

Exam #2

Wednesday, October 29

at 5:00 pm

The coverage is posted
A sample exam is posted

Project description
part #1

Sample MDA-EFSM for
ATMs 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

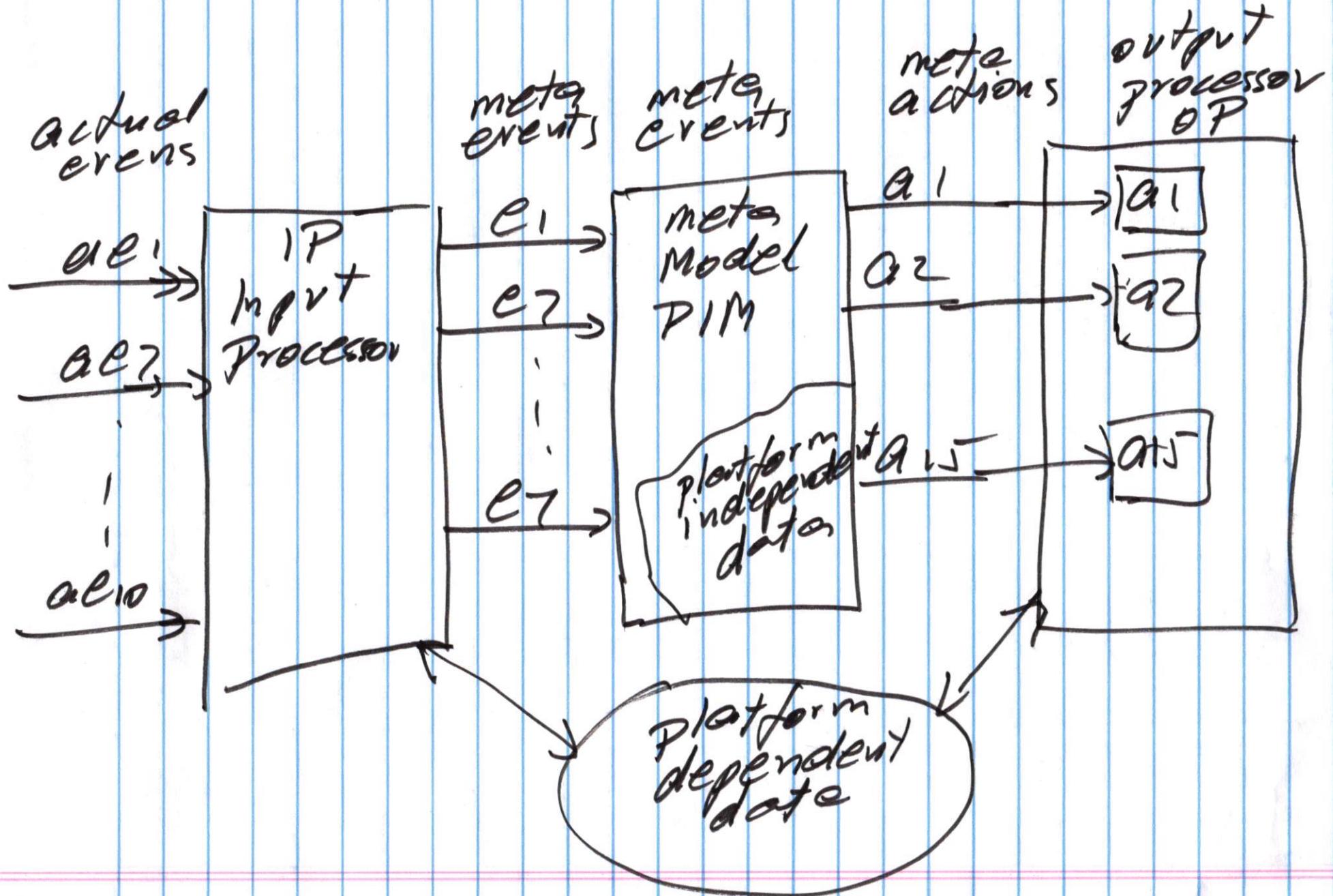
Project Model Driven Architecture MDA

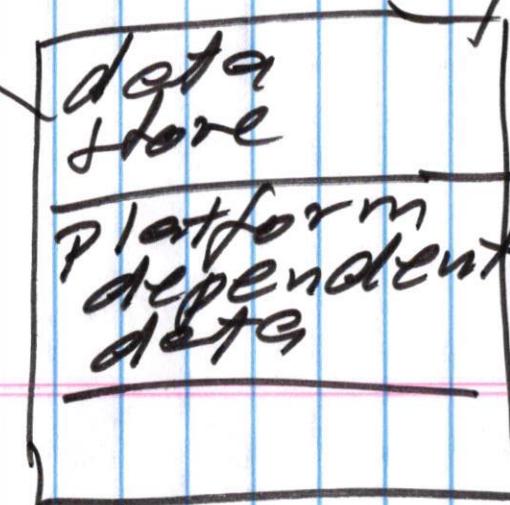
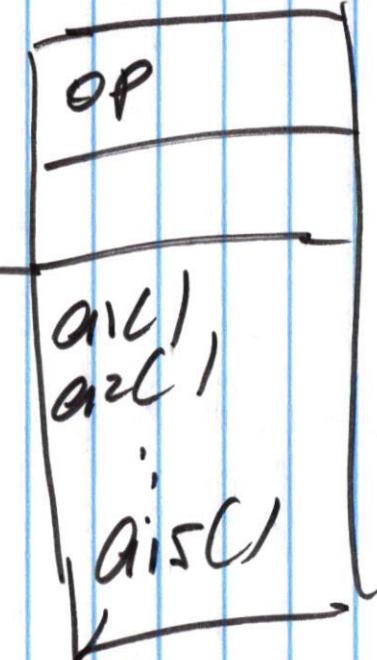
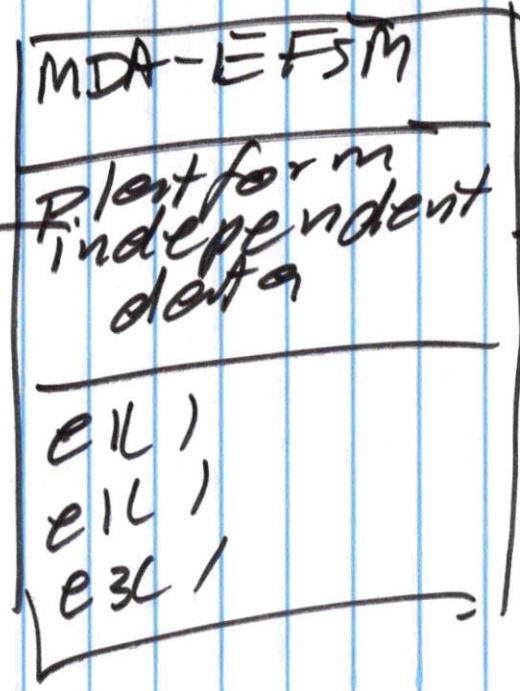
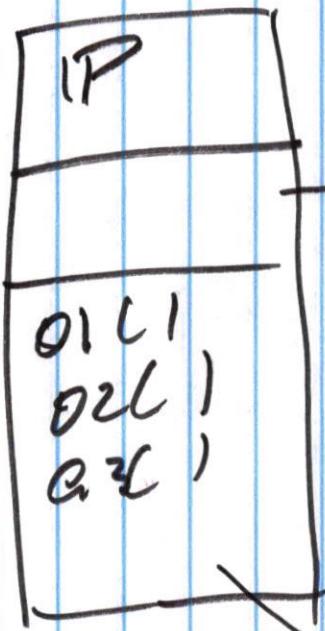
separate

platform independent
state-based behavior
of "family" of systems.

FROM

platform specific
behavior of the family
of systems.





meta state behavior

MDA - EFSM

EFSM with restrictions

+ no data

or

platform independent
data

* platform independent
meta events
* meta actions.

Responsibilities

IP operations

- * check conditions.
- * store parameters in temp. area in data store to be used by actions operations of DP.
- * invoke meta events) on MDA-EFSM.
- * DO NOT update state in data store.

MDA-EFSM

- * accept meta events.
- * execute MDA-EFSM model.
- * issue/invoke meta actions in BP for execution.
- < can have access to to its own platform independent data.
- < DOES NOT have access to data store.

OP

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

Project - part #1

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

Two Gas Pumps

GP-1

GP2.

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:

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

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

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

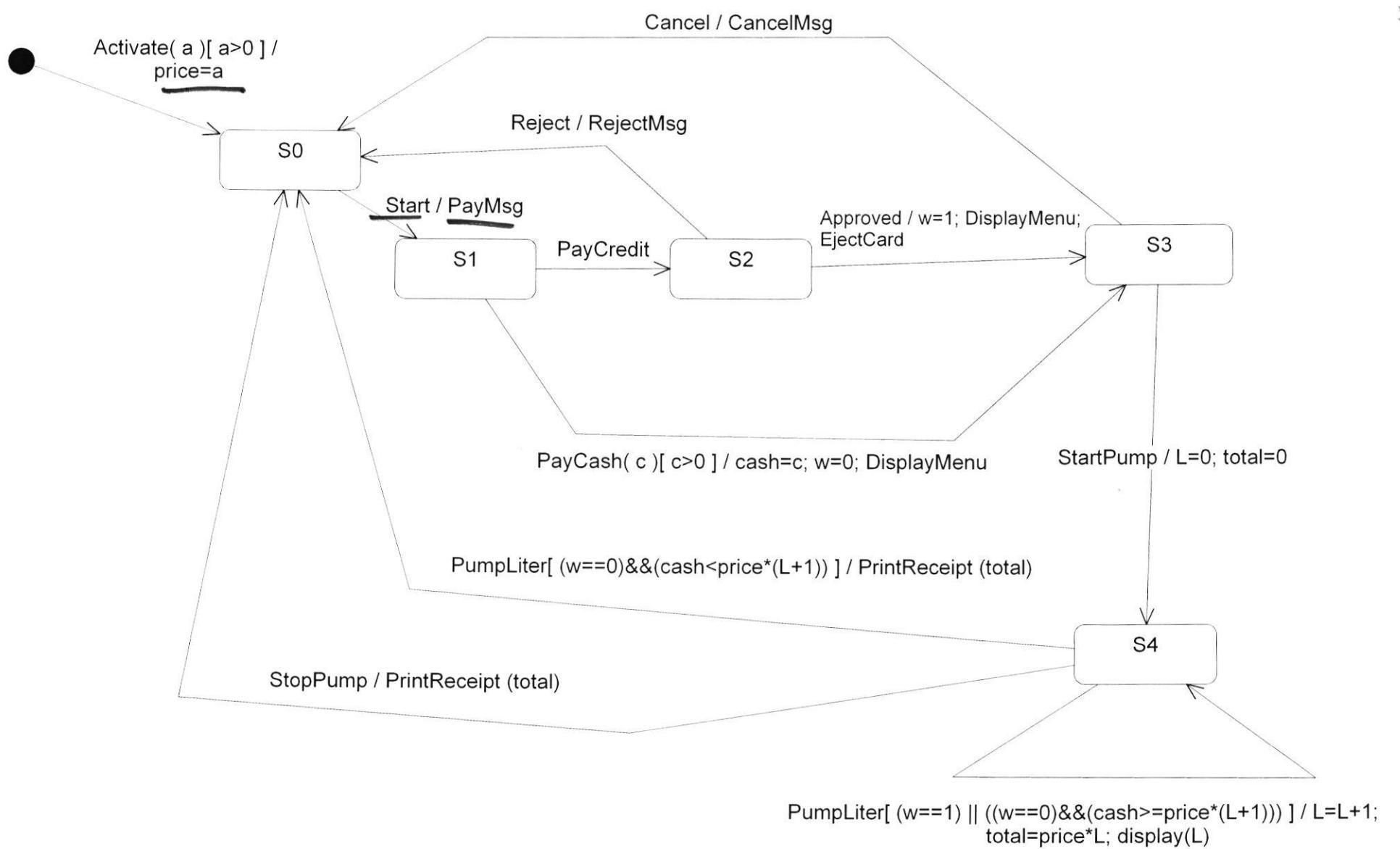


Figure 1: EFSM for gas pump GP-1

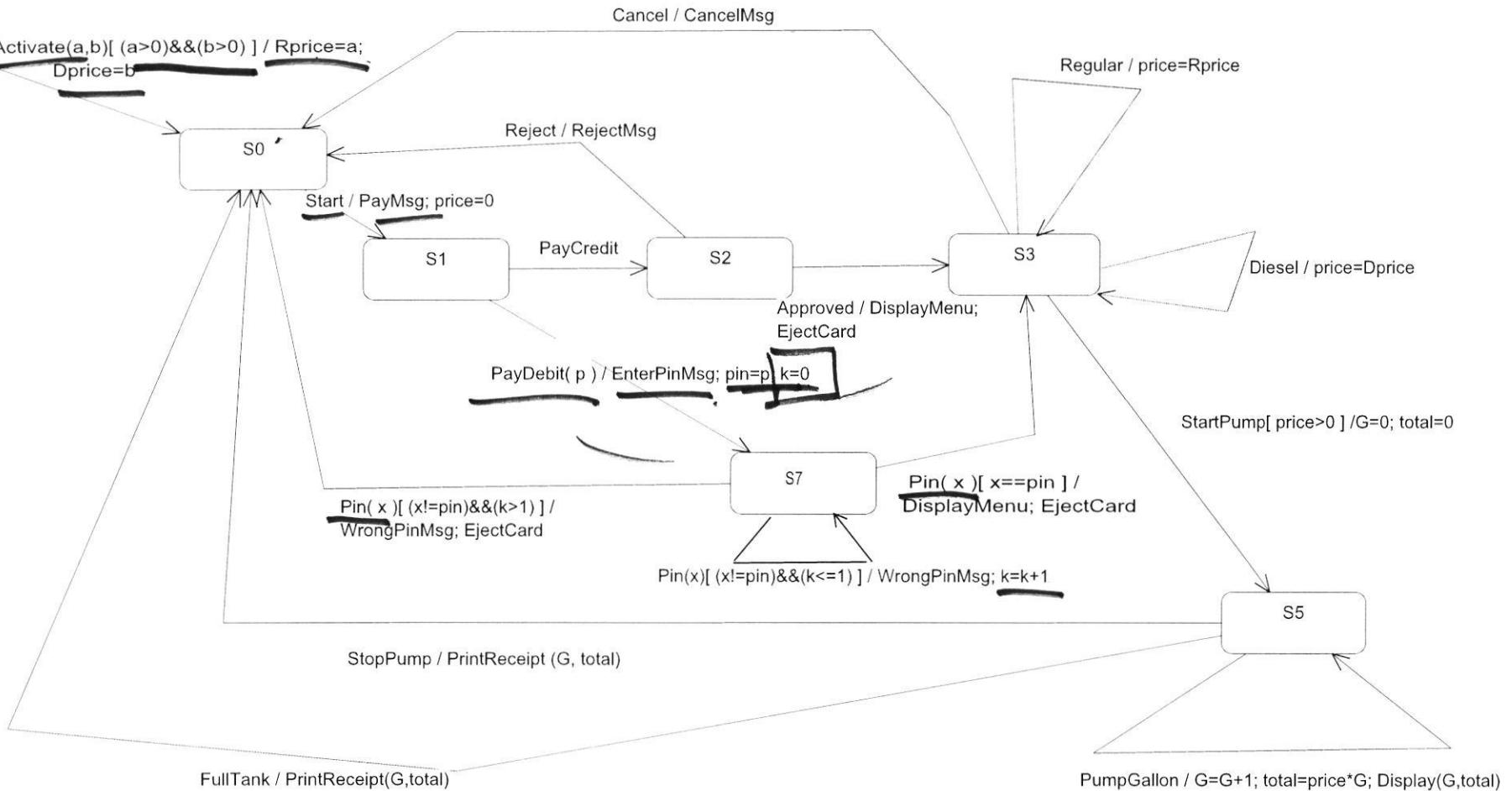


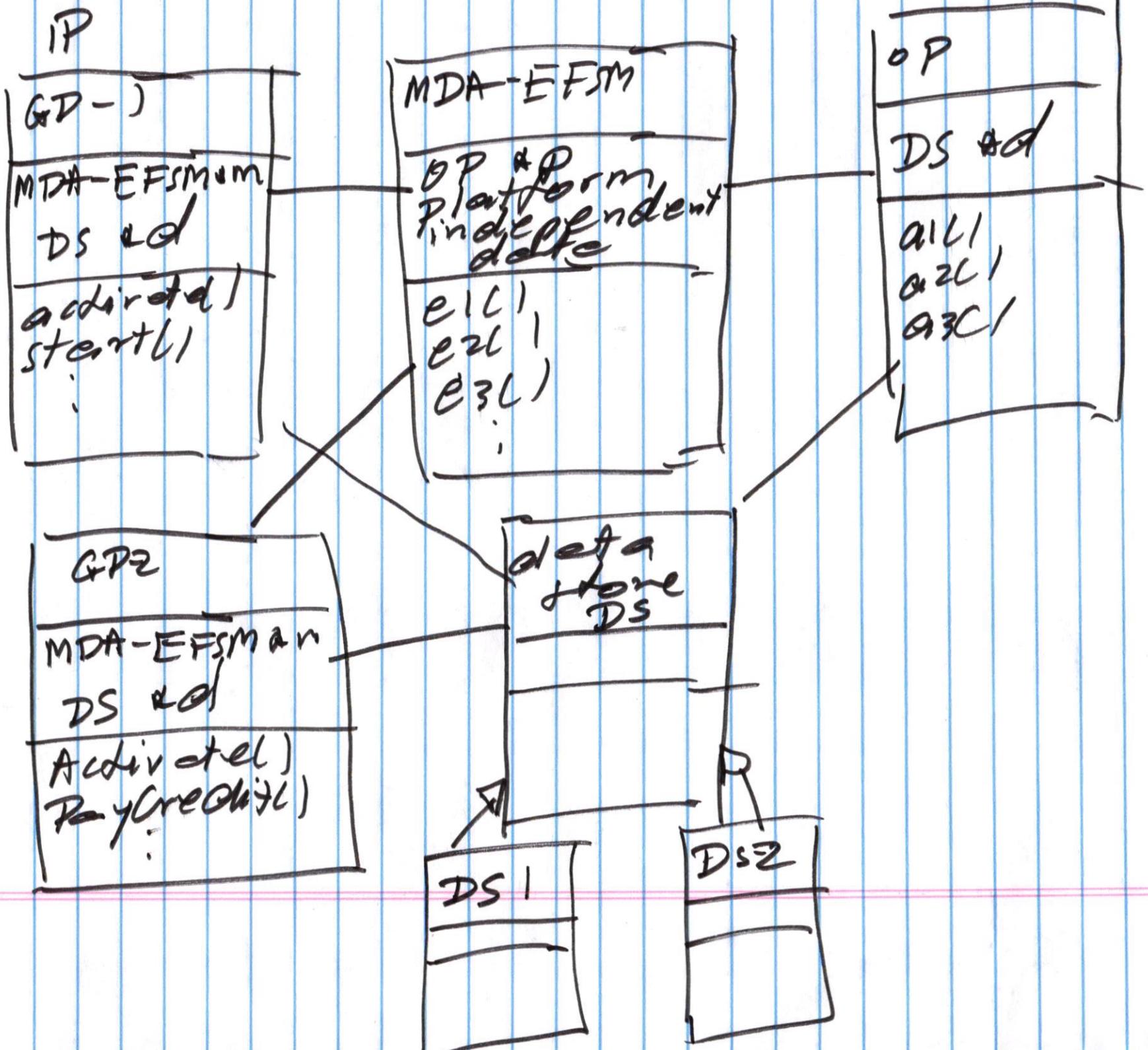
Figure 2: EFSM for gas pump GP-2

Differences

- * operation names / signatures
- * types of payment
- * types of gas
- * unit of disposed gas .
- * display menus
- * messages .

K ..

K ..



Part #1

create MDA-EFSM
that captures "meta/general"
behavior of GP-1 and GP-2

Deadline: November 7.

1. class diagram
2. Identify meta events
3. identify meta actions
(provide description
for each meta action)
4. State Diagram
MDA-EFSM.
 - * states
 - * transitions
5. Pseudo code for
all operations of
GP-1 & GP-2

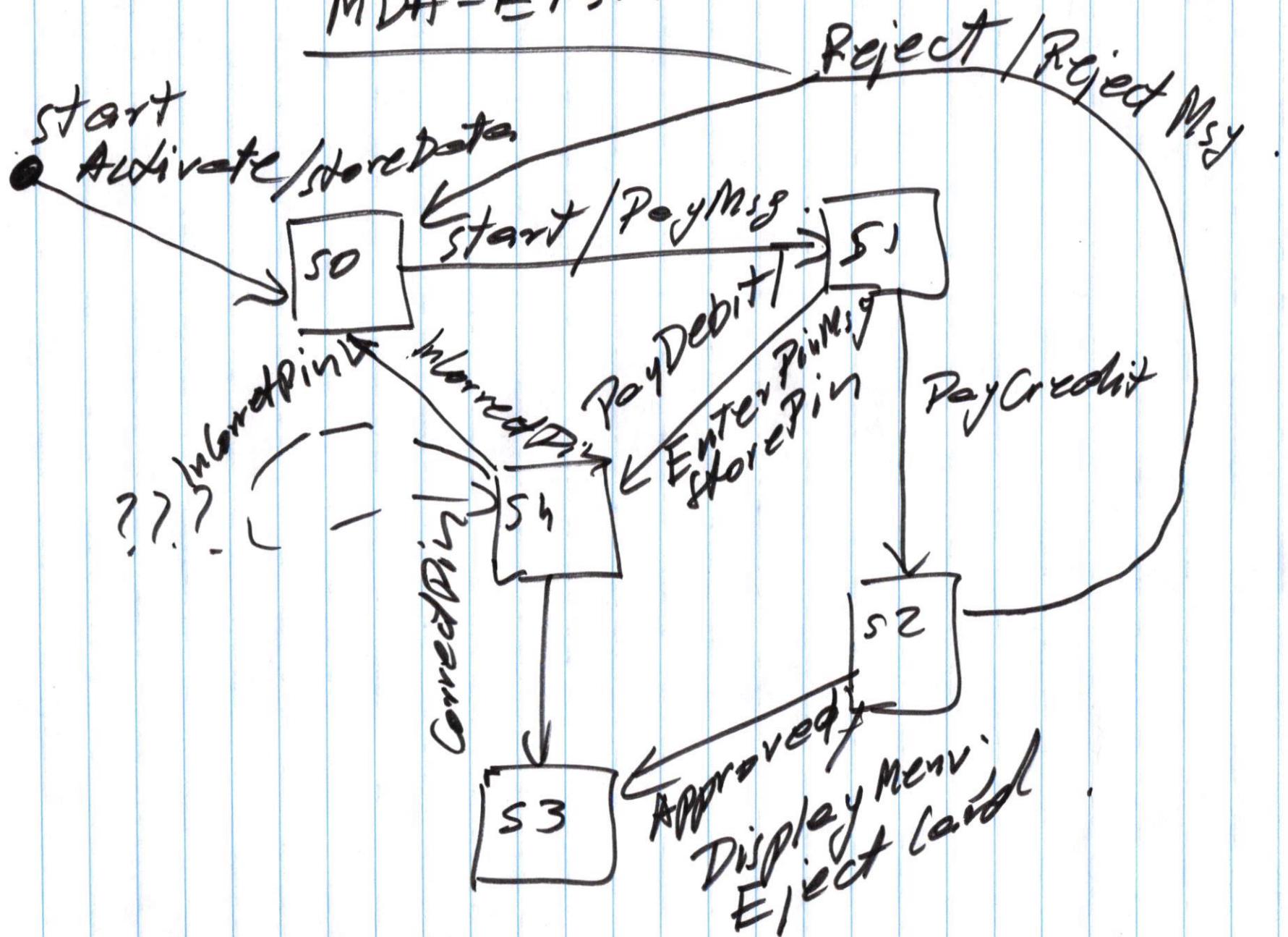
metas events

Activate() |
start()
PayCredit()
Approved()
Reject()
PayDebit() |
CorrectPin()
IncorrectPin() |

meta actions

StoreData()
PayMsg()
DisplayMenu()
EjectCard()
RejectedMsg()
EnterPinMsg.
StorePin()

MDA - EFSM



Q8-2

addint(a, , int b)

i. if($a > 0 \text{ and } b > 0$)

$$\begin{cases} d \rightarrow \text{temp_a} = a \\ d \rightarrow \text{temp_b} = b \end{cases}$$

$m \rightarrow \text{addintee}$)

h
h

DS 2

int temp-a
int temp-b

int Rprice
int Dprice

int temp-R
int ~~temp~~ pin

start()

↳ m → start()

↳

PayCredit()

↳ m → PayCredit()

↳

Approved()

↳ m → Approved()

↳

Reject()

↳ m → Reject()

↳

QD-2 class

PeyDebit (int p)

{ d → temp-p = p
m → PeyDebit()

↳

~~Pin (int x)~~

↳ ~~if (d > pin == x)
m → correctPin()~~

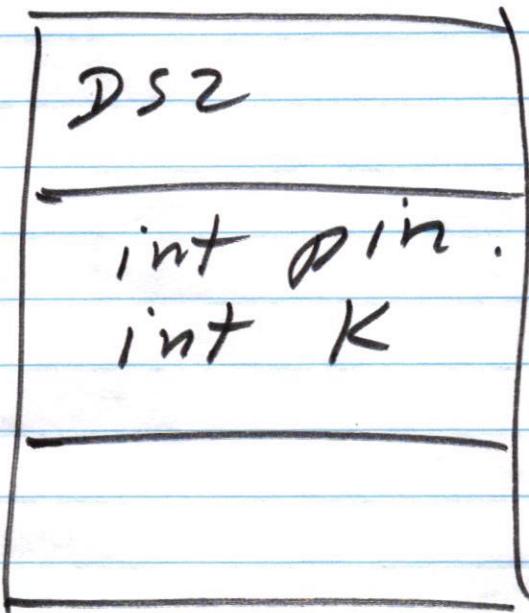
~~else m → IncorrectPin()~~

We need to introduce

counter k to keep
track of # of
incorrect attempts.

version # 1

K is in state store DS2



meta events

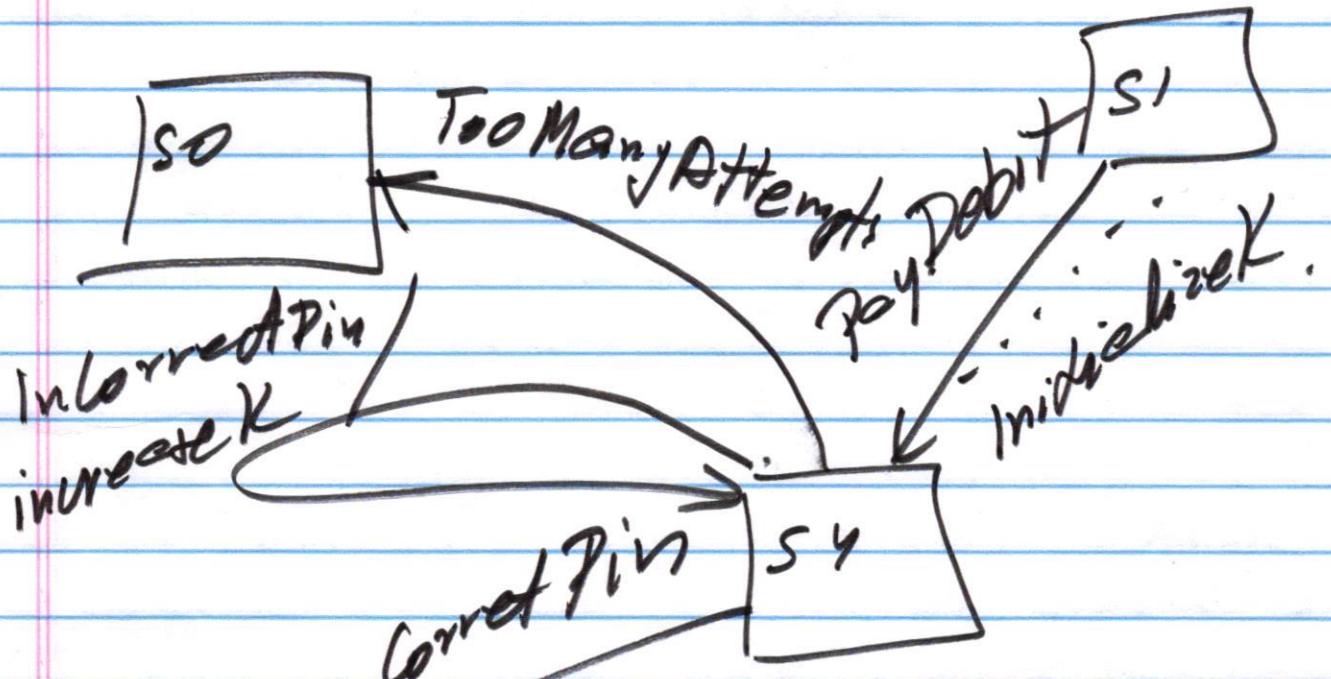
Pay Deposit()

Correct Pin

Incorrect Pin()

Too Many Attempts()

version # 1



S_3 Pin (int \leftarrow)

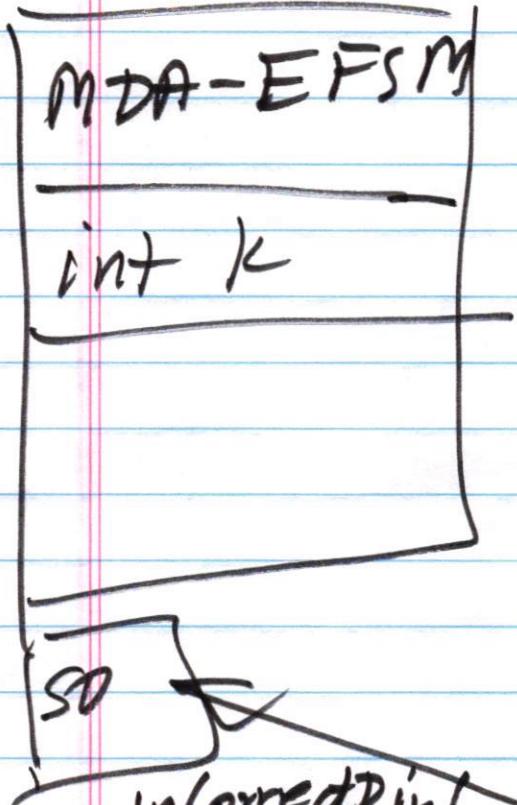
if ($d \rightarrow \text{pin} == v$)
m \rightarrow correctPin()

else if ($d \rightarrow \text{pin}.! = x$) \Rightarrow ($d \rightarrow k \leq 1$)
m \rightarrow IncorrectPin()

else if ($d \rightarrow \text{pin}.! = x$) \Rightarrow ($d \rightarrow k > 1$)
m \rightarrow TooManyAttempts()

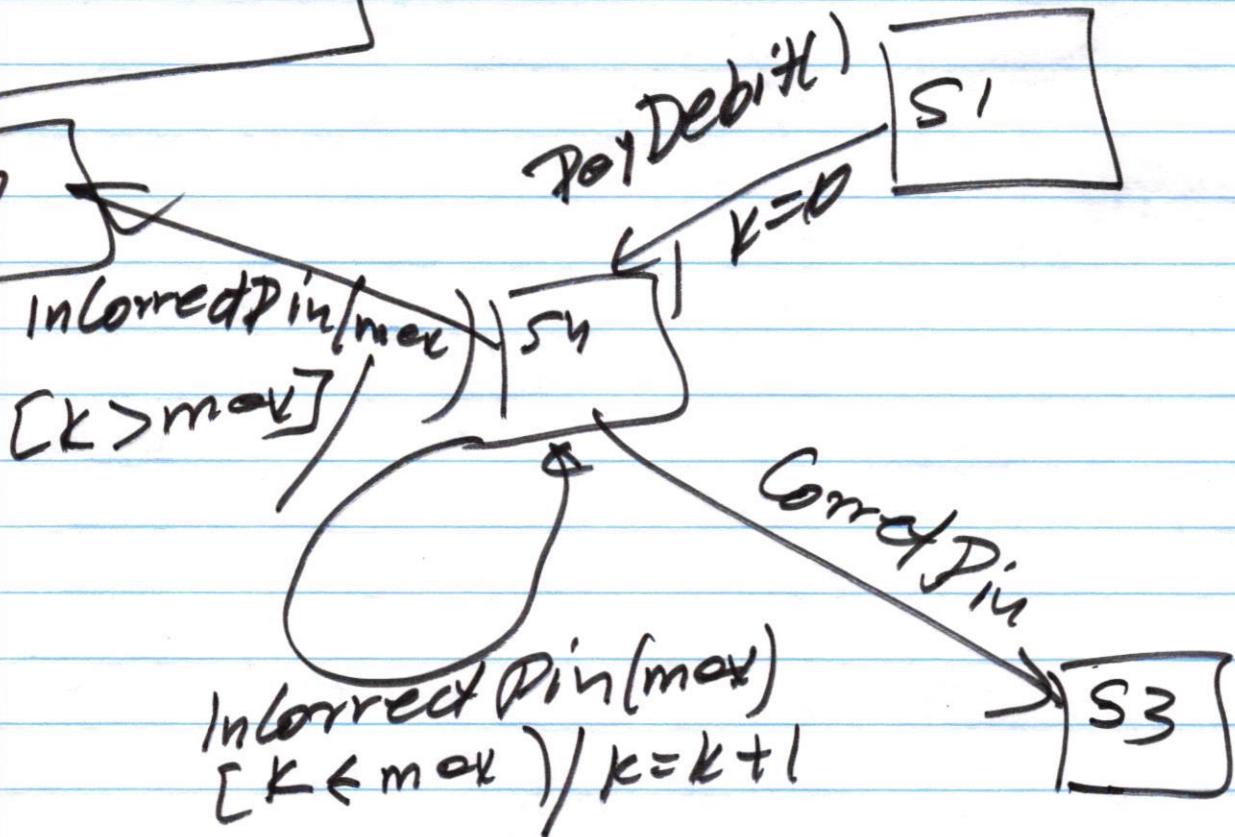
Version #2

K is in MDA-EFSM.



meta events

PayDebit()
CorretPin()
InCorrectPin()



Pin (int x)

↳ if ($d \rightarrow \text{pin} == x$) $m \rightarrow \text{correctPin}$

else $m \rightarrow \text{incorrectPin}(1)$

}

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

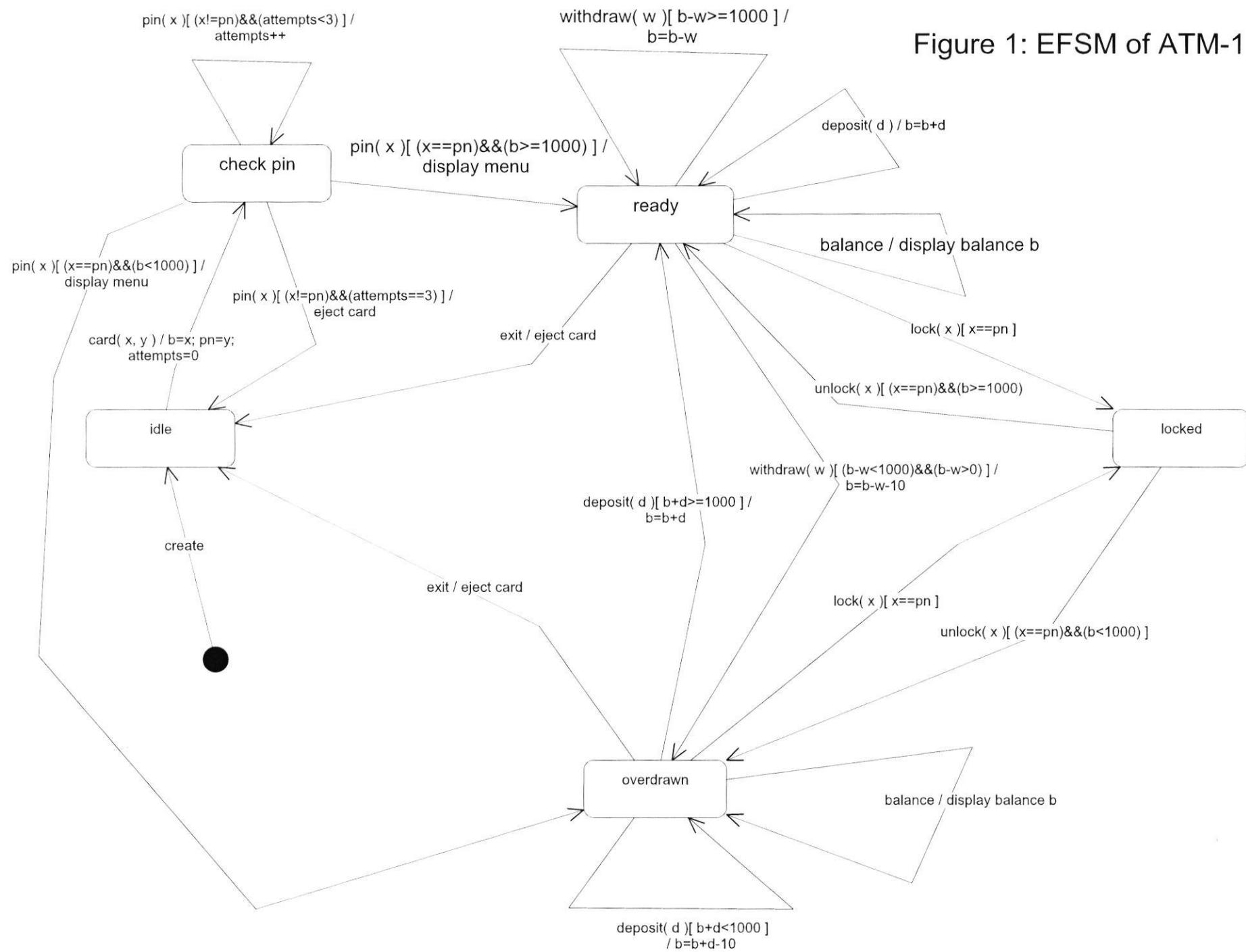
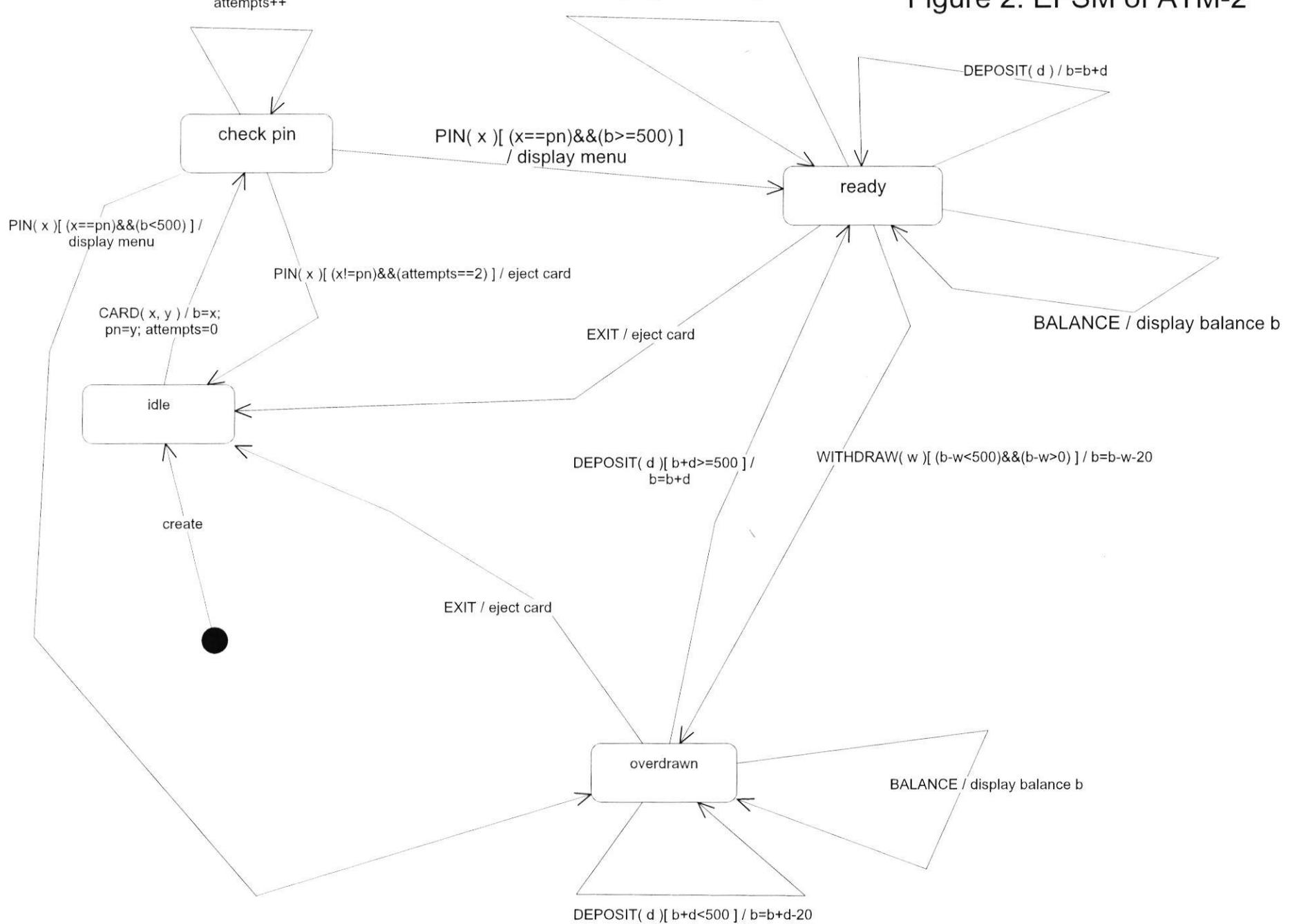
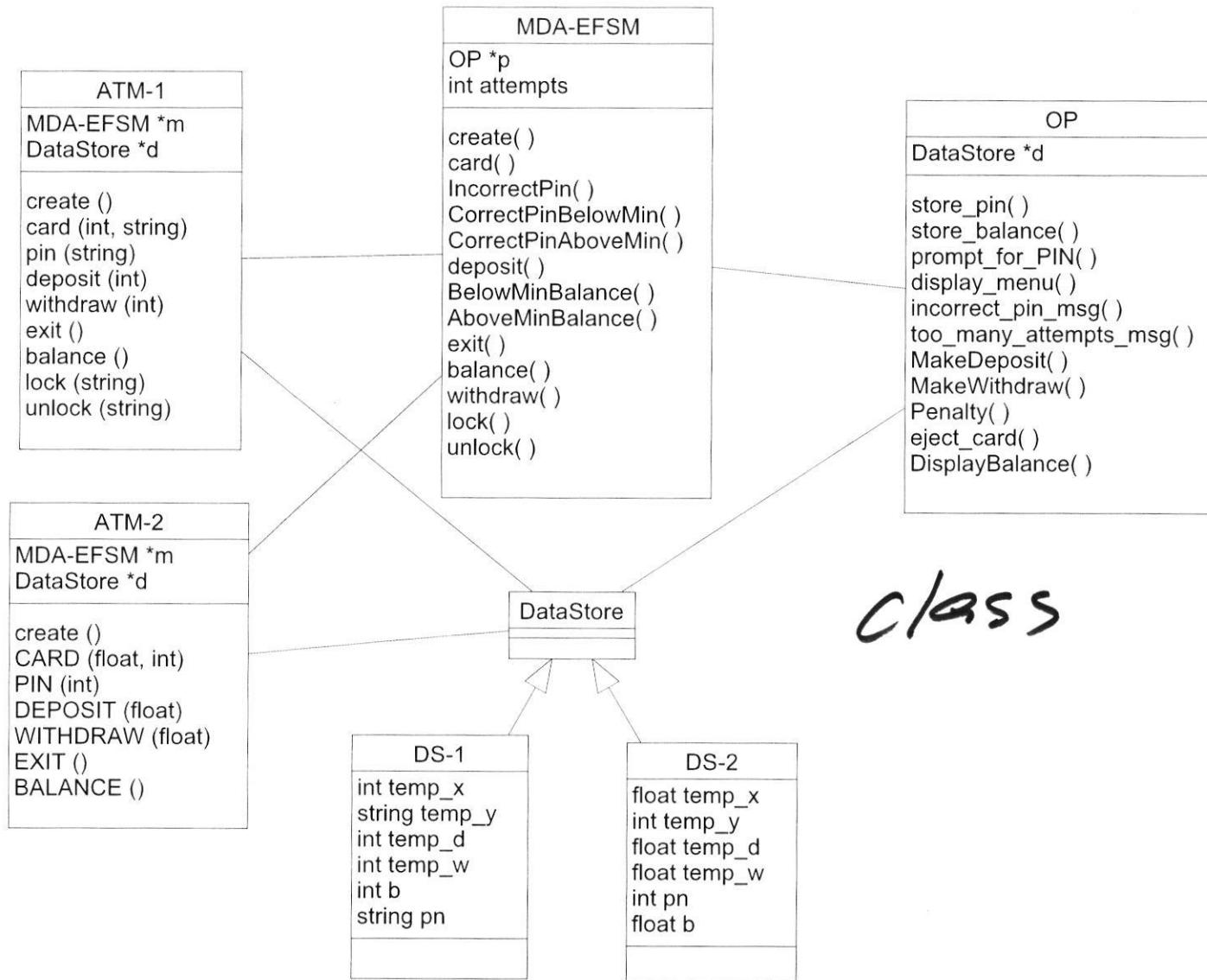
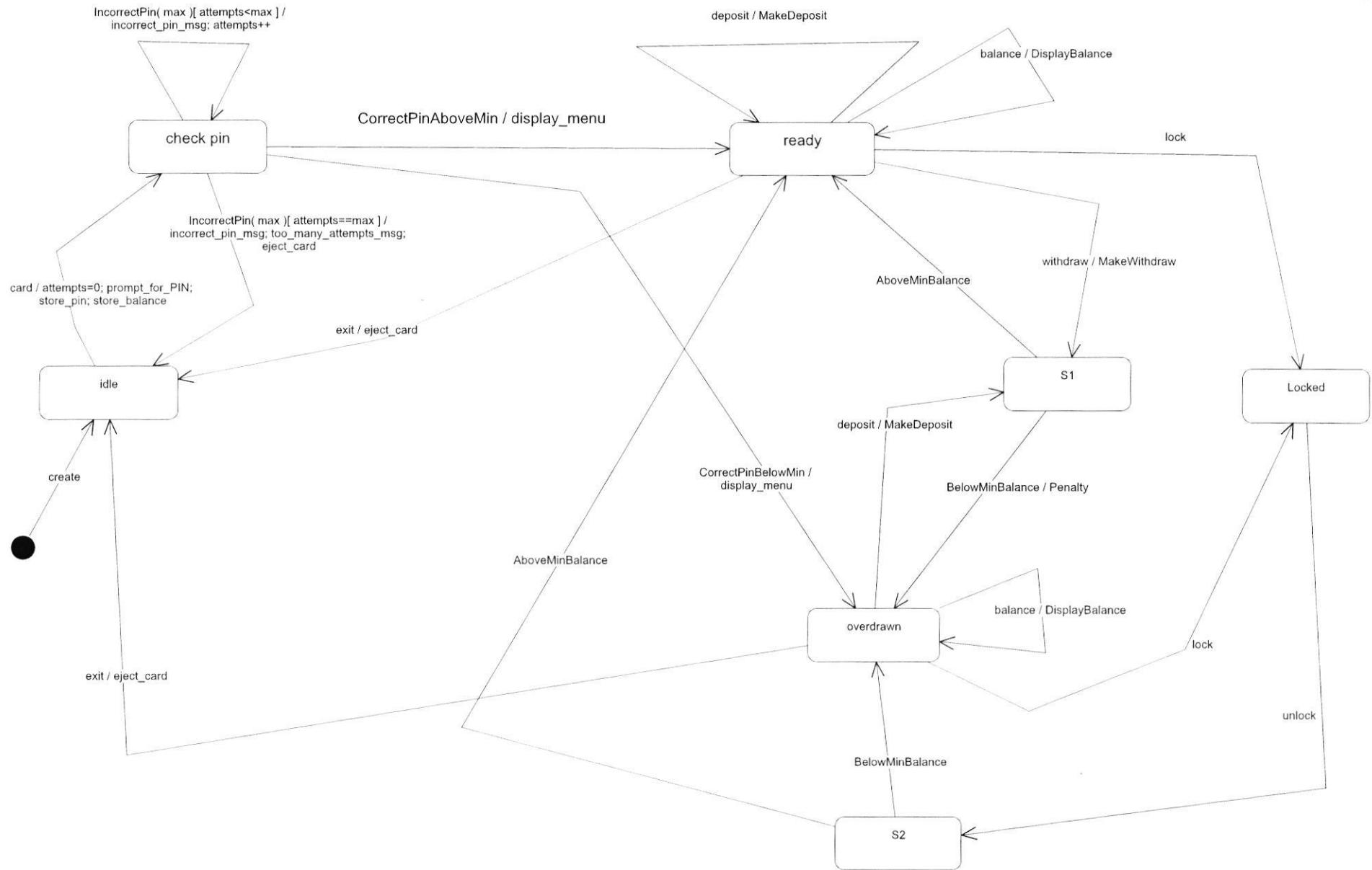


Figure 2: EFSM of ATM-2





class



MDA-EFSM Events:

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

MDA-EFSM Actions:

store_pin
store_balance
prompt_for_PIN
display_menu
incorrect_pin_msg
too_many_attempts_msg
MakeDeposit
MakeWithdraw
Penalty
eject_card
DisplayBalance

// stores pin from temporary data store to *pin* in data store
// stores pin from temporary data store to *b* in data store
// prompts to enter pin
// display a menu with a list of transactions
// displays incorrect pin message
// display too many attempts message
// makes deposit (increases balance by a value stored in temp. data store)
// makes withdraw (decreases balance by a value stored in temp. data store)
// applies penalty (decreases balance by the amount of penalty)
// ejects the card
// 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 #