Exam #2

Wednesday, October 29

Closed books and notes

The coverage is posted

A sample exam with solutions is posted

Project

Project description part #1 is posted

COVERAGE FOR EXAM #2

CS 586; Fall 2025

Exam #2 will be held on **Wednesday, October 29, 2025,** at **5:00 p.m.**
Location: 111 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

Model-Driven Architecture

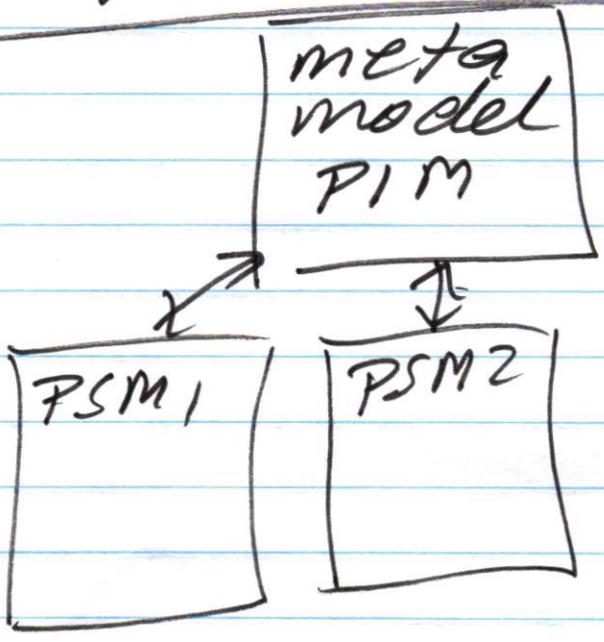
MDA

seperate

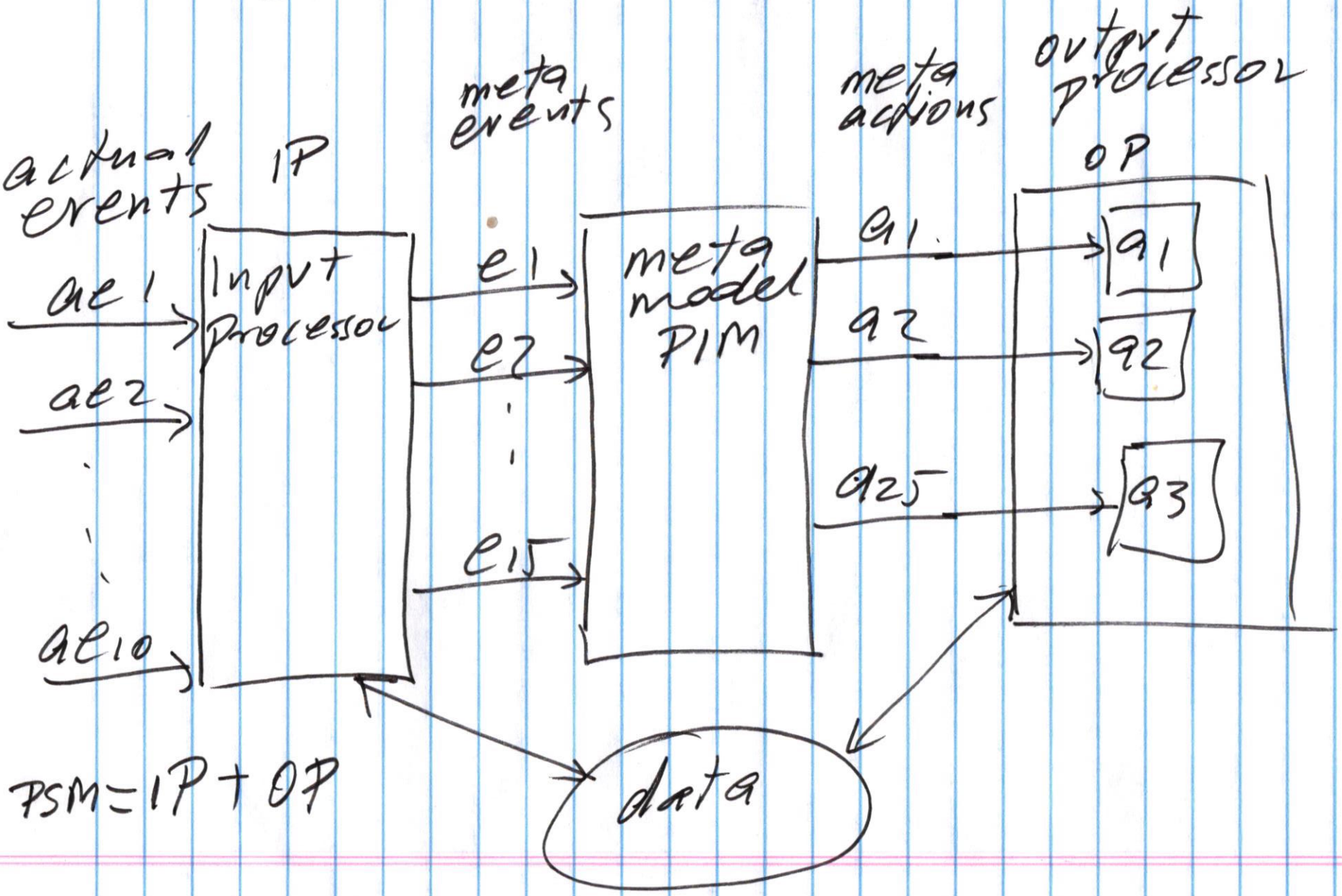
* platform independent behavior of a family of systems

FROM

* platform specific behavior of a given system

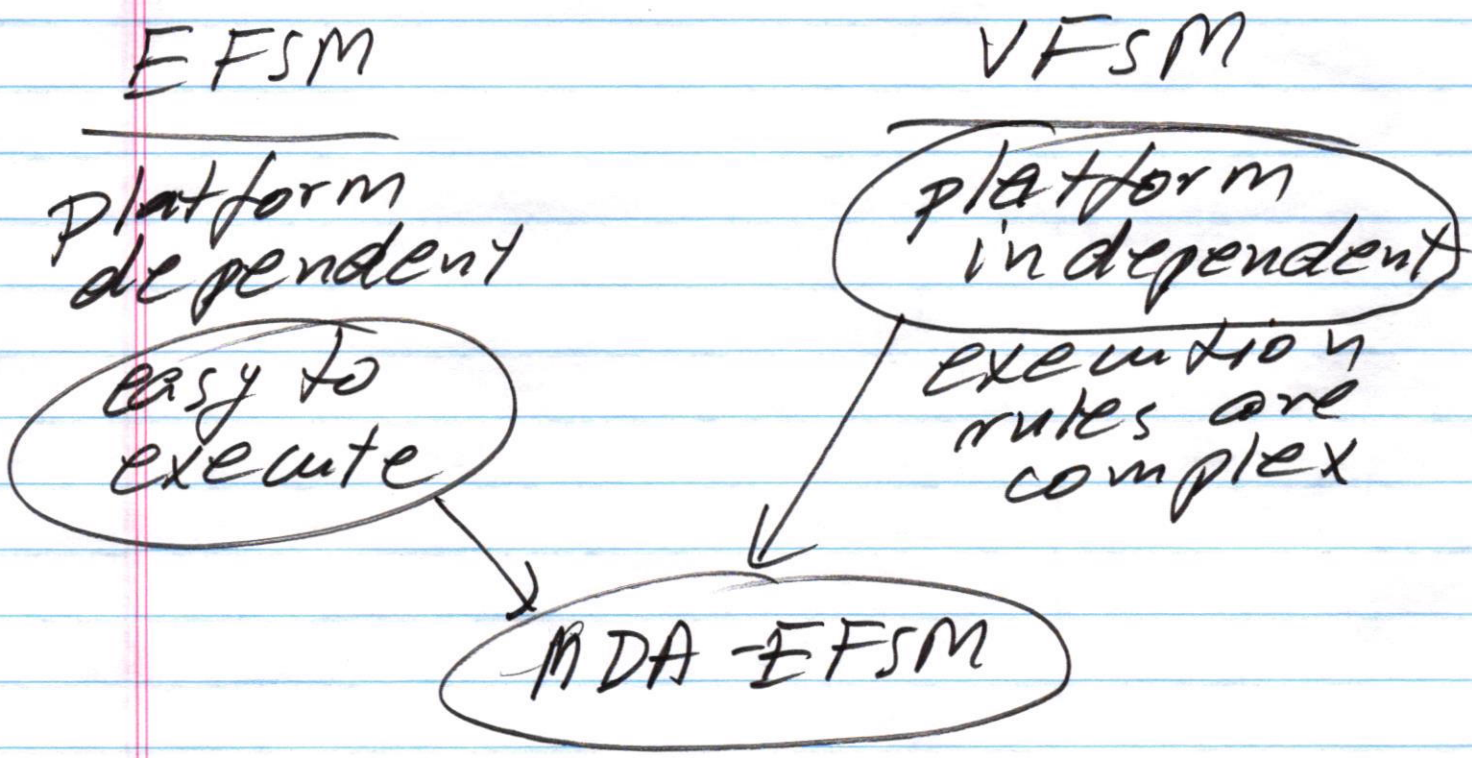


event/state based system



PSM = IP + OP

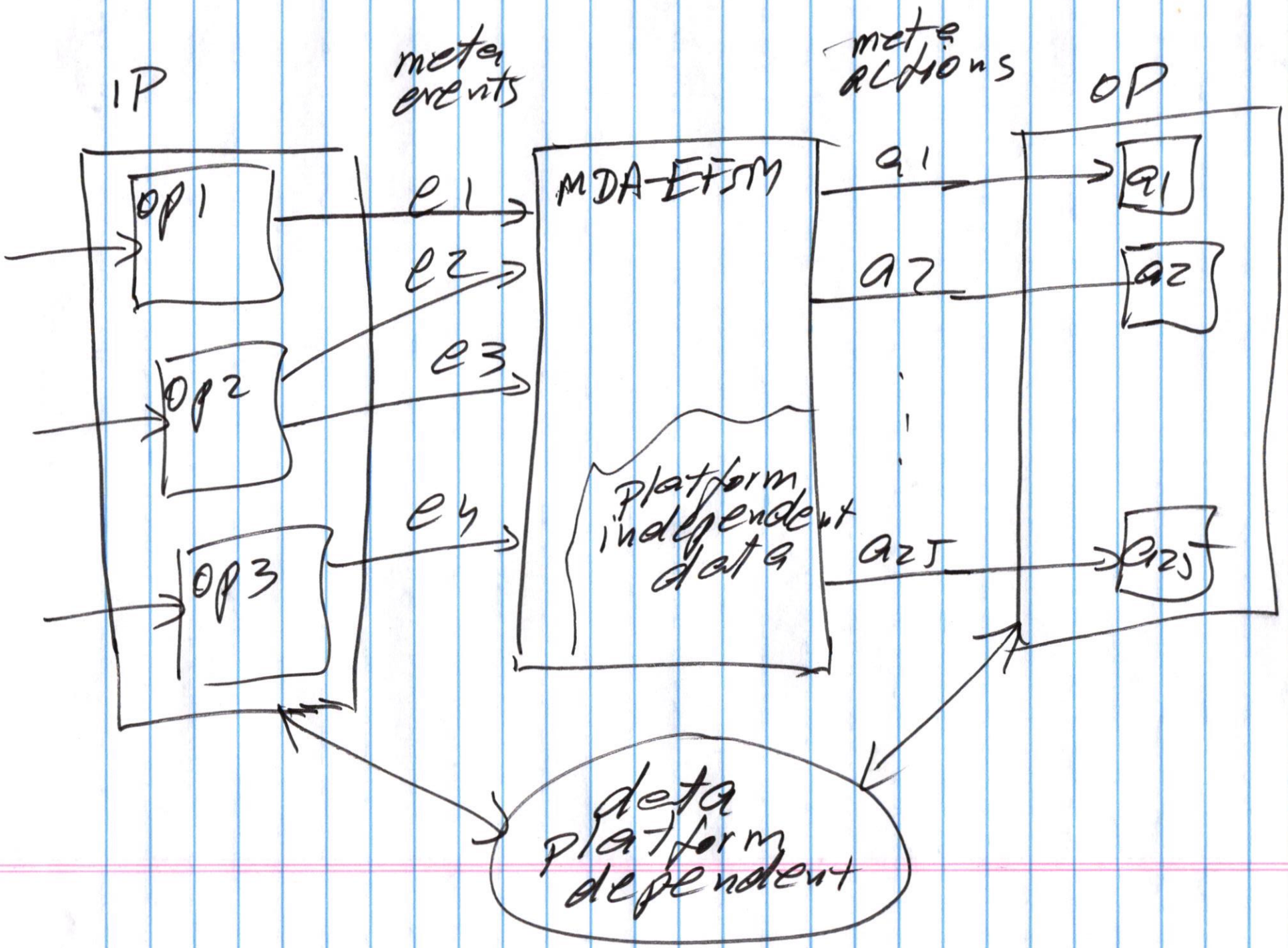
Meta state behavior

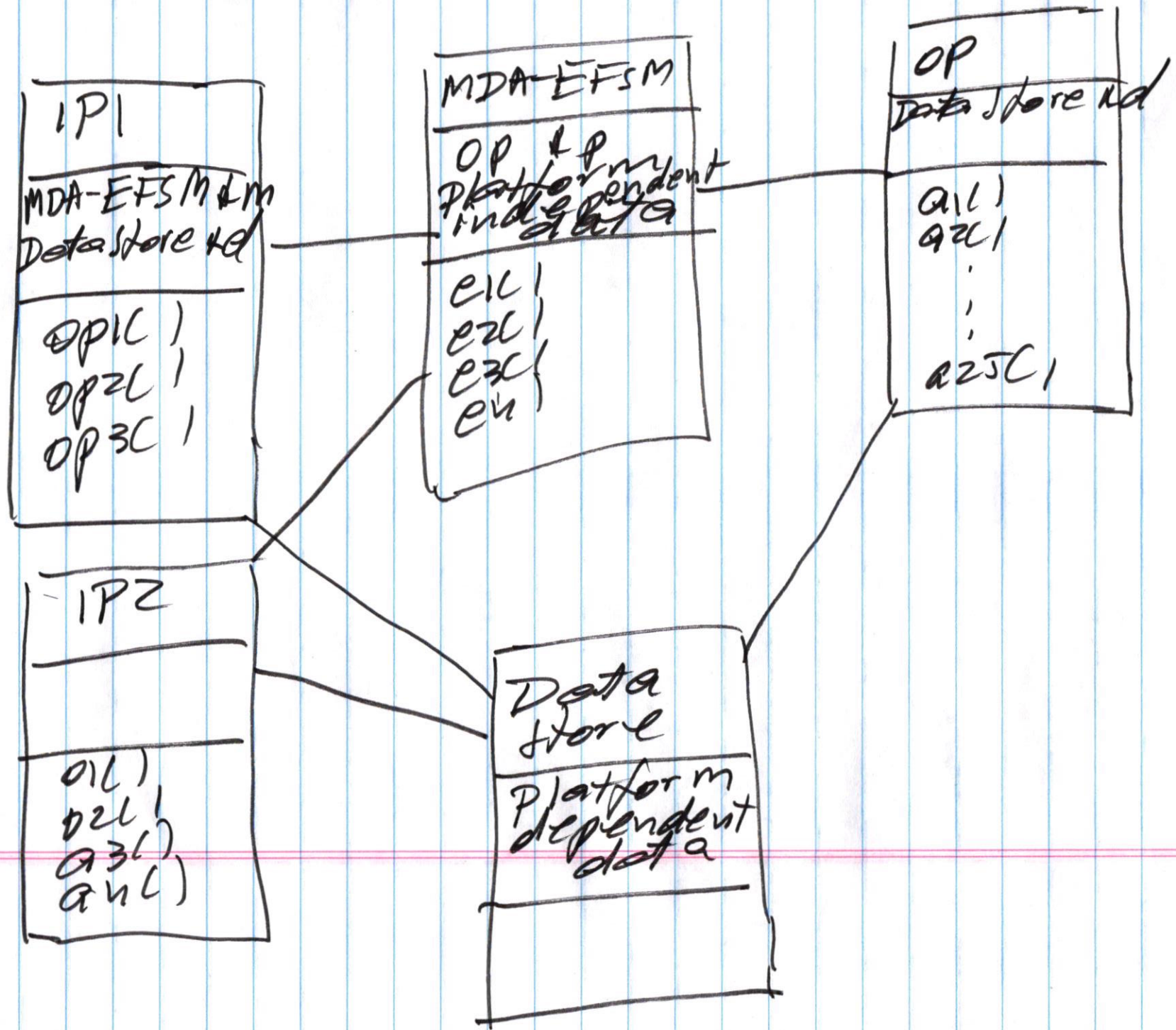


EFSM with restrictions

(a) no data
or
platform independent data

(b) platform independent events and actions


IP
meta events
meta actions
OP
op1
op2
op3
e1
e2
e3
e4
MDA-EFSM
Platform independent data
a1
a2
a2J
a1
a2
a2J
data platform dependent


IP1
MDA-EFSM
Data stored
op1()
op2()
op3()
MDA-EFSM
OP
Platform independent data
e1()
e2()
e3()
en()
OP
Data stored
q1()
q2()
q25()
IP2
o1()
o2()
o3()
on()
Data store
Platform dependent data

Project

Two Gas Pump components

GP-1

GP-2

CS586 PROJECT – General Description

CS 586; Fall 2025

Deadlines:

Part #1: MDA-EFSM (5 points): **Friday, November 7, 2025**

Late submissions: 50% off

After **November 11,** 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 Canvas. Your submission should be 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 using an object-oriented programming language.

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

The gas pump **GP-2** component supports the following operations:

Activate (int a, int b)	// the gas pump is activated, where *a* is the price of the Regular gas
	// *b* is the price of Diesel gas per gallon
Start()	//start the transaction
PayCredit()	// pay for gas by a credit card
Reject()	// credit card is rejected
Approved()	// credit card is approved
Diesel()	// Diesel gas is selected
Regular()	// Regular gas is selected
StartPump()	// start pumping gas
PayDebit(int p)	// pay for gas by a debit card, where *p* is a pin #
Pin(int x)	// pin # is provided, where *x* represents the pin #
Cancel()	// cancel the transaction
PumpGallon()	// one gallon of gas is dispensed
StopPump()	// stop pumping gas
FullTank()	// Tank is full and the pump is stopped

Both gas pump components are state-based and are used to control simple gas pumps. Users can pay by cash, a credit card, or a debit card. The gas pump may dispense 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 pumped
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 gas pump components using the Model-Driven Architecture (MDA) covered in the course. In the first part of the project, you should design an executable meta-model, referred to as MDA-EFSM, for gas pump components. This MDA-EFSM should capture the "generic behavior" of both gas pump components and should be decoupled from data and implementation details. Notice that in your design, there should be **ONLY** one MDA-EFSM for both gas pump 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.

SUBMISSIONS & DEADLINES

Part I: MDA-EFSM

MDA-EFSM model report for the gas pump 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 the Input Processors of *GP-1* and *GP-2*

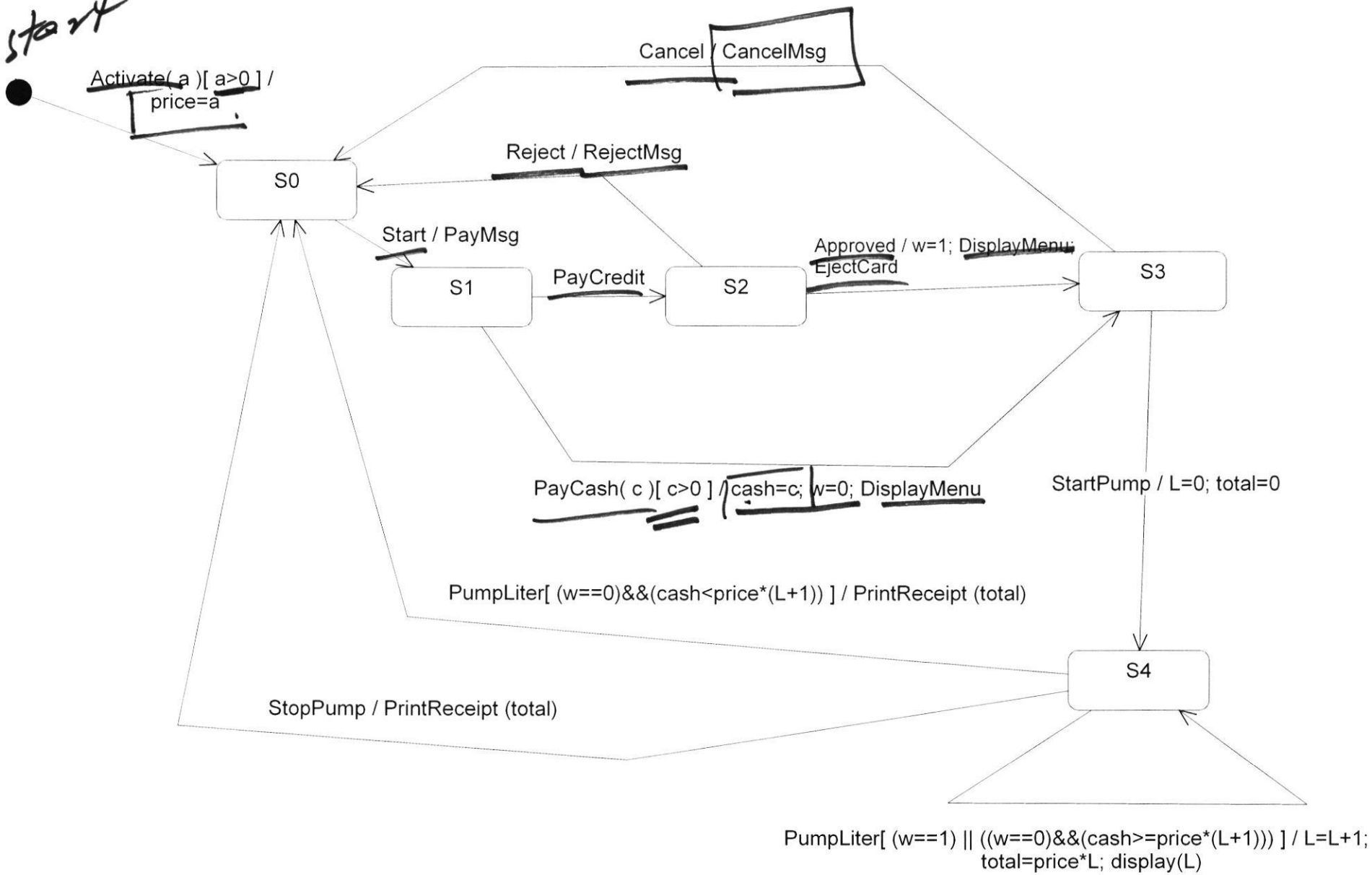


Figure 1: EFSM for gas pump GP-1

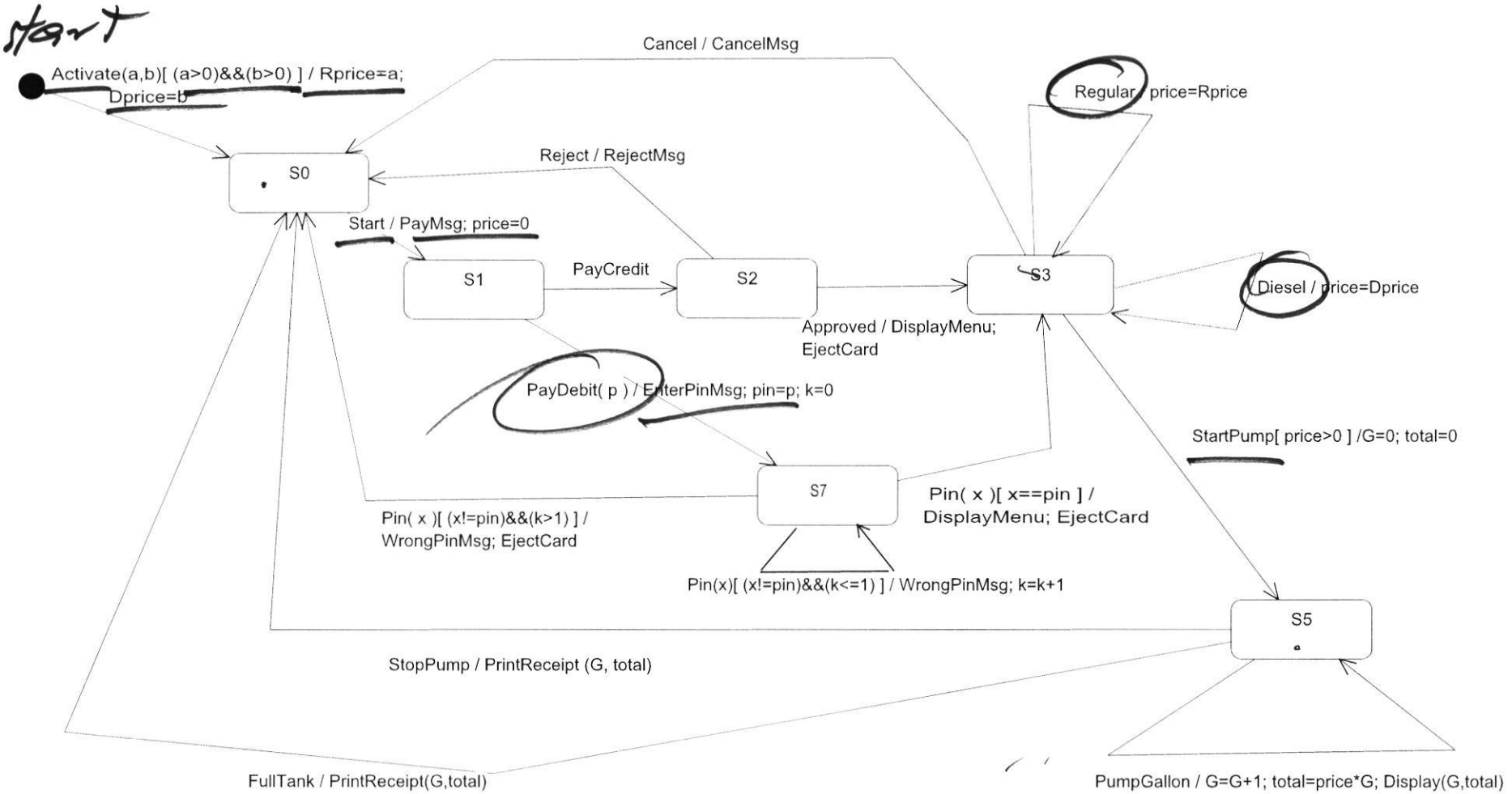


Figure 2: EFSM for gas pump GP-2

Project

Use MDA architecture to design and implement a "family" of Gas Pump systems/components using Model driven architecture.

Two Gas Pumps

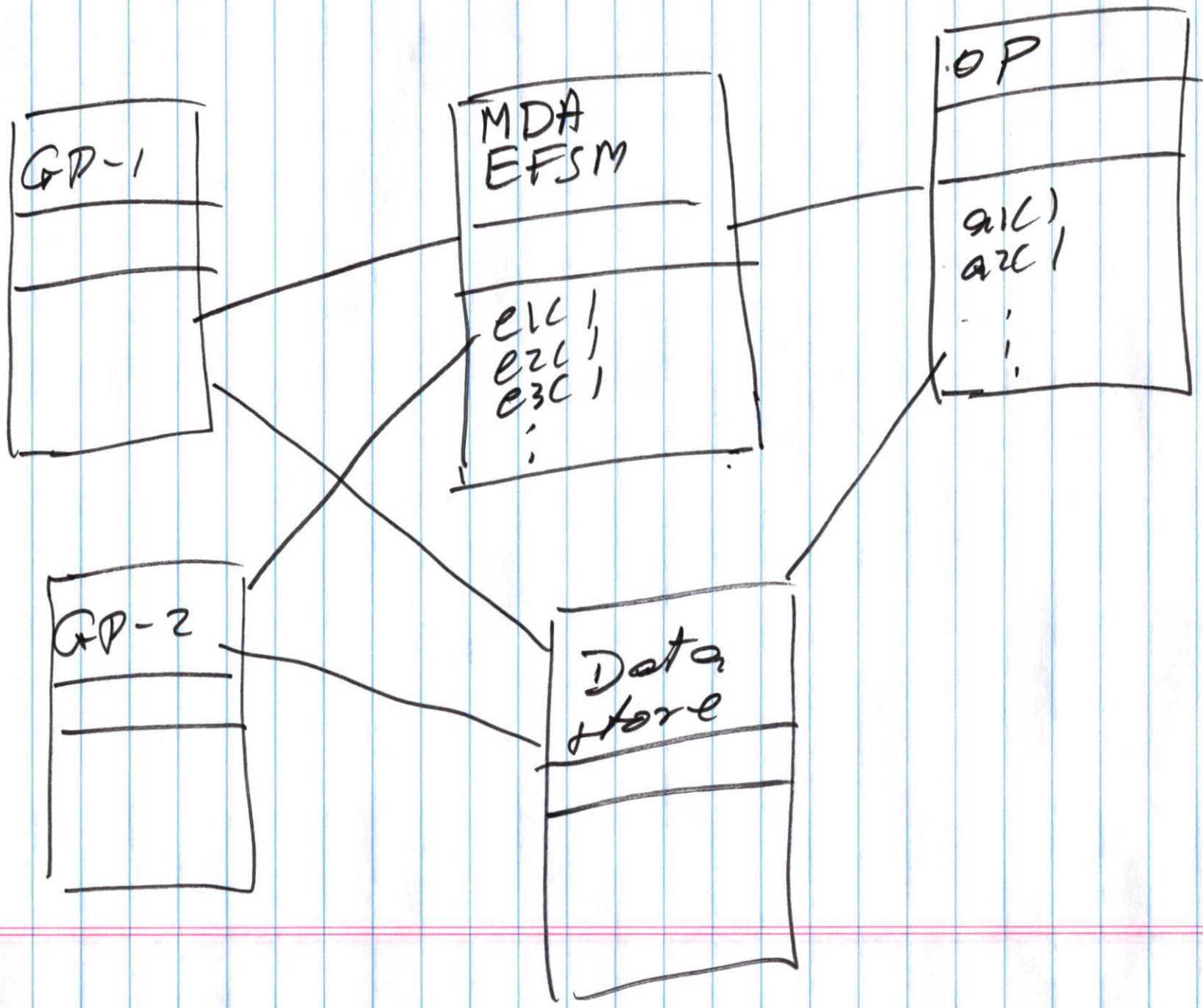
GP-1 GP-2.

Behavior of GP1 is described as EFSM of Fig 1

Behavior for GP-2 in Fig 2.

Differences between GPs

* operation names
* different signatures.
* different type of gas is disposed.
* types of payment
* units of gas.
* dispay menus
* messages
* data types
* . . .


GP-1
MDA
EFSM
e1c1
e2c1
e3c1
OP
q1c1
q2c1
GP-2
Data
store

First Step

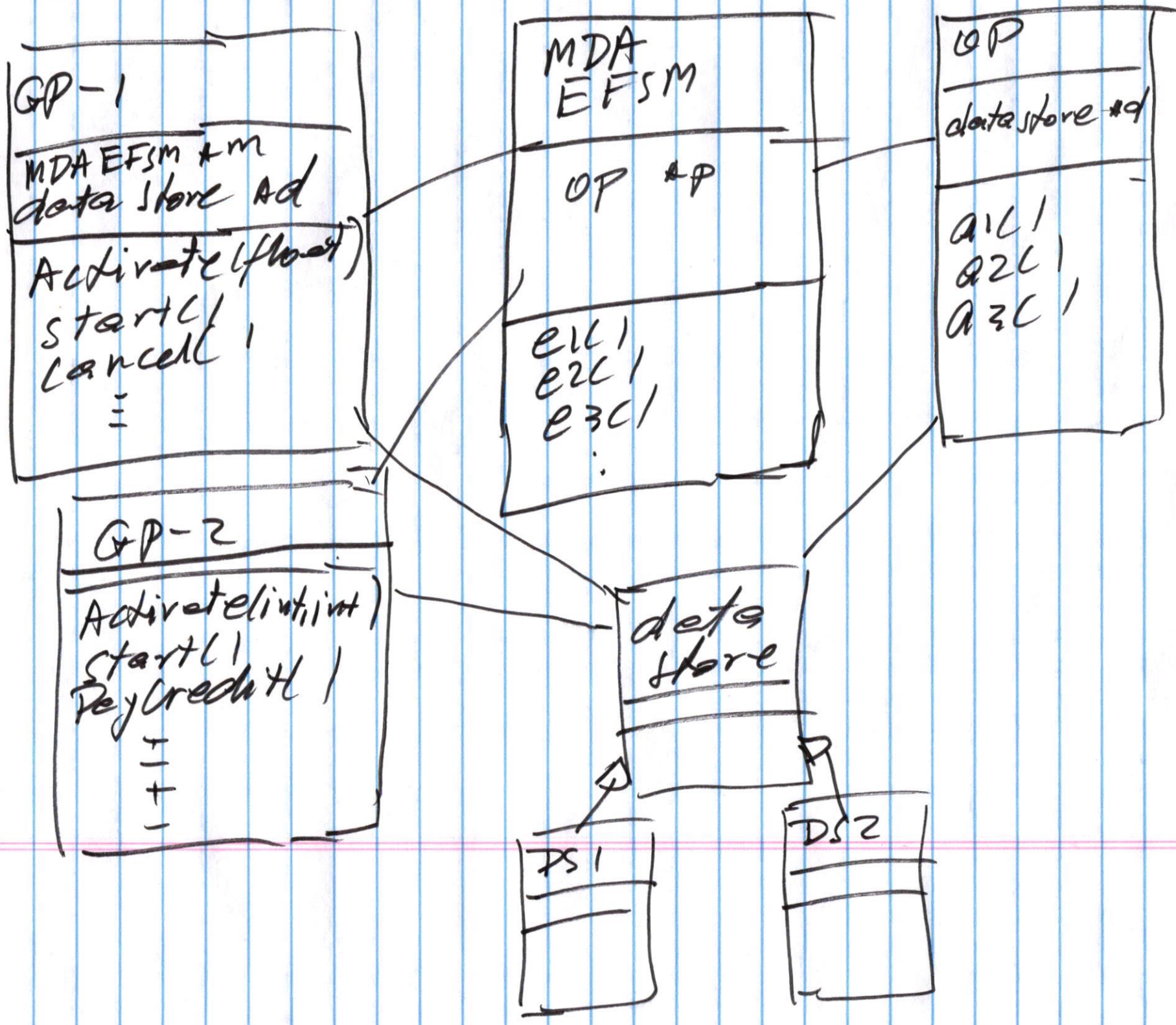
deadline: November 7

to create MDA-EFSM that captures "meta/generic" behavior of two gas pumps.

(1) identify meta events.

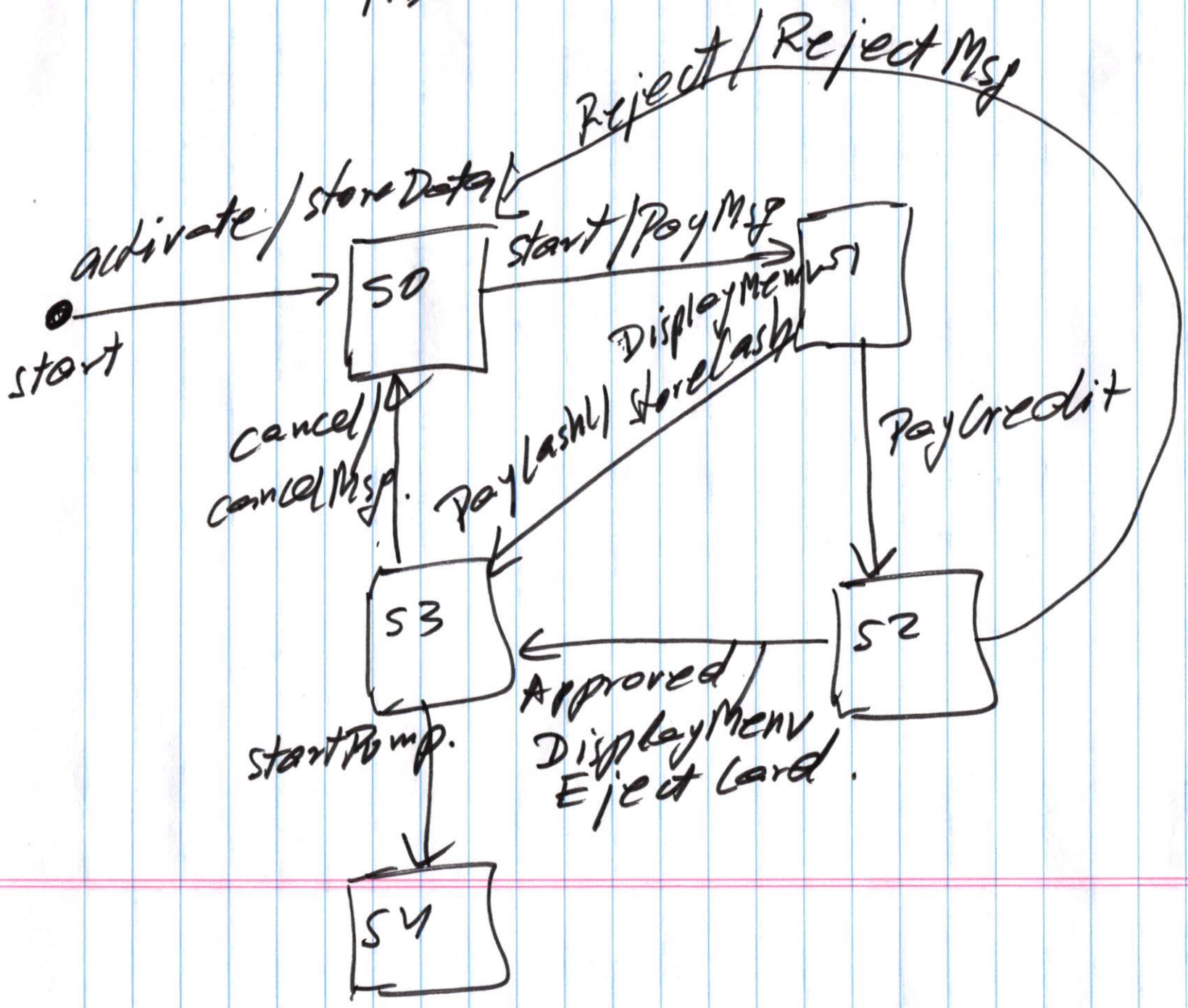
(2) identify meta actions.

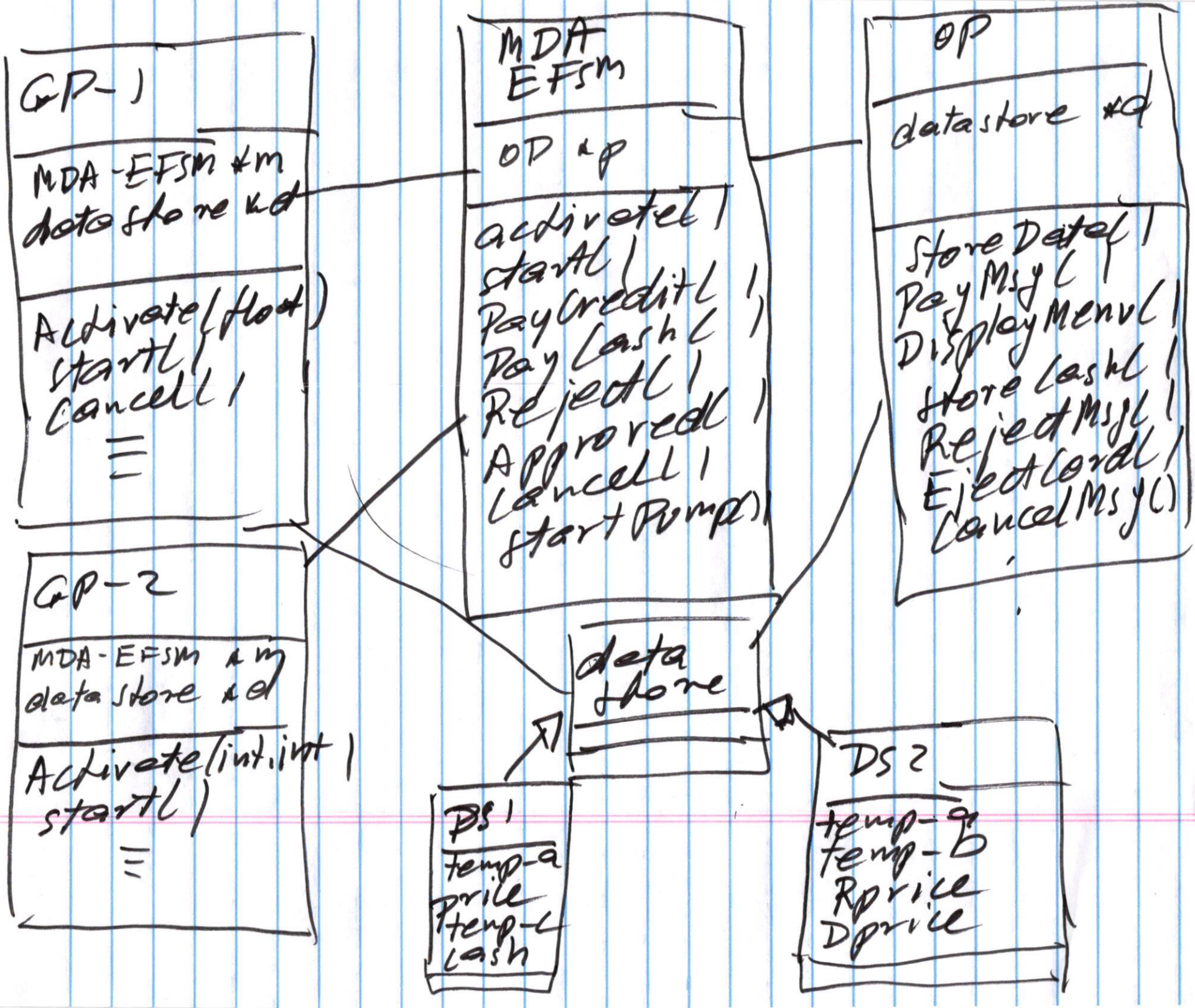
(3) MDA-EFSM state diagram
- states
- transitions.


GP-1
MDAEFsm *m
data Store Ad
Activate(float)
start()
cancel()
GP-2
Activate(int,int)
start()
KeyCredit()
MDA
EFSM
OP *p
e1()
e2()
e3()
OP
dataStore *d
a1()
a2()
a3()
data
Store
DS1
DS2

MDA-EFSM events	MDA-EFSM actions
activate() start() PayCredit() PayCash() Reject() Approved() Cancel() StartPump()	StoreData() PayMsg() DisplayMenu() StoreCash() RejectMsg() EjectCard() CancelMsg()

MDA - EFSM


Reject / Reject Msg
activate / storeData
start / PayMsg
start
S0
S1
DisplayMenu
cancel /
cancelMsg.
payCash / storeCash
PayCredit
S3
S2
Approved /
DisplayMenu
Eject card.
startPromp.
S4


GP-1
MDA-EFSM *m
data store *d
Activate(float)
start()
cancel()
=
MDA EFSM
OD *p
activate()
start()
PayCredit()
PayCash()
Reject()
Approved()
Cancel()
startPump()
OP
datastore *d
StoreData()
PayMsg()
DisplayMenu()
StoreCash()
RejectMsg()
EjectCard()
CancelMsg()
GP-2
MDA-EFSM *m
data store *d
Activate(int,int)
start()
=
data store
DS1
temp-a
price
temp-c
cash
DS2
temp-a
temp-b
Rprice
Dprice

GP-1 class

Activate(float a)

{ if (a > 0) {

d -> temp_a = a

m -> activate()

}

---

start()

{ m -> start()

}

PayCredit()

{ m -> PayCredit()

}

PayCash(float c)

{ if (c > 0) {

d -> temp_c = c

m -> PayCash()

}

GP-1 class

Reject()
{ m -> Reject()
}

Approved()
{ m -> Approved()
}

startPump()
{ m -> startPump()
}

GP-2 class

```
Activate(int a, int b)
{ if ((a>0)&&(b>0)) {
      d->temp_a = a
      d->temp_b = b
      m->activate()

}
```

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

```
Pay credit()
{  m-> pay credit()
}
```

```
Reject()
{  m->reject()

}
```

GP-2

PayDebit(int p)

{

=

m → ??? . Debit

}

selection of gas type.

GP-2:

Regular
Diesel

~~meta events~~

~~Regular~~
~~Diesel~~

???

Posted sample
MDA - EFSM for
(two 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.

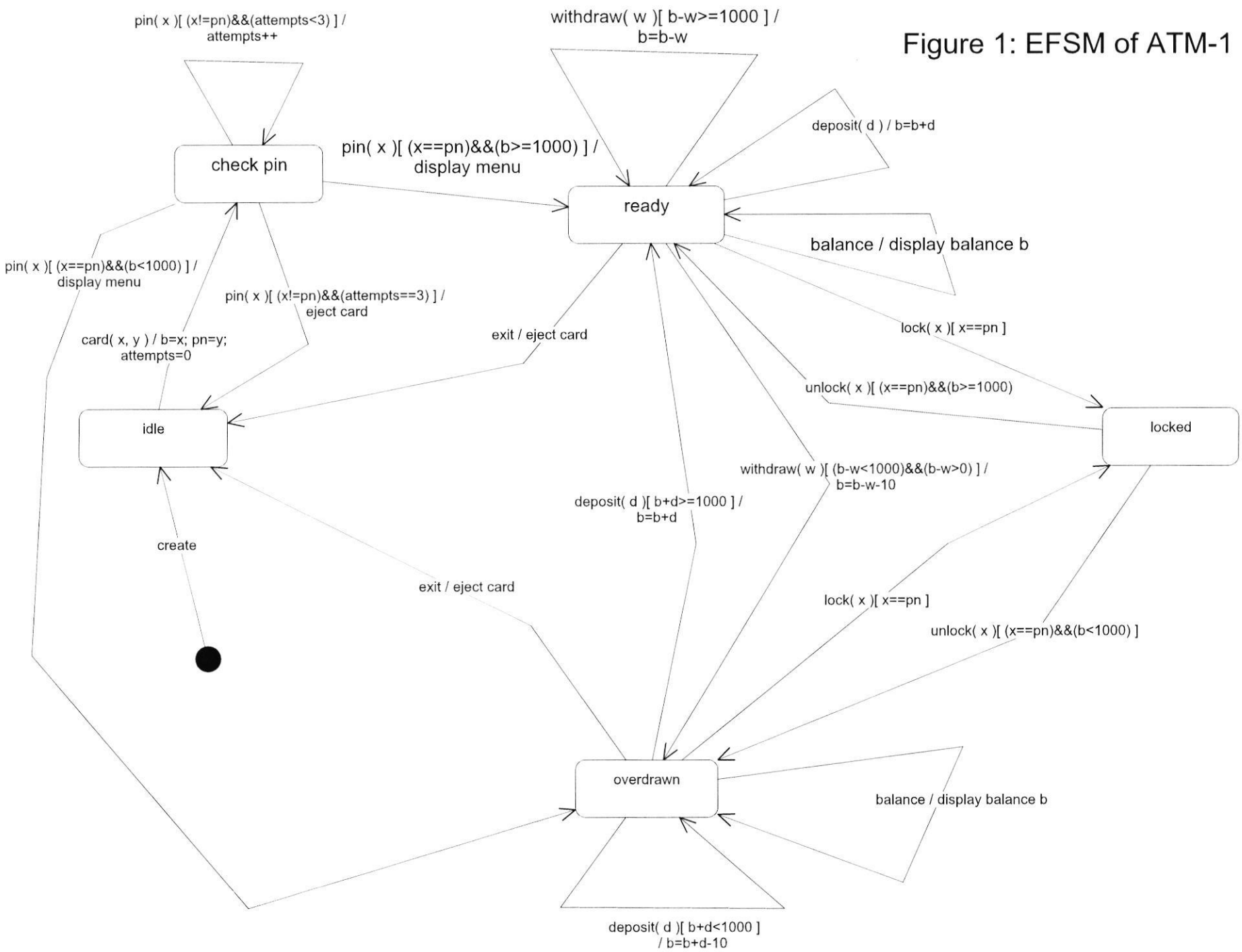


Figure 1: EFSM of ATM-1

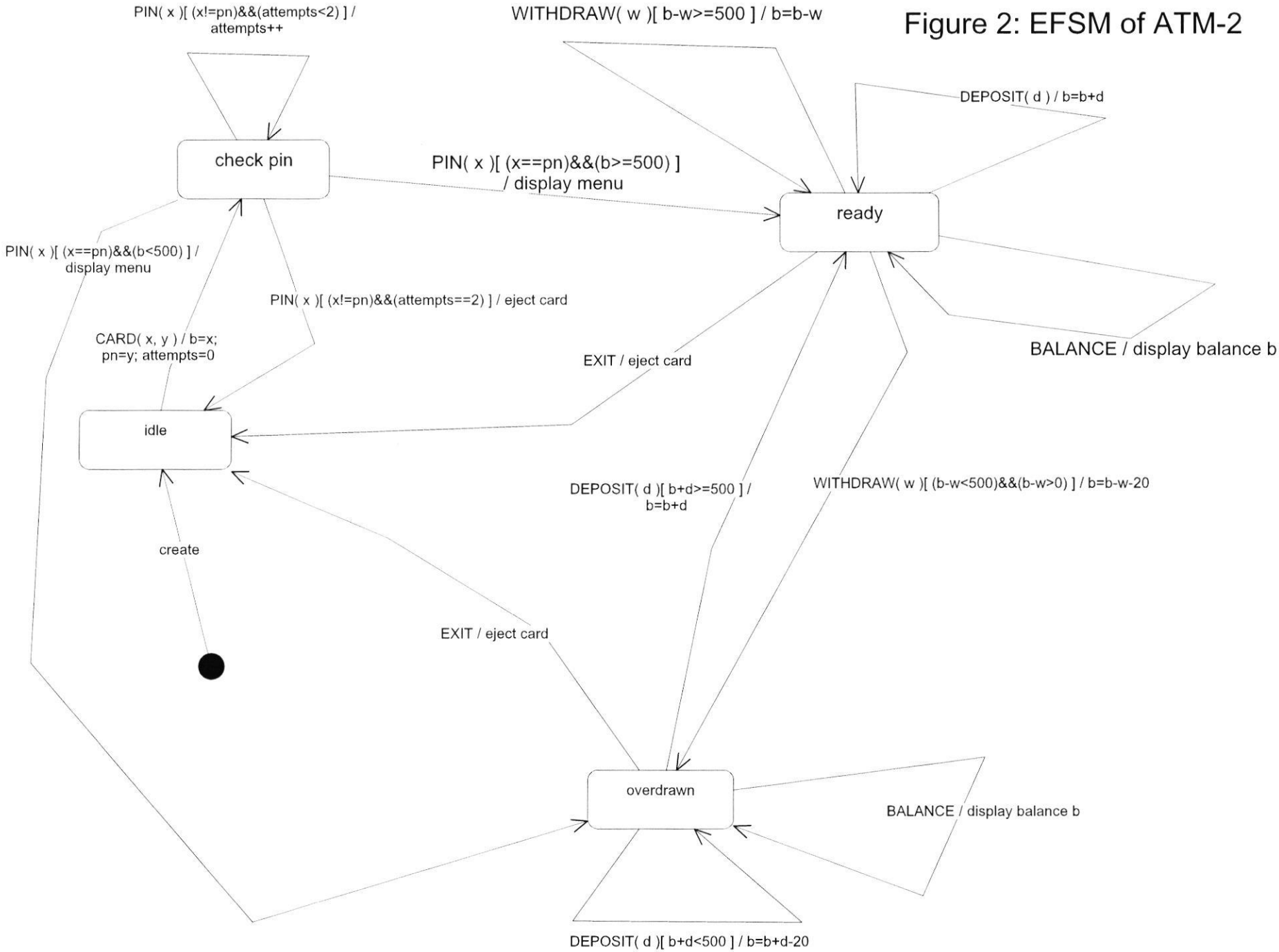
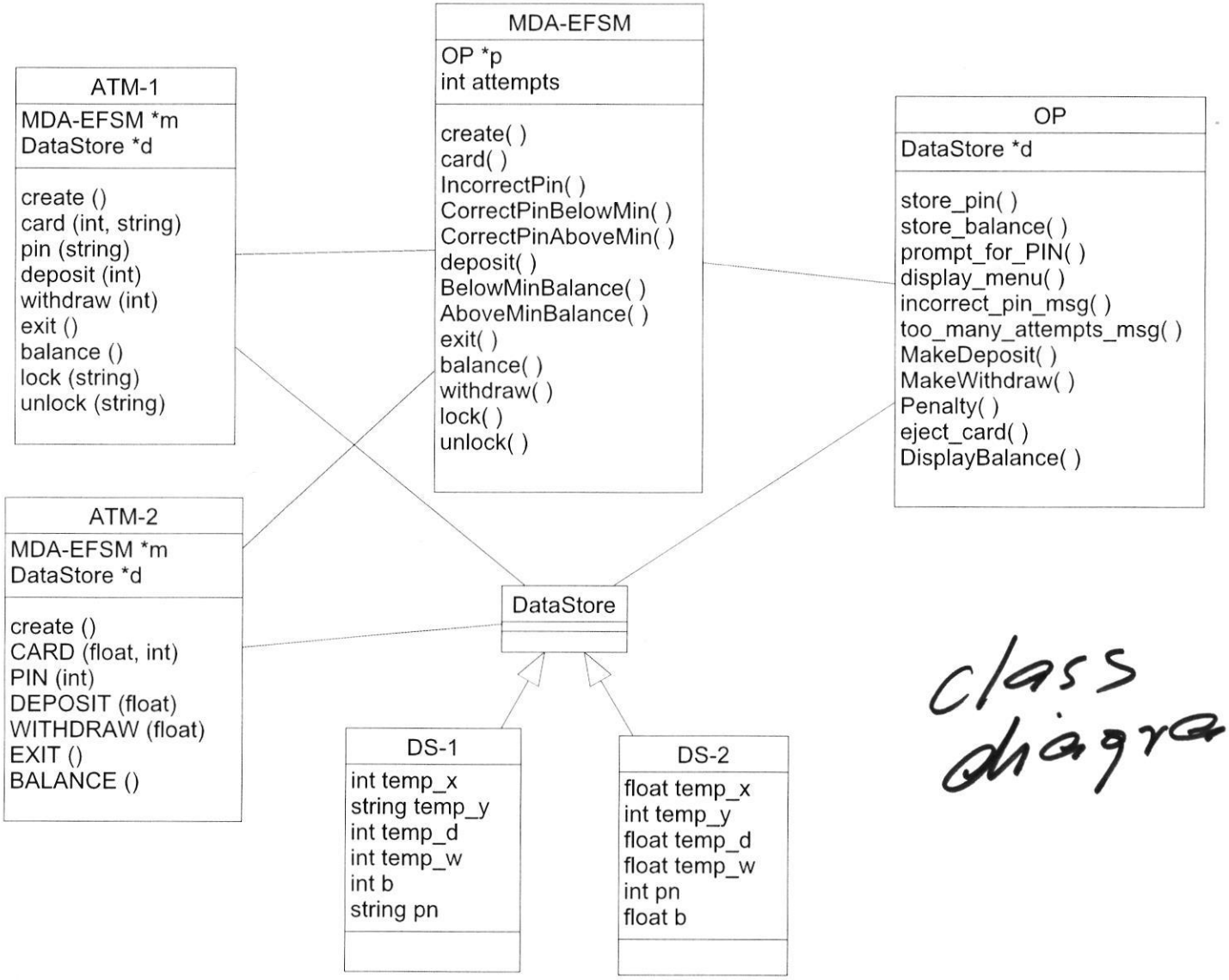
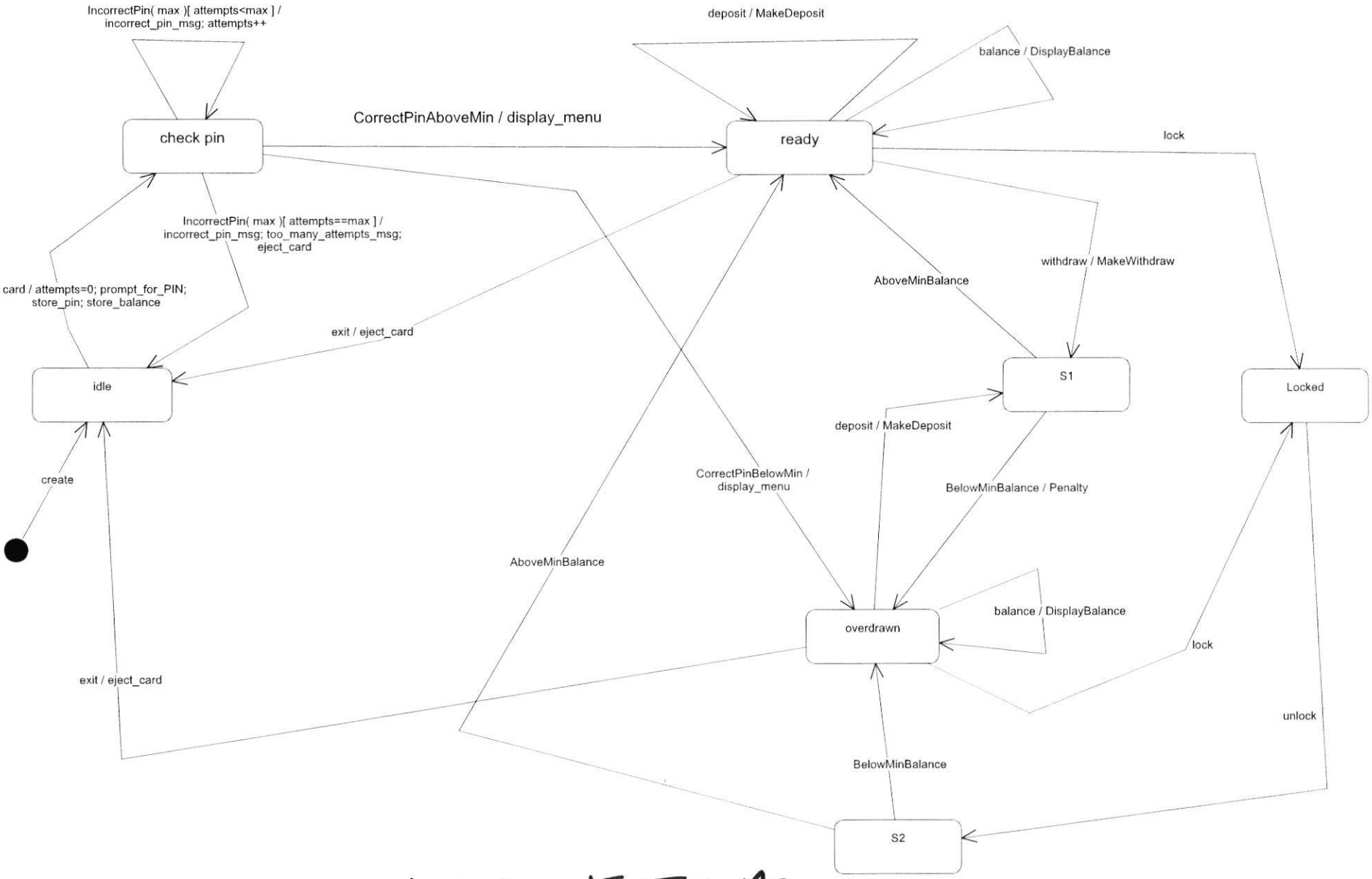


Figure 2: EFSM of ATM-2



class diagram.


IncorrectPin(max)[attempts<max] /
incorrect_pin_msg; attempts++
deposit / MakeDeposit
balance / DisplayBalance
check pin
CorrectPinAboveMin / display_menu
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.

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

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 #