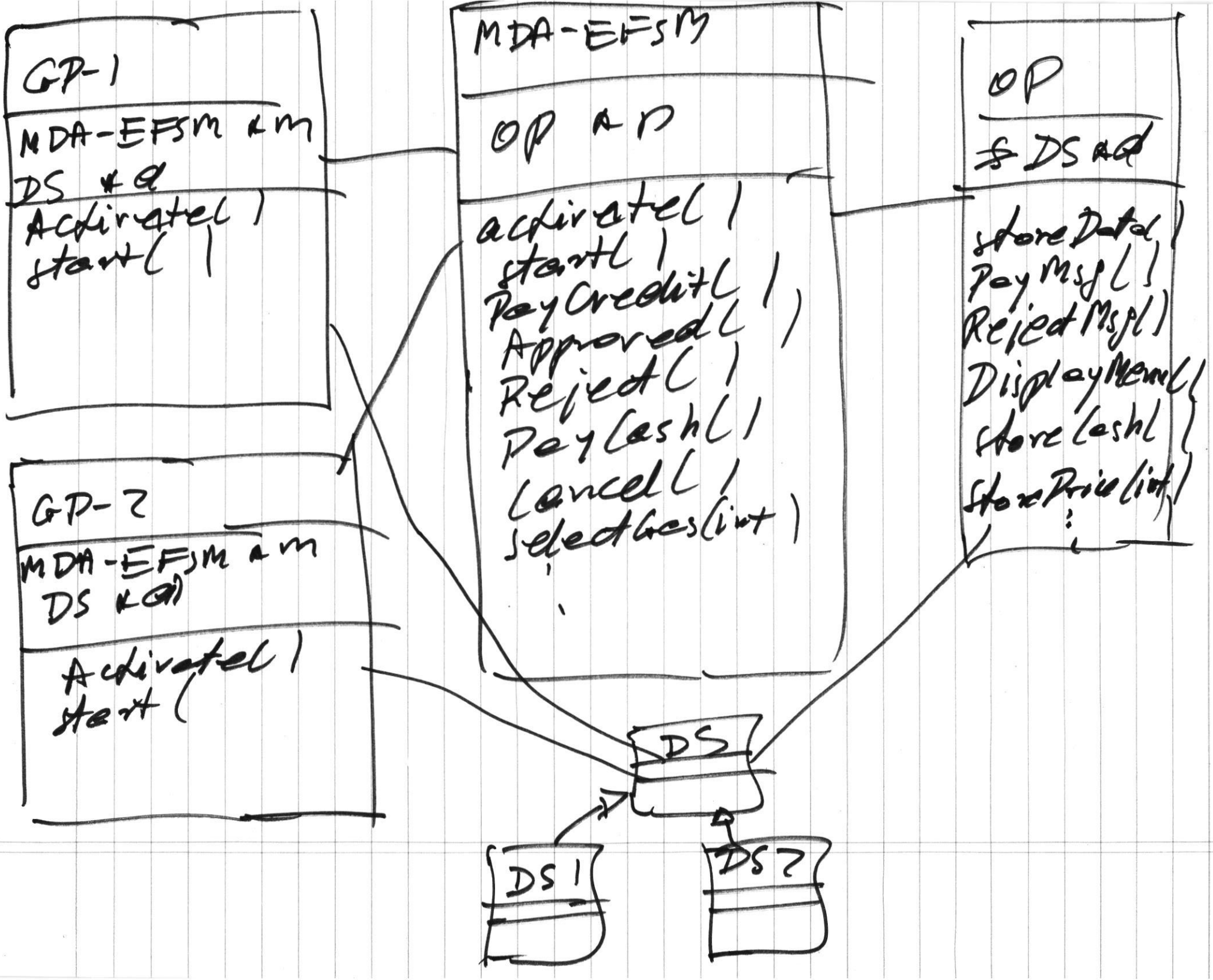
storePrice(int g)

g = 1 Regular
g = 2 Diesel
g = 3 Premium



```
Regular()
{
  m -> selectGas(1)
}

Diesel()
{
  m -> selectGas(2)
}
```


GP-1
MDA-EFSM *m
DS *d
Activate( )
start( )
GP-2
MDA-EFSM *m
DS *d
Activate( )
start( )
MDA-EFSM
OP *p
activate( )
start( )
PayCredit( )
Approved( )
Reject( )
PayCash( )
Cancel( )
SelectGas(int)
OP
$DS *d
storeData( )
PayMsg( )
RejectMsg( )
DisplayMenu( )
storeCash( )
storePrice(int)
DS
DS1
DS2

Sample MDA-EFSM for ATM components

There are two ATM components: ATM-1 and ATM-2.

The **ATM-1** component supports the following operations:

```
create()                    // ATM is created
card (int x, string y)      // ATM card is inserted where x is a balance and y is a pin #
pin (string x)              // provides pin #
deposit (int d);            // deposit amount d
withdraw (int w);           // withdraw amount w
balance ();                 // display the current balance
lock(string x)              // lock the ATM, where x is a pin #
unlock(string x)            // unlock the ATM, where x is pin #
exit()                      // exit from the ATM
```

The **ATM-2** component supports the following operations:

```
create()                    // ATM is created
CARD (float x, int y)       // ATM card is inserted where x is a balance and y is a pin #
PIN (int x)                 // provides pin #
DEPOSIT (float d);          // deposit amount d
WITHDRAW (float w);         // withdraw amount w
BALANCE ();                 // display the current balance
EXIT()                      // exit from the ATM
```

These ATM components are state-based components and support three types of transactions: withdrawal, deposit, and balance inquiry. Before any transaction can be performed, operation *card*(x, y) (or *CARD*(x, y)) must be issued, where x is an initial balance in the account and y is a pin used to get permission to perform transactions. Before any transaction can be performed, operation *pin(x)* (or *PIN(x)*) must be issued. The *pin(x)* (or *PIN(x)*) operation must contain the valid pin # that must be the same as the pin # provided in *card*(x, y) (or *CARD*(x, y)) operation. There is a limit on the number of attempts with an invalid pin. The account can be overdrawn (below minimum balance), but a penalty may apply. If the balance is below the minimum balance then the withdrawal transaction cannot be performed. In addition, ATM-1 component can be locked by issuing *lock(x)* operation, where x is a pin #. The ATM-1 can be unlocked by *unlock(x)* operation. The detailed behavior of ATM components is specified using EFSM. The EFSM of Figure 1 shows the detail behavior of ATM-1, and the EFSM of Figure 2 shows the detailed behavior of ATM-2. Notice that there are several differences between ATM components.

Aspects that vary between these ATM components:

a. Maximum number of times incorrect pin can be entered
b. Minimum balance
c. Display menu(s)
d. Messages, e.g., error messages, etc.
e. Penalties
f. Operation names and signatures
g. Data types
h. etc.

The goal is to design an executable meta-model, referred to as **MDA-EFSM**, for all ATM components. The MDA-EFSM should capture the "generic behavior" of these two ATM components and should be de-coupled from data and implementation details. Notice that there should be **ONLY** one MDA-EFSM for these two ATM components.

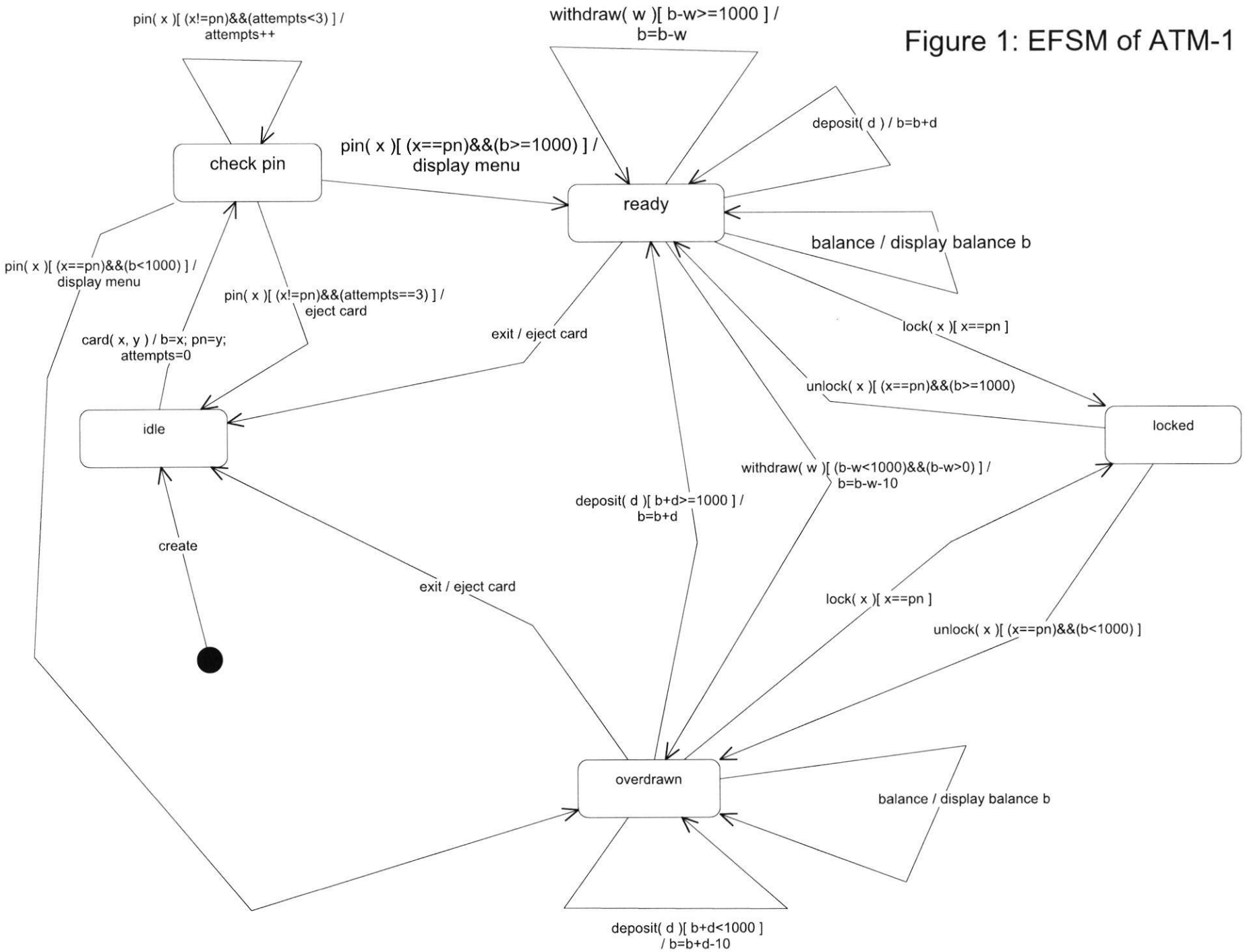


Figure 1: EFSM of ATM-1

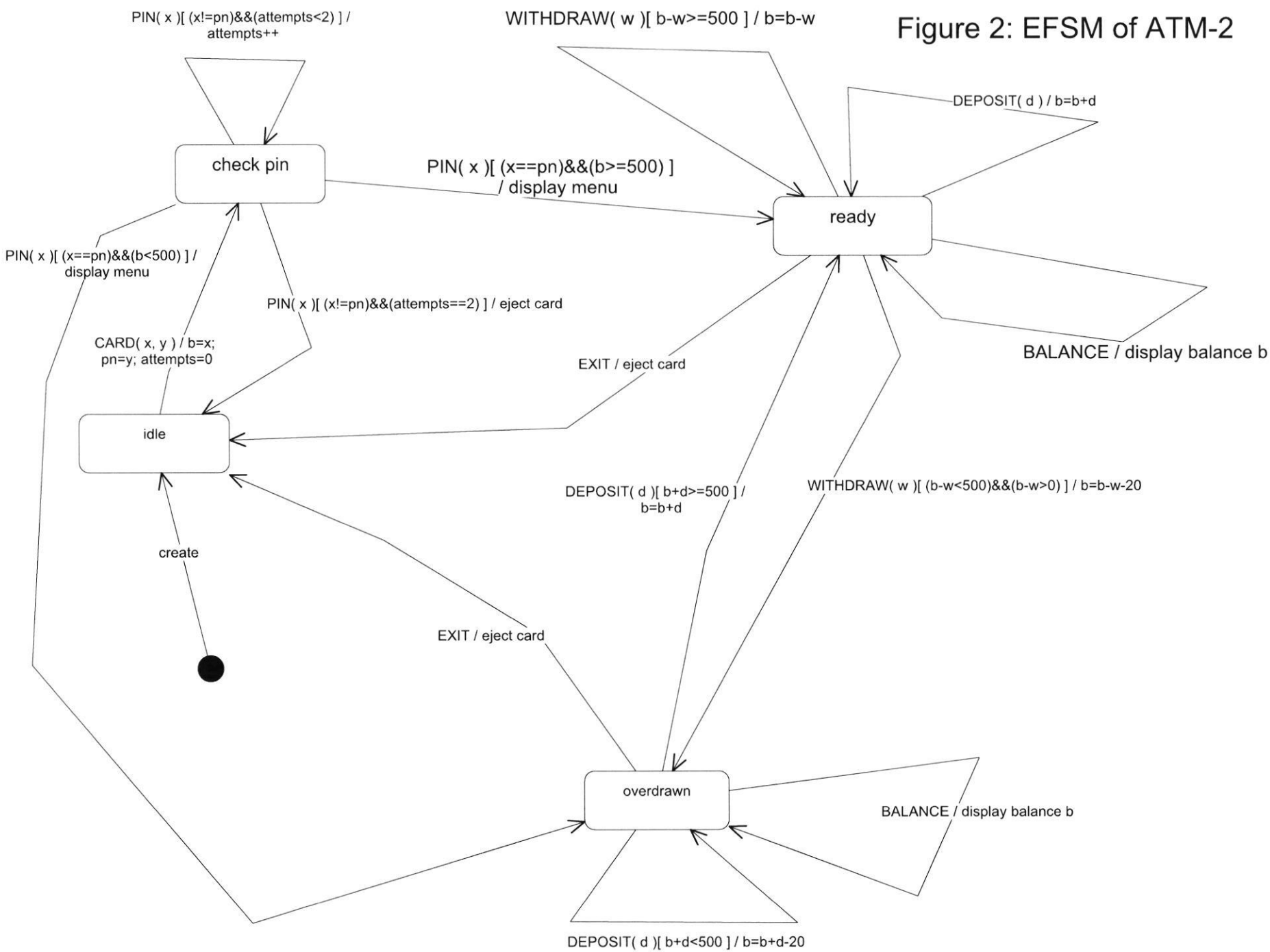
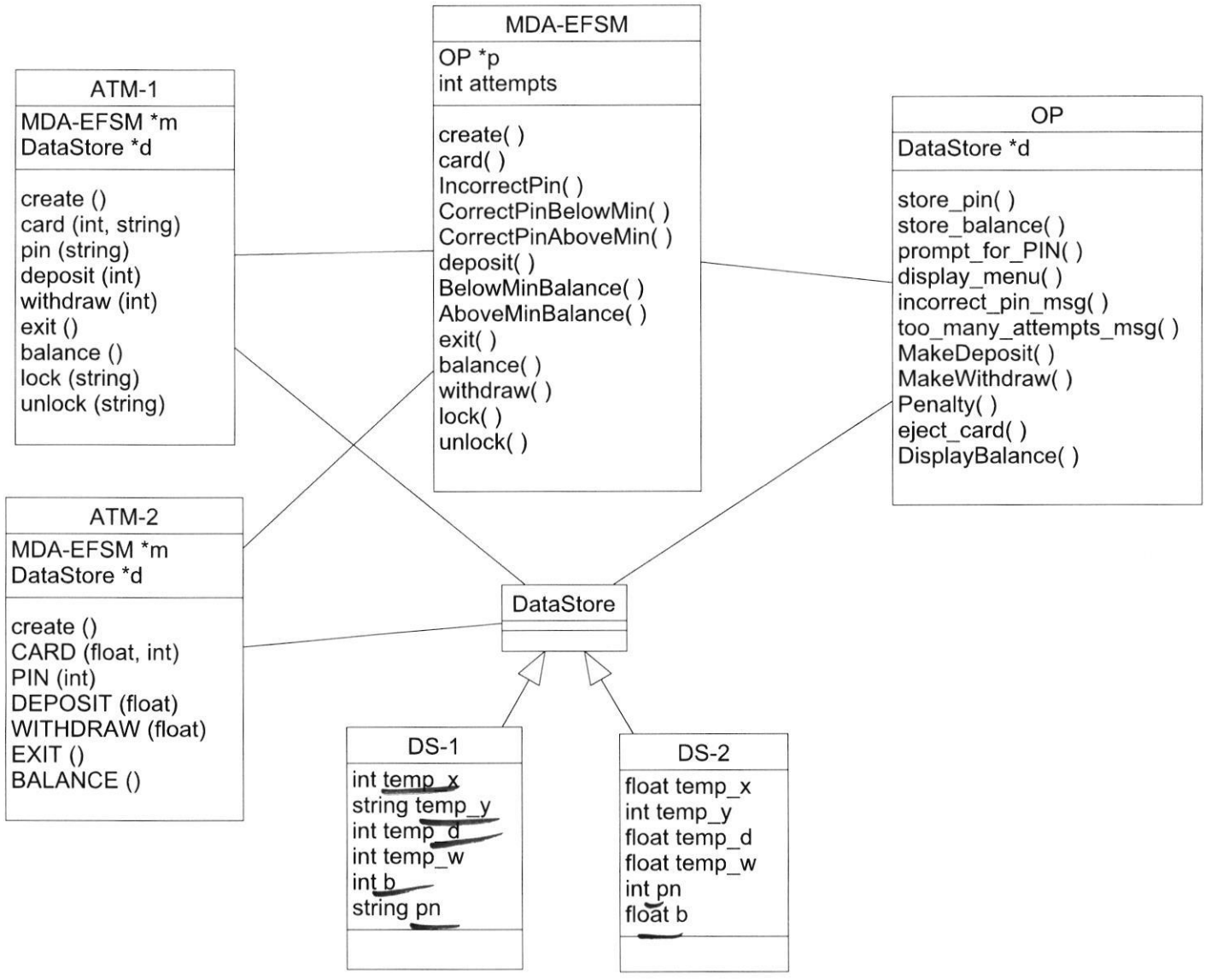


Figure 2: EFSM of ATM-2

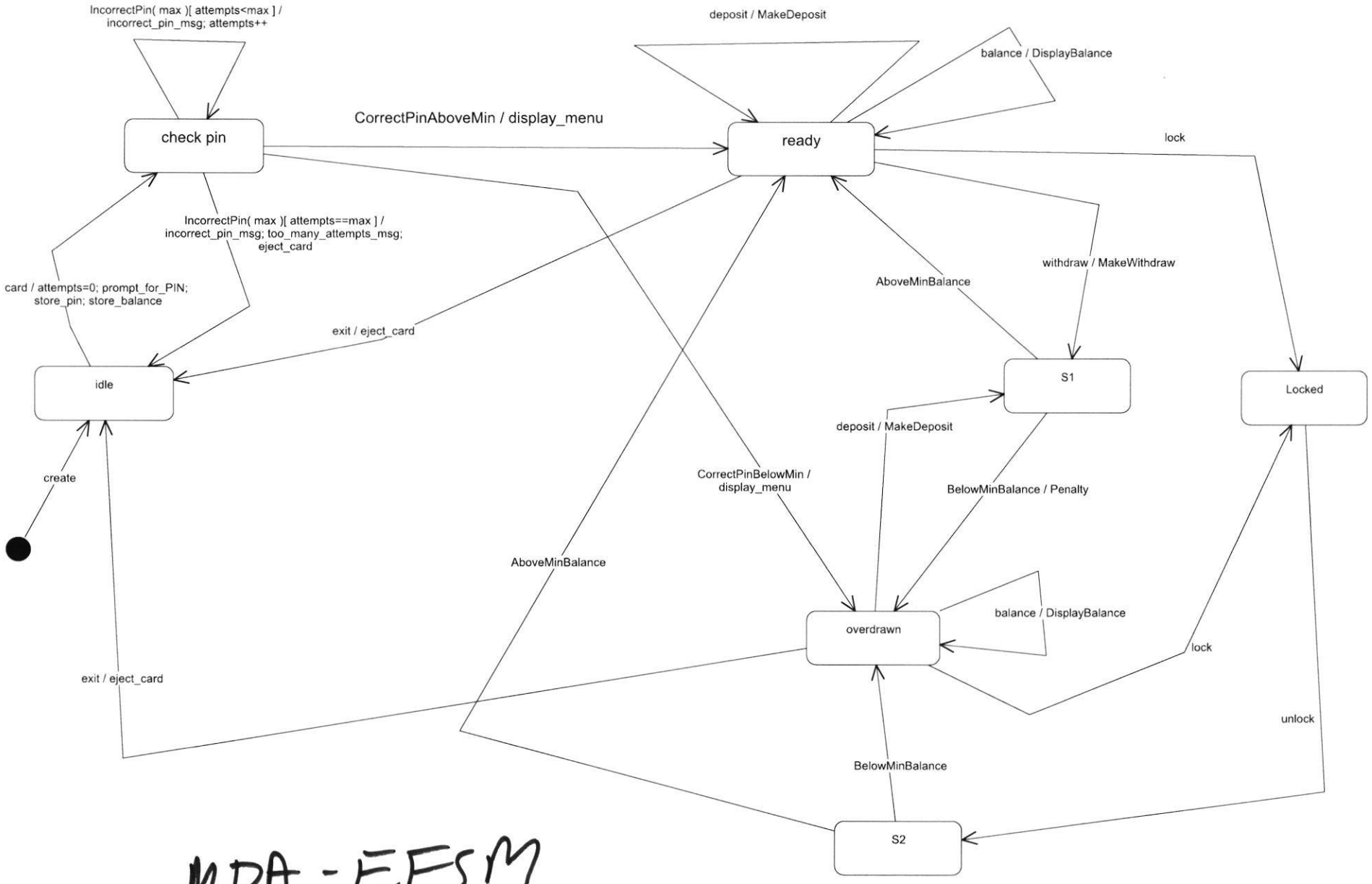

MDA-EFSM
OP *p
int attempts
create( )
card( )
IncorrectPin( )
CorrectPinBelowMin( )
CorrectPinAboveMin( )
deposit( )
BelowMinBalance( )
AboveMinBalance( )
exit( )
balance( )
withdraw( )
lock( )
unlock( )
ATM-1
MDA-EFSM *m
DataStore *d
create ()
card (int, string)
pin (string)
deposit (int)
withdraw (int)
exit ()
balance ()
lock (string)
unlock (string)
OP
DataStore *d
store_pin( )
store_balance( )
prompt_for_PIN( )
display_menu( )
incorrect_pin_msg( )
too_many_attempts_msg( )
MakeDeposit( )
MakeWithdraw( )
Penalty( )
eject_card( )
DisplayBalance( )
ATM-2
MDA-EFSM *m
DataStore *d
create ()
CARD (float, int)
PIN (int)
DEPOSIT (float)
WITHDRAW (float)
EXIT ()
BALANCE ()
DataStore
DS-1
int temp_x
string temp_y
int temp_d
int temp_w
int b
string pn
DS-2
float temp_x
int temp_y
float temp_d
float temp_w
int pn
float b

MDA-EFSM Events:

create()
card()
IncorectPin(int max)
CorrectPinBelowMin()
CorrectPinAboveMin()
deposit()
BelowMinBalance()
AboveMinBalance()
exit()
balance()
withdraw()
lock()
unlock

MDA-EFSM Actions:

store_pin	// stores pin from temporary data store to *pin* in data store
store_balance	// stores pin from temporary data store to *b* in data store
prompt_for_PIN	// prompts to enter pin
display_menu	// display a menu with a list of transactions
incorrect_pin_msg	// displays incorrect pin message
too_many_attempts_msg	// display too many attempts message
MakeDeposit	// makes deposit (increases balance by a value stored in temp. data store)
MakeWithdraw	// makes withdraw (decreases balance by a value stored in temp. data store)
Penalty	// applies penalty (decreases balance by the amount of penalty)
eject_card	// ejects the card
DisplayBalance	// displays the current value of the balance


IncorrectPin(max)[attempts<max] /
incorrect_pin_msg; attempts++
deposit / MakeDeposit
balance / DisplayBalance
CorrectPinAboveMin / display_menu
check pin
ready
lock
IncorrectPin(max)[attempts==max] /
incorrect_pin_msg; too_many_attempts_msg;
eject_card
withdraw / MakeWithdraw
AboveMinBalance
card / attempts=0; prompt_for_PIN;
store_pin; store_balance
exit / eject_card
idle
S1
Locked
deposit / MakeDeposit
create
CorrectPinBelowMin /
display_menu
BelowMinBalance / Penalty
AboveMinBalance
balance / DisplayBalance
overdrawn
lock
exit / eject_card
unlock
BelowMinBalance
S2

MDA - EFSM

Operations of the Input Processor
(ATM-1)

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

card (int x, string y) {
        d->temp_x=x;
        d->temp_y=y;
        m->card();
}

deposit (int d) {
        d->temp_d=d;
        m->deposit();
        if (d->b < 1000)
            m->BelowMinBalance();
        else m->AboveMinBalance();
}

withdraw (int w) {
        d->temp_w=w;
        if ((d->b-w) > 0) m->withdraw();
        if (d->b<1000)
            m->BelowMinBalance();
        else m->AboveMinBalance();
}

pin (string x) {
        if (x==d->pn) {
            if (d->b<1000)
                m->CorrectPinBelowMin ();
            else m->CorrectPinAboveMin();
        }
        else m->IncorrectPin(3)
}
```

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

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

lock (string x) {
        if (d->pn==x) m->lock();
}

unlock (string x) {
        if (x==d->pn) {
            m->unlock();
            if (d->b<1000)
                m->BelowMinBalance ()
            else m->AboveMinBalance();
        }
}
```

Notice:
m: pointer to the MDA-EFSM
d: pointer to the data store
In the data store:
b: contains the current balance
pn: contains the correct pin #

Operations of the Input Processor (ATM-2)

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

CARD (float x, int y) {
        d->temp_x=x;
        d->temp_y=y;
        m->card();
}

DEPOSIT (float d) {
        d->temp_d=d;
        m->deposit();
        if (d->b<500)
             m->BelowMinBalance();
        else m->AboveMinBalance();
}

WITHDRAW (float w) {
        d->temp_w=w;
        if ((d->b-w) > 0) m->withdraw();
        if (d->b<500)
             m->BelowMinBalance();
        else m->AboveMinBalance();
}

PIN (int x) {
        if (x==d->pn) {
             if (d->b<500)
                  m->CorrectPinBelowMin ();
             else m->CorrectPinAboveMin();
        }
        else m->IncorrectPin(2)
}

EXIT() {m->exit();}

BALANCE() {m->balance();}
```

Notice:
m: pointer to the MDA-EFSM
d: pointer to the data store
In the data store:
b: contains the current balance
pn: contains the correct pin #