Exam #3

Monday, April 29

5:00 - 7:00 pm

closed books and notes exam

the coverage is posted.

This is a comprehensive exam.

Project #2 description is posted.

deadline: Thursday, April 25

COVERAGE FOR EXAM #3

CS 586; Spring 2024

Exam #3 will be held on **Monday, April 29, 2024,** between **5:00-7:00p.m.**
Location: 104 Stuart Building
The exam is a **CLOSED books and notes** exam.

Coverage for the exam:

- OO design patterns: item description, whole-part, observer, state, proxy, adapter, strategy and abstract factory patterns. [Textbook: Sections 3.1, 3.2; Section 3.4 (pp. 263-275); Section 3.6 (pp.339-343); Handout #1, class notes]
- Interactive systems. Model-View-Controller architectural pattern [Textbook: Section 2.4, pp. 123-143]
- Client-Server Architecture
  - Client-Dispatcher-Server [Section 3.6: pp. 323-337]
  - Client-Broker-Server Architecture [Textbook: Section 2.3; pp. 99-122]
- Layered architecture [Textbook: Section 2.2; pp. 31-51]
- Pipe and Filter architecture [Textbook: Section 2.2; pp. 53-70]
- Adaptable Systems:
  - Micro-kernel architectural pattern [Textbook: Section 2.5, pp. 169-192]
- Fault-tolerant architecture [Handout #2]
  - N-version architecture
  - Recovery-Block architecture
  - N-Self Checking architecture
- Repository architecture [Textbook: Section 2.2; pp. 71-95]

Sources:

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

Project

OO Design +
implementation of the
Design
of GP components.
based on MDA-EFSM
from part #1.

you may use the posted
MDA-EFSM!!

PART #2: PROJECT DESIGN, IMPLEMENTATION, and REPORT

CS 586; Spring 2024

Final Project Deadline: **Thursday, April 25, 2024**
Late submissions: 50% off
After **April 30** the final project will not be accepted.

Submission: The project must be submitted on Blackboard. The hardcopy submissions will not be accepted.

This is an **individual** project, not a team project. Identical or similar submissions will be penalized.

DESIGN and IMPLEMENTATION

The goal of the second part of the project is to design two *Gas Pump (GP)* components using the Model-Driven Architecture (MDA) and then implement these *GP* components based on this design using the OO programming language. This OO-oriented design should be based on the MDA-EFSM (for both *GP* components) that was identified in the first part of the project. You may use your own MDA-EFSM (assuming that it was correct) or you can use the posted sample MDA-EFSM. In your design, you **MUST** use the following OO design patterns:

- state pattern
- strategy pattern
- abstract factory pattern

In the design, you need to provide the class diagram, in which the coupling between components should be minimized and the cohesion of components should be maximized (components with high cohesion and low coupling between components). In addition, a sequence diagram should be provided as described on the next page (Section 4 of the report).

After the design is completed, you need to implement the *GP* components based on your design using the OO programming language. In addition, the driver for the project to execute and test the correctness of the design and its implementation for the *GP* components must be implemented.

The Report and Deliverables

I: REPORT

The report **must** be submitted as one PDF-file (otherwise, a **10% penalty will be applied).**

1. MDA-EFSM model for the *GP* components
   a. A list of meta events for the MDA-EFSM
   b. A list of meta actions for the MDA-EFSM with their descriptions
   c. A state diagram of the MDA-EFSM
   d. Pseudo-code of all operations of Input Processors of Gas Pump*: GP-1* and *GP-2*

2. Class diagram(s) of the MDA of the *GP* components. In your design, you **MUST** use the following OO design patterns:
   a. State pattern
   b. Strategy pattern
   c. Abstract factory pattern

3. For each class in the class diagram(s), you should:
   a. Describe the purpose of the class, i.e., responsibilities.
   b. Describe the responsibility of each operation supported by each class.

no pseudo code

4. Dynamics. Provide two sequence diagrams for two Scenarios:
   a. Scenario-I should show how one liter of gas is disposed in the Gas Pump *GP-1* component, i.e., the following sequence of operations is issued:
   Activate(4), Start(), PayCredit(), Approved(), StartPump(), Pump(), StopPump()
   b. Scenario-II should show how one gallon of Premium gas is disposed in the Gas Pump *GP-2* component, i.e., the following sequence of operations is issued:
   Activate(4.2, 7.2, 5.3), Start(), PayCash(10), Premium(), StartPump(), PumpGallon(), PumpGallon(), Receipt()

II: Well-documented (commented) source code

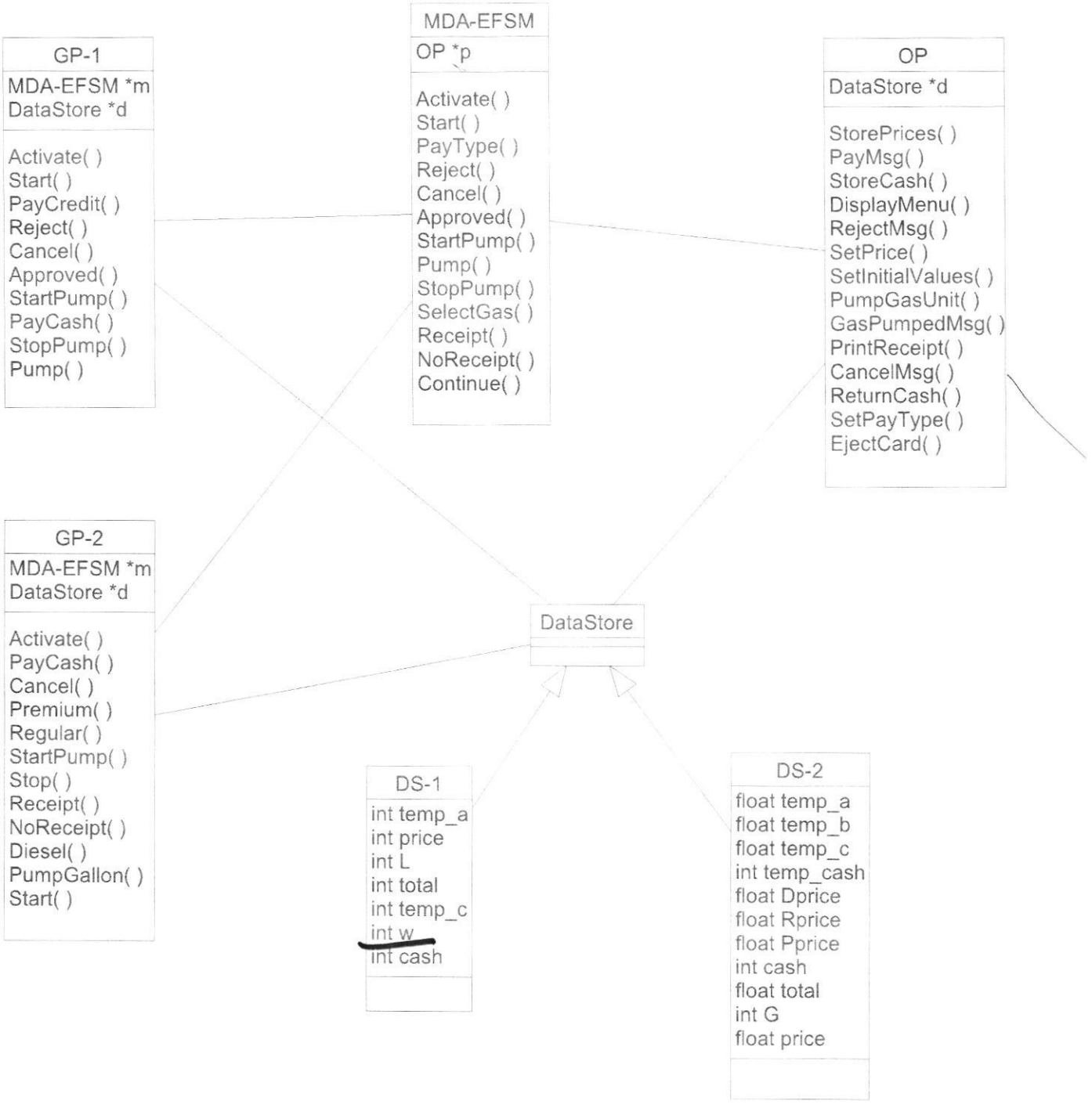
In the source-code you should indicate/highlight which parts of the source code are responsible for the implementation of the three required design patterns (**if this is not indicated in the source code, 20 points will be deducted**):

- state pattern
- strategy pattern
- abstract factory pattern.

The source-code must be submitted on the Blackboard. Note that the source code may be compiled during the grading and then executed. If the source-code is not provided, **15 POINTS** will be deducted.

III: Project executables

The project executable(s) of the *GP* components with detailed instructions explaining the execution of the program must be prepared and made available for grading. The project executable should be submitted on Blackboard. If the executable is not provided (or not easily available), **20 POINTS** will be automatically deducted from the project grade.

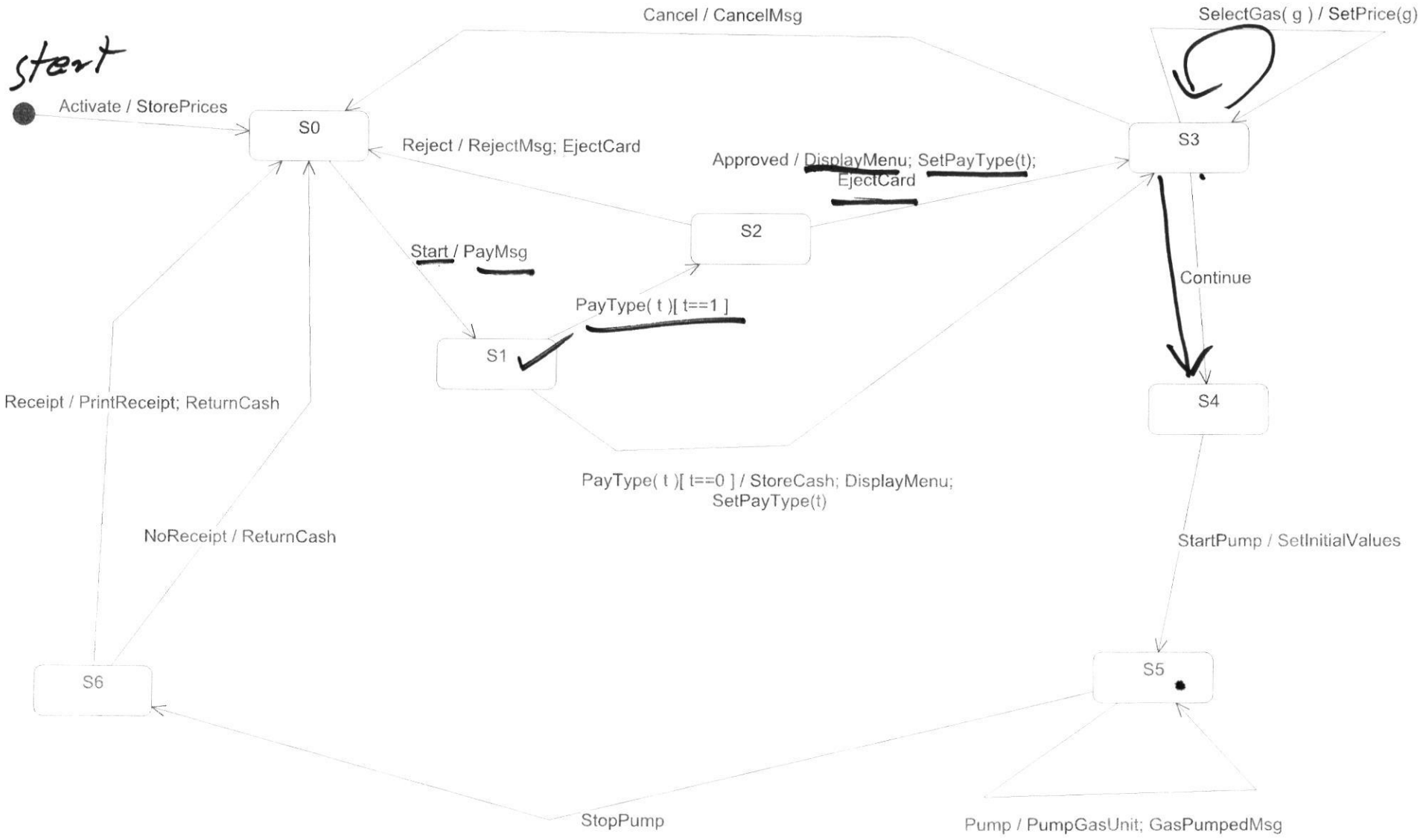

GP-1
MDA-EFSM *m
DataStore *d
Activate( )
Start( )
PayCredit( )
Reject( )
Cancel( )
Approved( )
StartPump( )
PayCash( )
StopPump( )
Pump( )
MDA-EFSM
OP *p
Activate( )
Start( )
PayType( )
Reject( )
Cancel( )
Approved( )
StartPump( )
Pump( )
StopPump( )
SelectGas( )
Receipt( )
NoReceipt( )
Continue( )
OP
DataStore *d
StorePrices( )
PayMsg( )
StoreCash( )
DisplayMenu( )
RejectMsg( )
SetPrice( )
SetInitialValues( )
PumpGasUnit( )
GasPumpedMsg( )
PrintReceipt( )
CancelMsg( )
ReturnCash( )
SetPayType( )
EjectCard( )
GP-2
MDA-EFSM *m
DataStore *d
Activate( )
PayCash( )
Cancel( )
Premium( )
Regular( )
StartPump( )
Stop( )
Receipt( )
NoReceipt( )
Diesel( )
PumpGallon( )
Start( )
DataStore
DS-1
int temp_a
int price
int L
int total
int temp_c
int w
int cash
DS-2
float temp_a
float temp_b
float temp_c
int temp_cash
float Dprice
float Rprice
float Pprice
int cash
float total
int G
float price

MDA-EFSM Events:

Activate()
Start()
PayType(int t) //credit: t=1; cash: t=0;
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g) // Regular: g=1; Diesel: g=2; Premium: g=3
Receipt()
NoReceipt()
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) // set the price for the gas identified by *g* identifier as in SelectGas(int g);
SetInitialValues() // set *G* (or *L*) and *total* to 0;
PumpGasUnit() // disposes unit of gas and counts # of units disposed and computes Total
GasPumpedMsg() // displays the amount of disposed gas
PrintReceipt() // print a receipt
CancelMsg() // displays a cancellation message
ReturnCash() // returns the remaining cash
SetPayType(t) // Stores pay type *t* to variable *w* in the data store
EjectCard() // Card is ejected



MDA-EFSM for Gas Pumps

Operations of the Input Processor (GasPump-1)

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

Start() {
        m->Start();
}
PayCash(int c) {
        if (c>0) {
            d->temp_c=c;
            m->PayType(0)
        }
}

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

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

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

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

Pump() {
if (d->w==1) m->Pump()
else if (d->cash < d->price*(d->L+1)) {
            m->StopPump();
            m->Receipt(); }
     else m->Pump()
}

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

Notice:
cash: contains the value of cash deposited
price: contains the price of the gas
L: contains the number of liters already pumped
w: pay type flag (cash: w=0; credit: w=1)
cash, *L*, *price*, *w:* are in the data store
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)

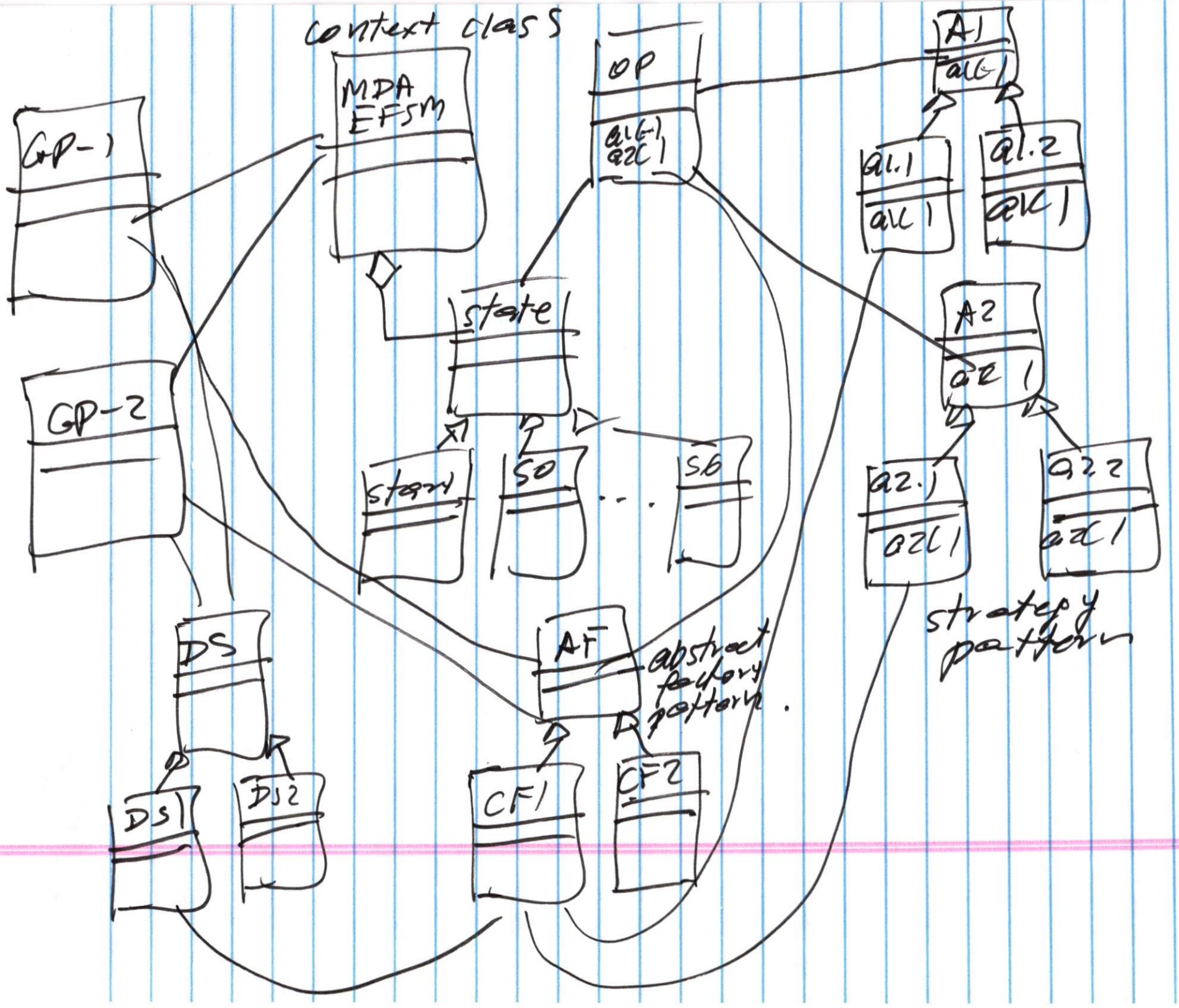
```
Activate(float a, float b, float c) {
        if ((a>0)&&(b>0)&&(c>0)) {
            d->temp_a=a;
            d->temp_b=b;
            d->temp_c=c
            m->Activate()
        }
}

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


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

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

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

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

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

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

PumpGallon() {
if (d->cash < d->price*(d->G+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
G: contains the number of Gallons already pumped

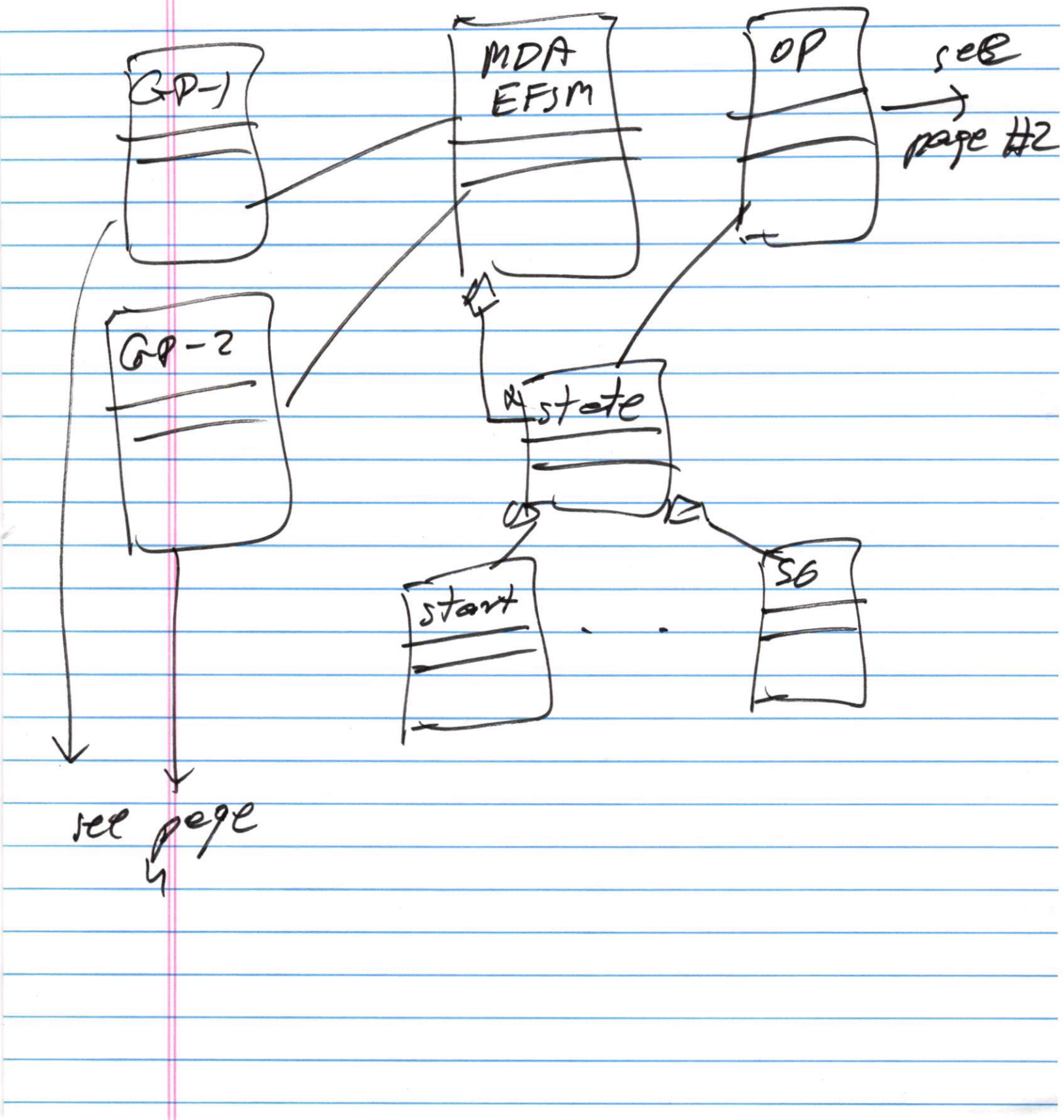
cash, *G*, *price* are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

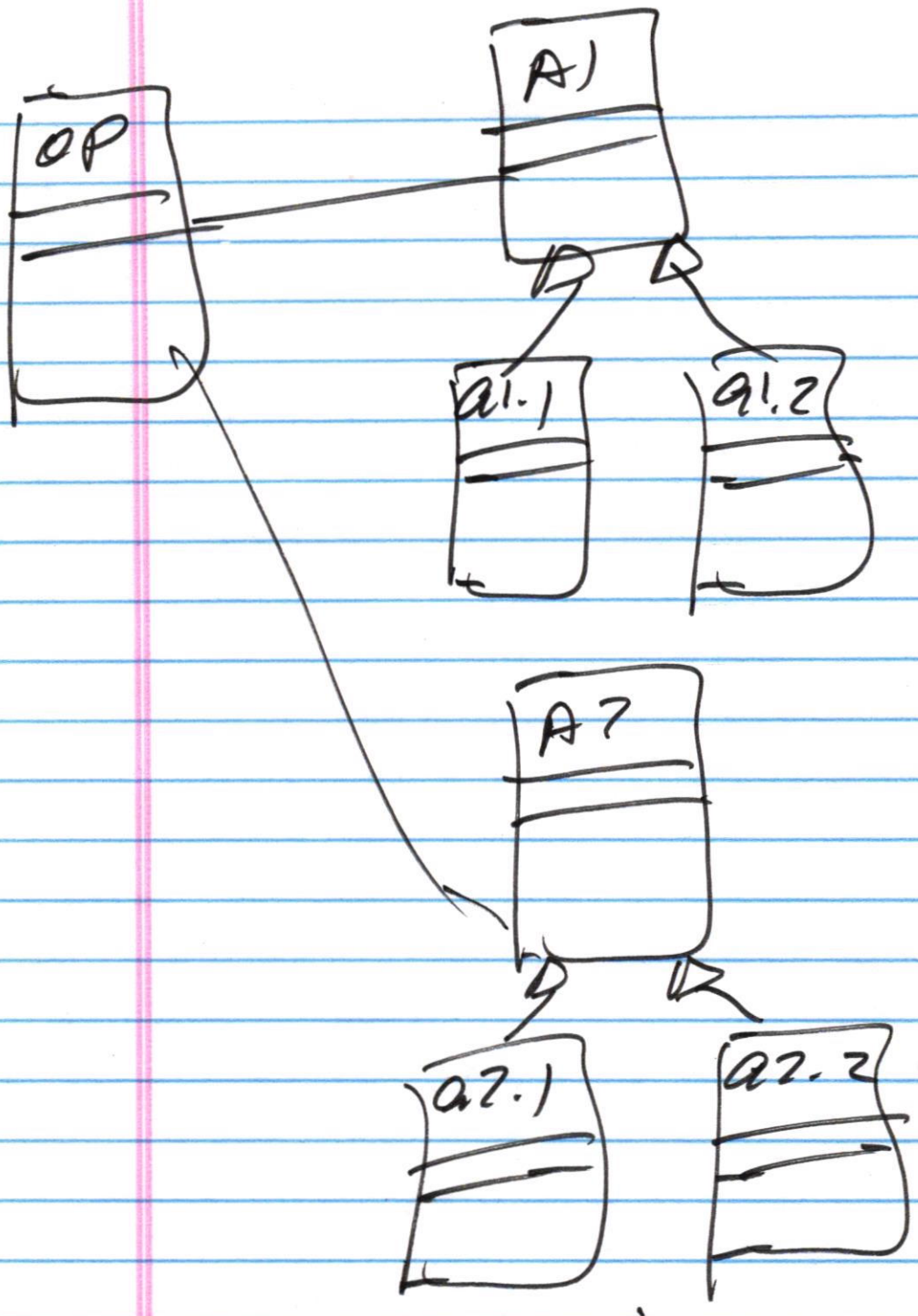
In the class diagram you need to incorporate.

* state pattern
* strategy —"—
* abstract factory pattern.

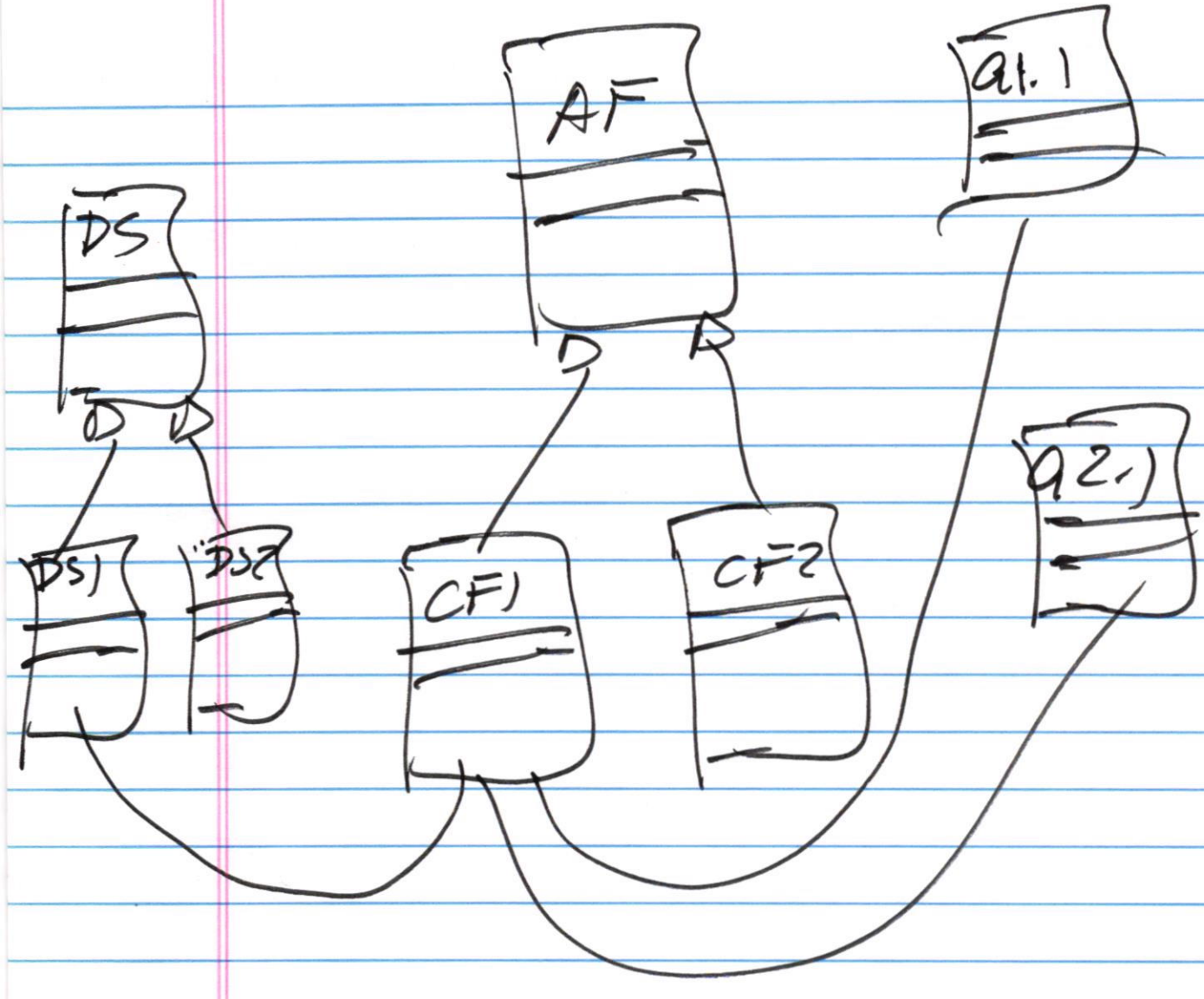

context class
MDA EFSM
GP-1
GP-2
OP
Q1(-1) Q2(-1
state
state-1
S0
S6
A1
Q1(-1)
Q1.1
Q1.2
A2
Q2(-1)
Q2.1
Q2.2
DS
DS1
DS2
AF
abstract factory pattern.
CF1
CF2
strategy pattern

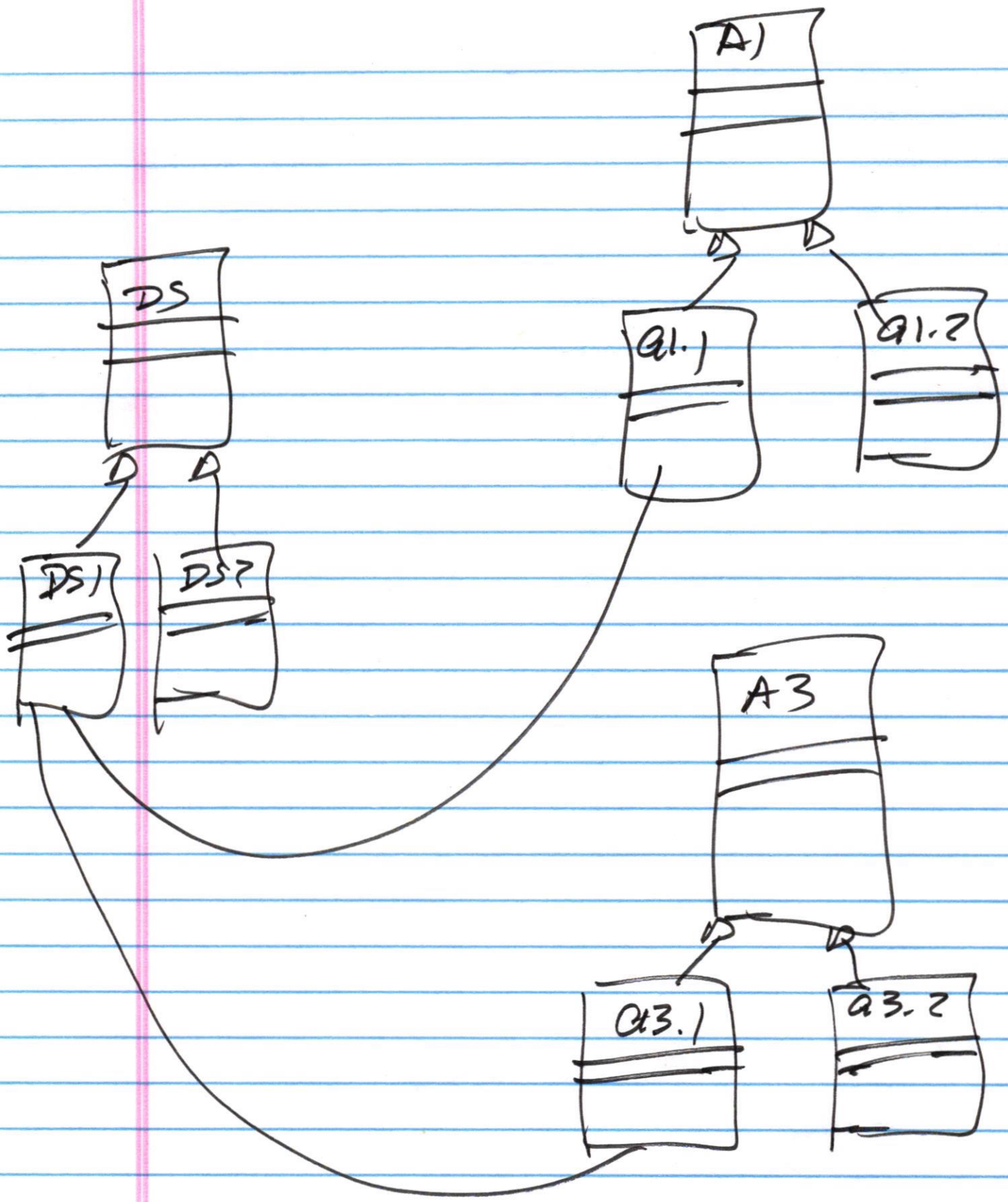
class diagram



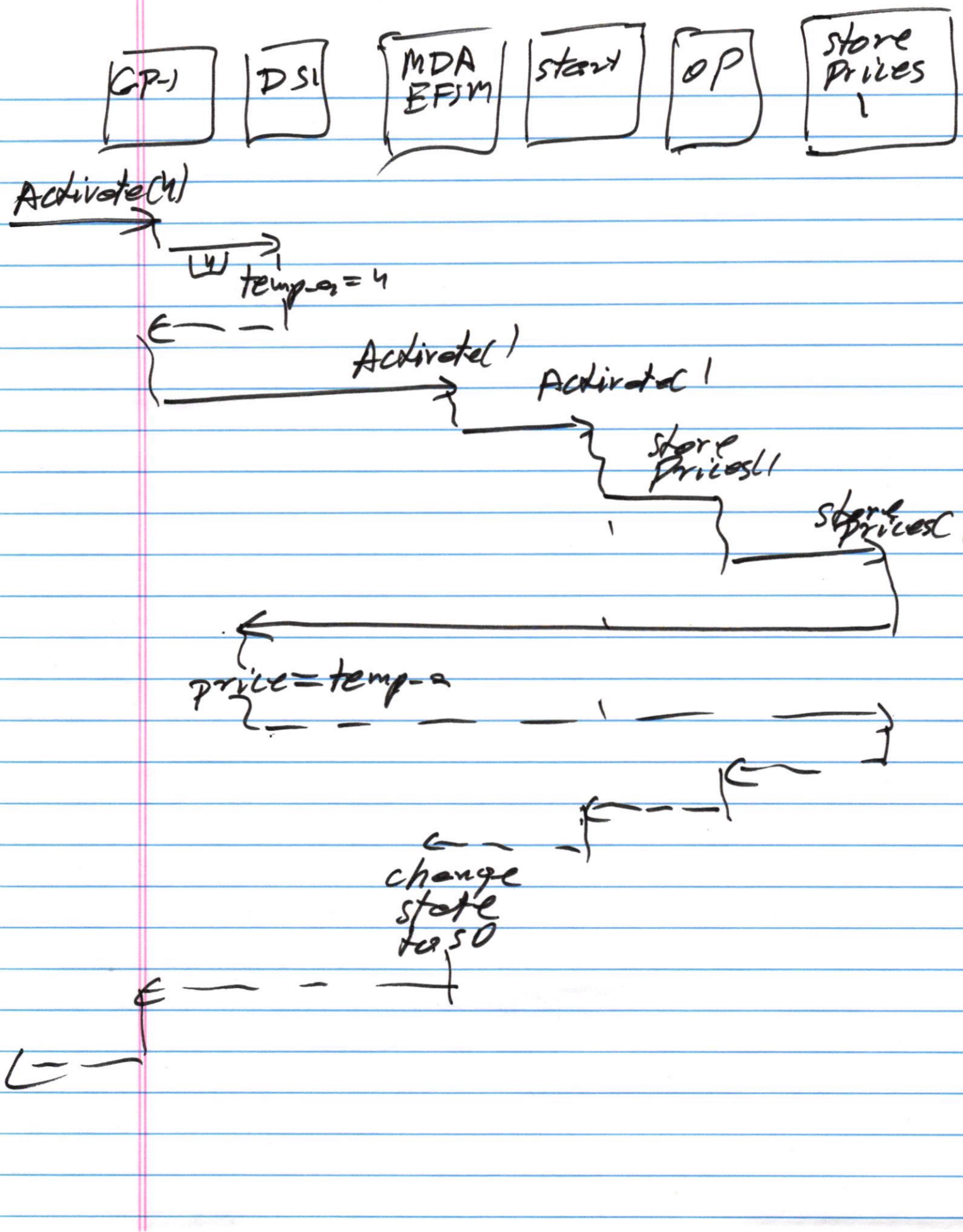


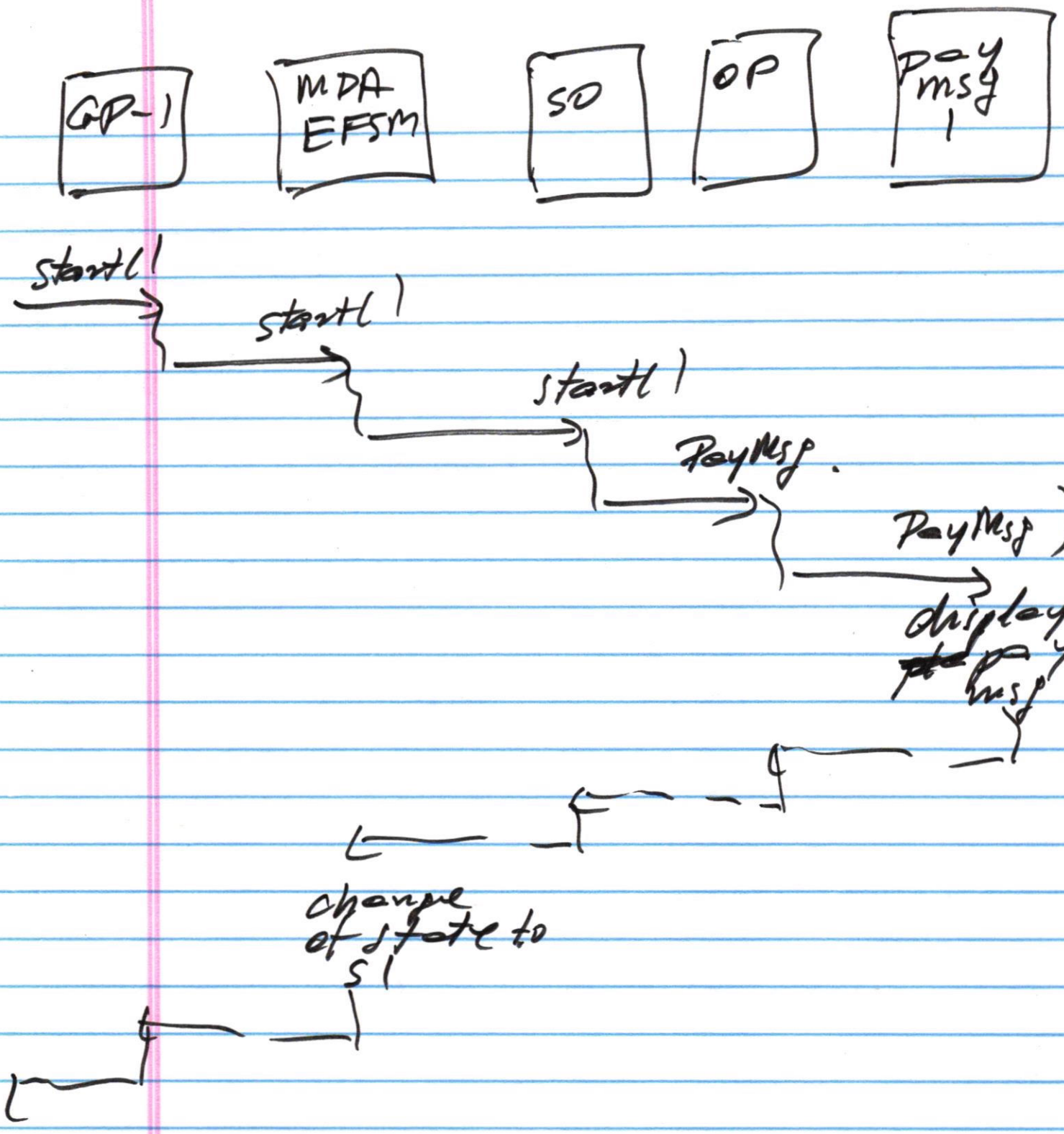
see page 3

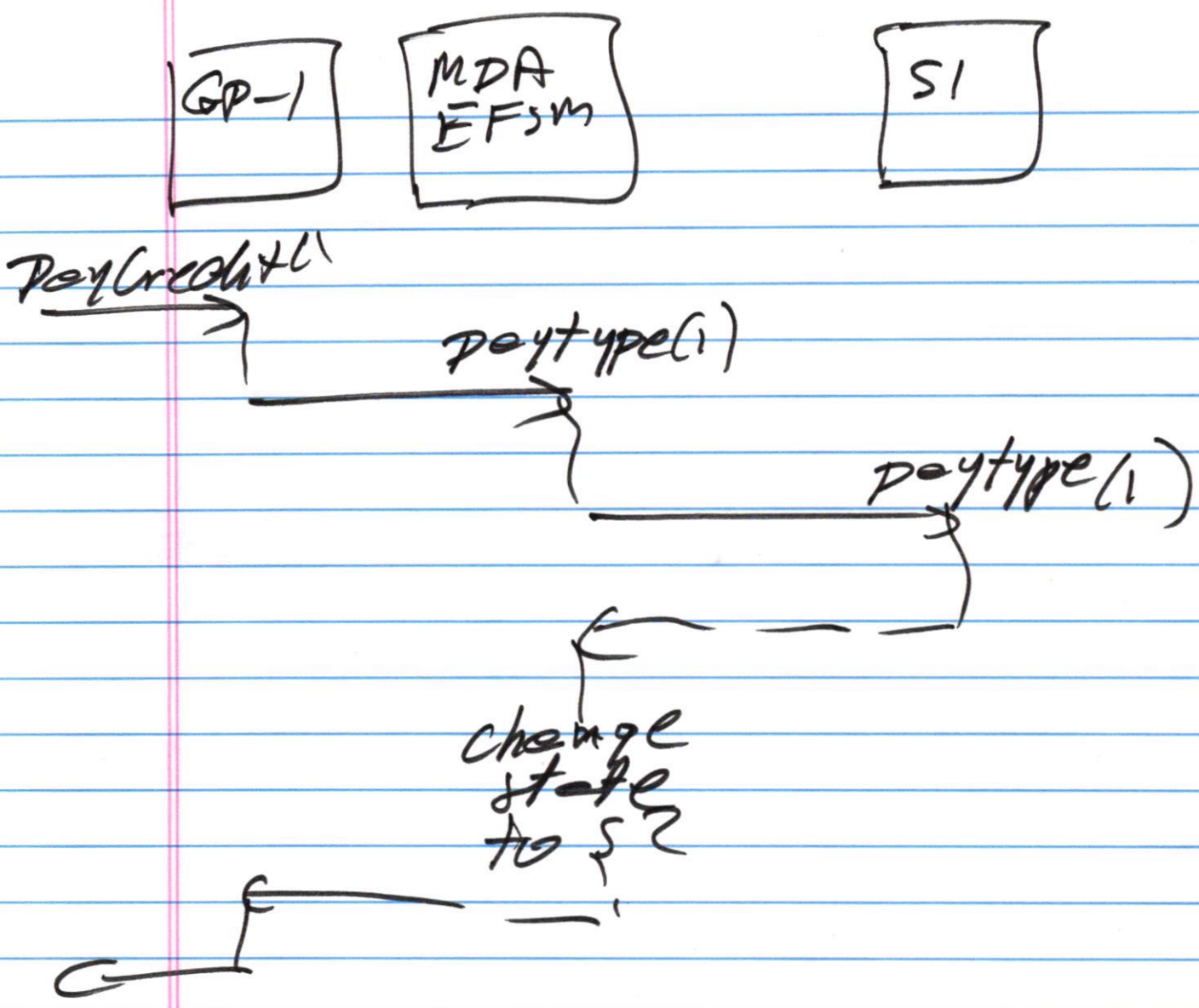

AF
q1.1
DS
DS1
DS2
CF1
CF2
q2.1

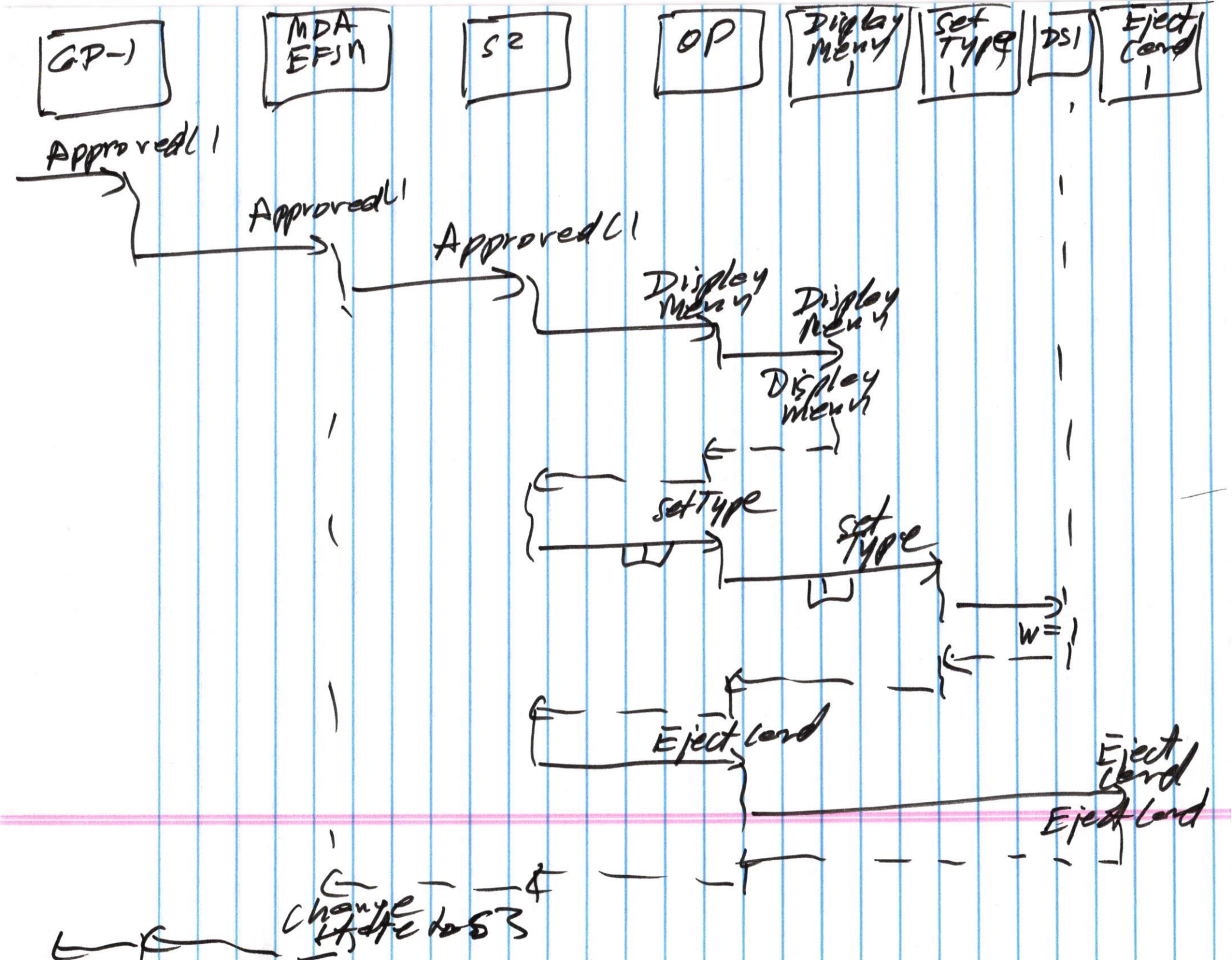

A1
DS
Q1.1
Q1.2
DS1
DS2
A3
Q3.1
Q3.2

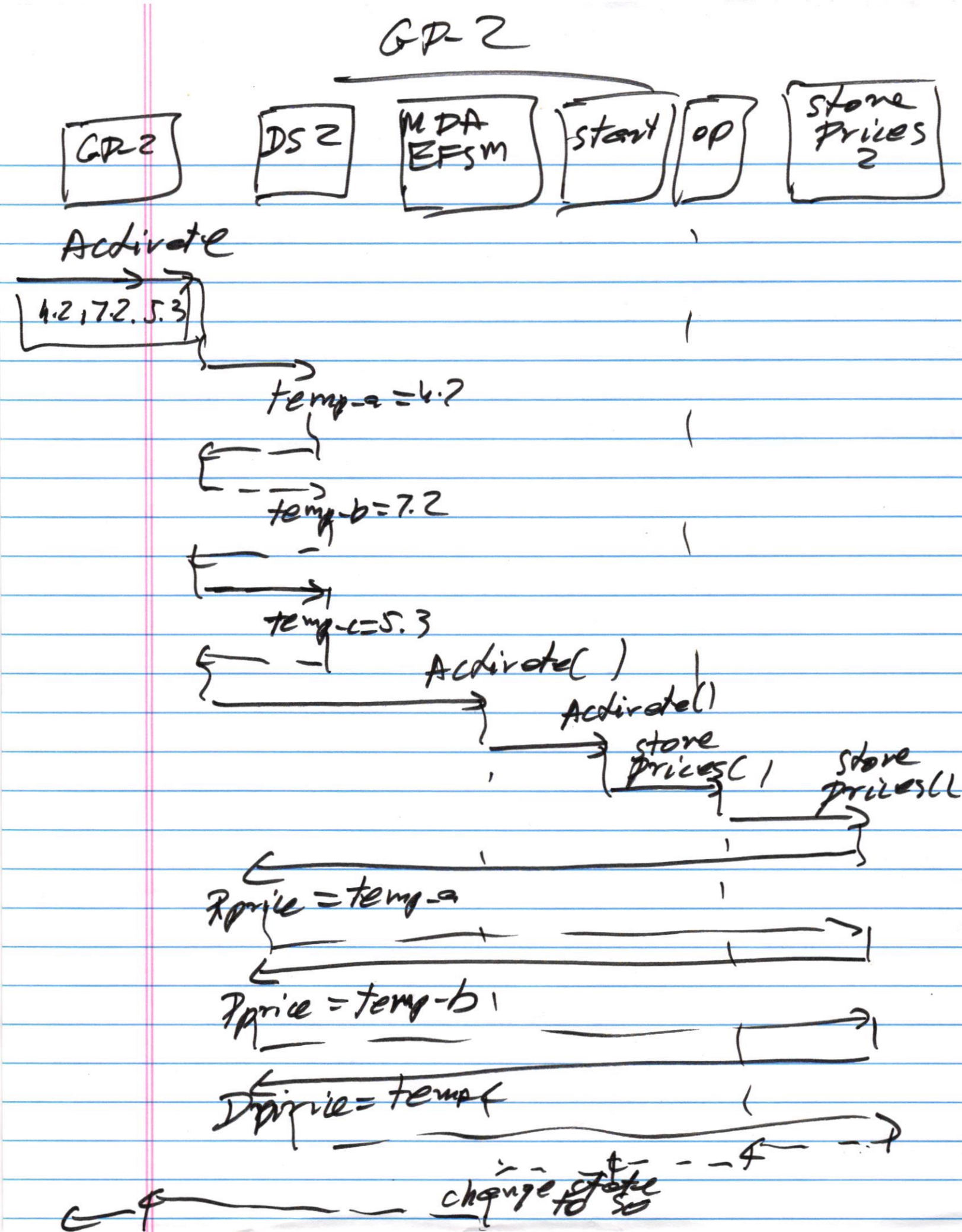
GP-1


GP-1
DSL
MDA EFSM
store
OP
Store Prices
Activate(4)
4
temp-a = 4
Activate(1
Activate(1
Store Prices(1
Store Prices(1
price = temp-a
change state to S0


GP-1
MDA EFSM
SD
OP
Pay msg 1
start()
start()
start()
PayMsg.
PayMsg)
display Pay msg.
change of state to S1


GP-1
MDA EFSM
S1
PayCredit()
paytype(1)
paytype(1)
change state to S2


GP-1
MDA EFSM
S2
OP
Display Menu 1
Set Type 1
DS1
Eject Card 1
Approved(1)
Approved(1)
Approved(1)
Display Menu
Display Menu
Display Menu
setType
Set Type
w=1
Eject Card
Eject Card
Eject Card
Change Fields


GP 2
GP 2
DS 2
MDA EFSM
start
op
Store Prices 2
Activate
4.2, 7.2, 5.3
temp_a = 4.2
temp_b = 7.2
temp_c = 5.3
Activate( )
Activate()
store prices( )
store prices( )
Rprice = temp_a
Rprice = temp_b
Dprice = temp_c
change state to so

Implementation of the design

Driver/main.

should be able to invoke.
any operation at any time!!

CS586 Project
Spring 2024

Sample Test Cases:

Gas Pump GP-1

Test #1
Activate(4), Start(), PayCredit(), Approved(), StartPump(), Pump(), StopPump()
Expected result at the last operation: Total=$4

Test #2
Activate(5), Start(), PayCredit(), Approved(), Activate(8), StartPump(), Pump(), Pump(), StopPump()
Expected result at the last operation: Total=$10

Test #3
Activate(5), Start(), PayCash(8), StartPump(), Cancel(), Pump(), Pump()
Expected result at the last operation: Total=$5; Returned cash=$3

Gas Pump GP-2

Test #1
Activate(4.2, 7.2, 5.3), Start(), PayCash(10), Premium(), StartPump(), PumpGallon(), PumpGallon(), Receipt()
Expected result at the last operation: Total=$7.2; Returned cash=$2.8

Test #2
Activate(4, 7, 5), Start(), PayCash(10), Premium(), Diesel(), StartPump(), PumpGallon(), Stop(), Receipt()
Expected result at the last operation: Total=$7; Returned cash=$3

Blackboard Architecture

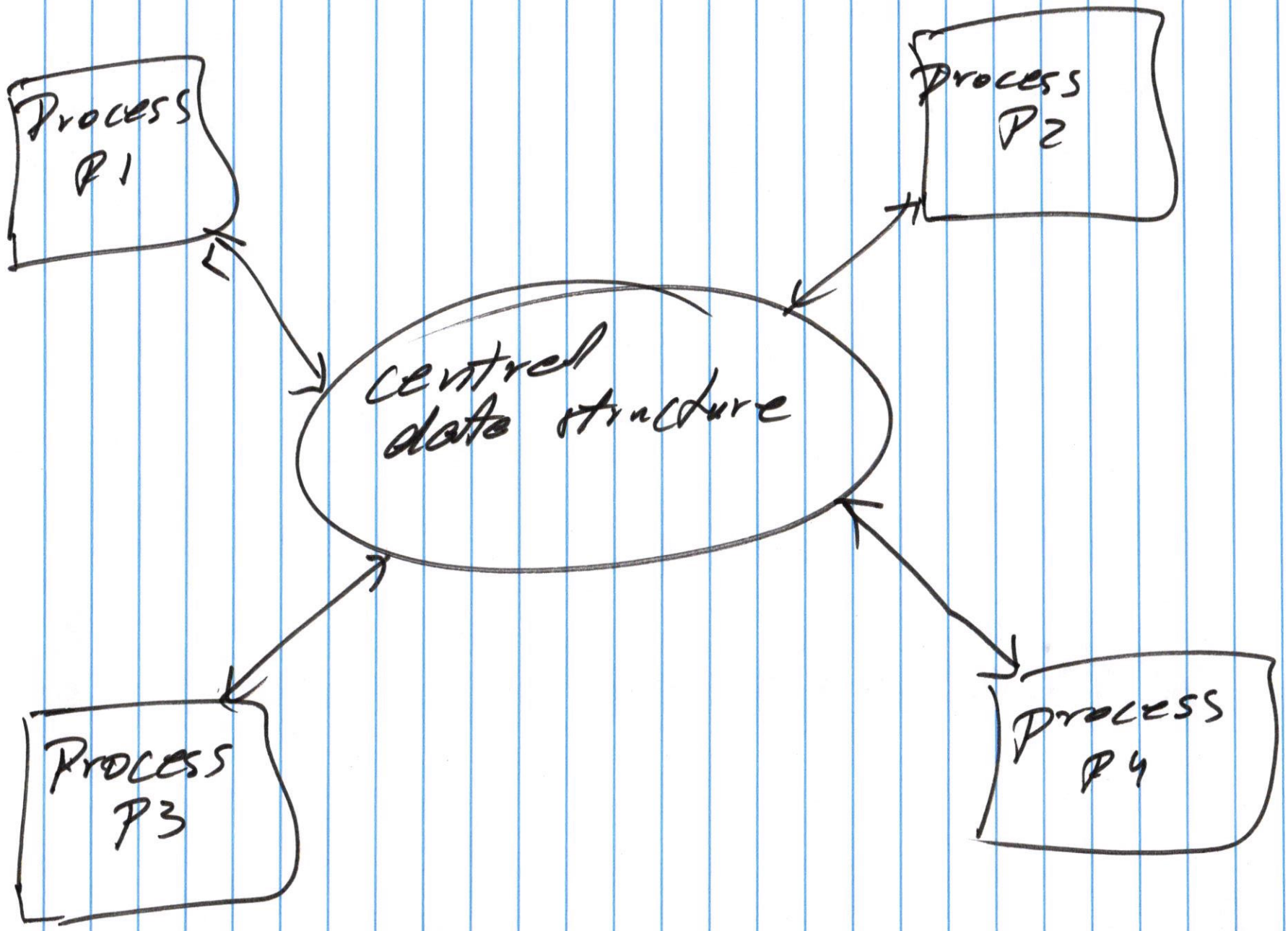
it is useful for problems in which no deterministic algorithm/solution is know

1. speech recognition
2. expert systems

Repository Architecture

* central data structure (data base)

* a set of processes that operate on this central data structure.


Process P1
Process P2
central data structure
Process P3
Process P4

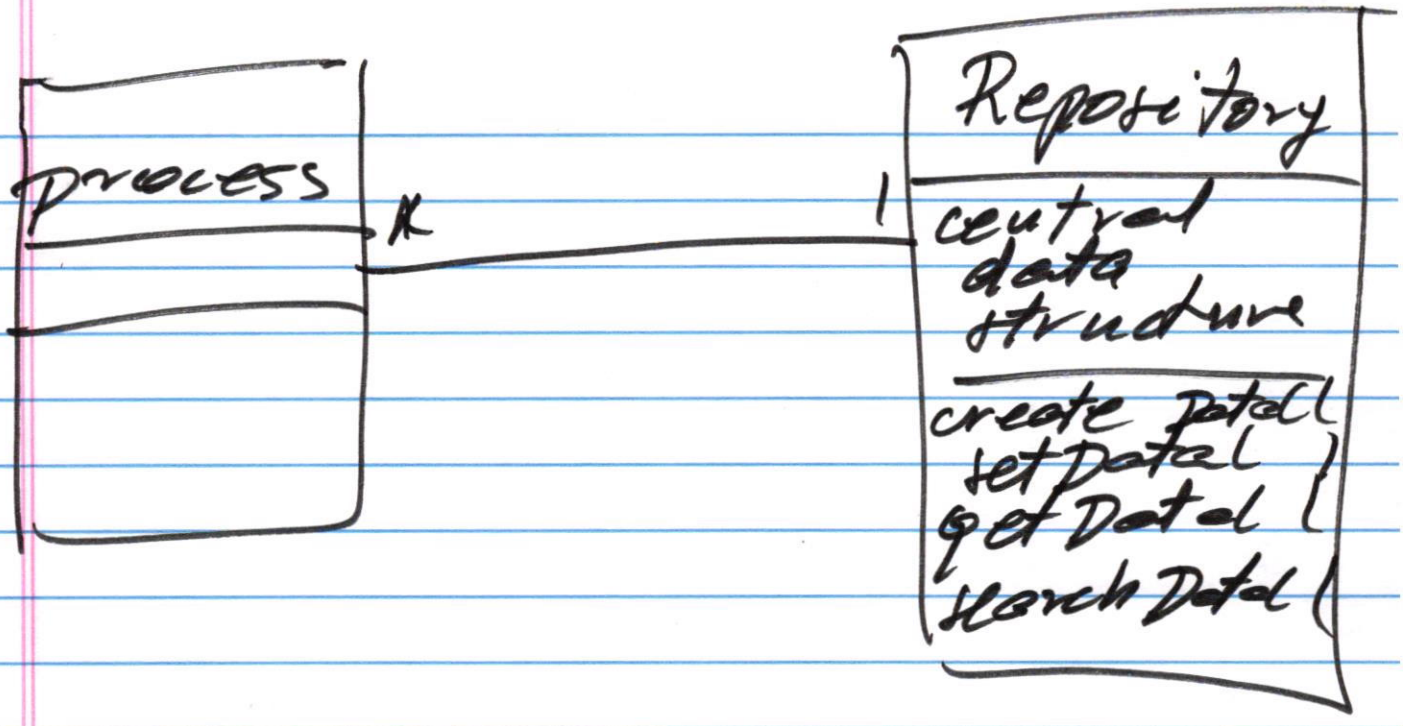
selection of processes for execution

* Transition Type -
  input: stream of transactions.

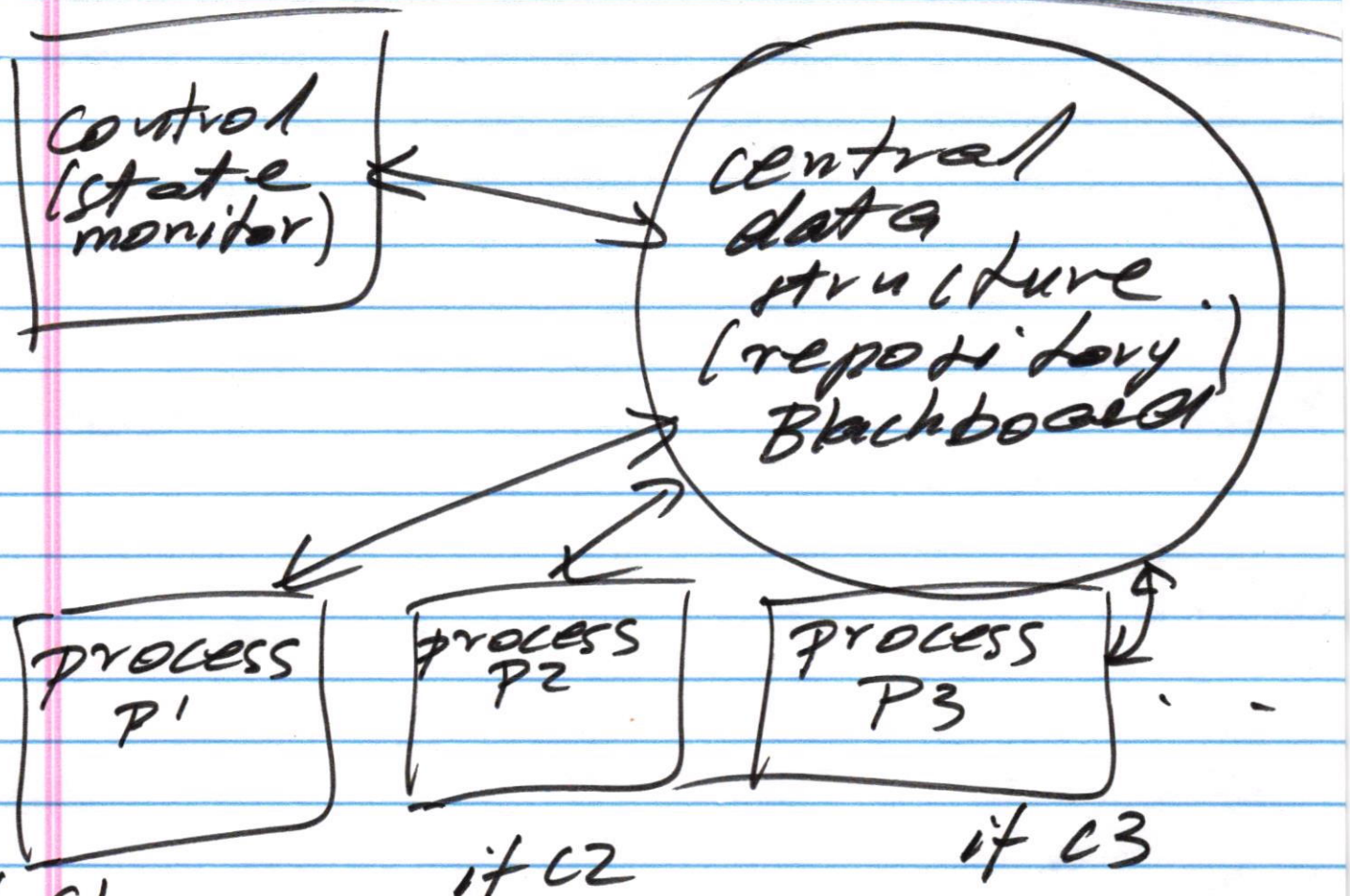
Transaction type may be used to select a process for execution

↓

traditional data base.


Process
K
1
Repository
central data structure
createDatal()
setDatal()
getDatal()
searchDatal()

Blackboard Architecture



1. P1 and P3 conditions evaluate to true
   P1 is selected randomly to execute
2. P2 and P4 condition evaluate to true
   P2 is selected randomly for execution
3. . .

4. Blackboard

1. central data structure.

2. Processes solve some specific subtasks.

   Process has 2 elements.

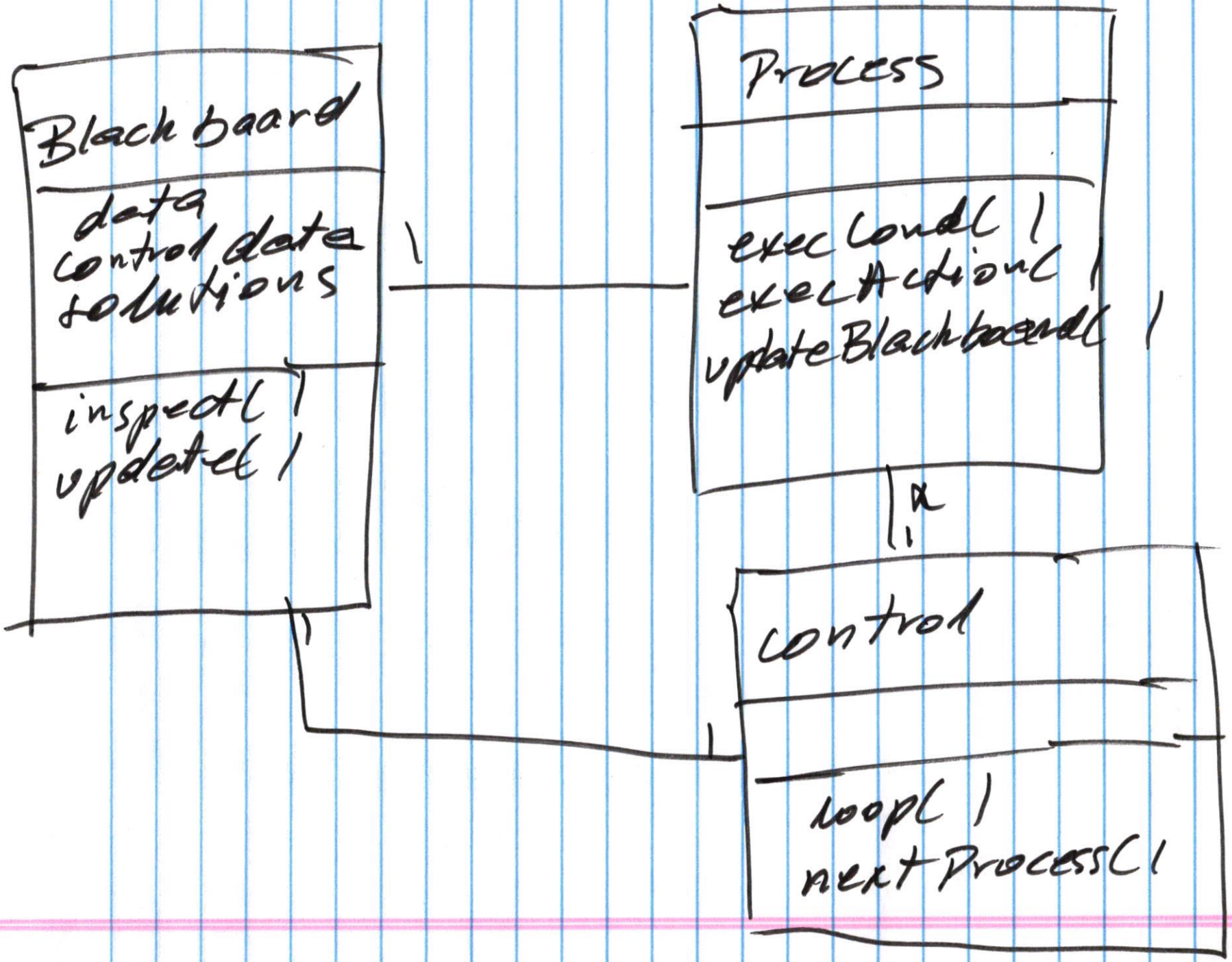
   (a) condition

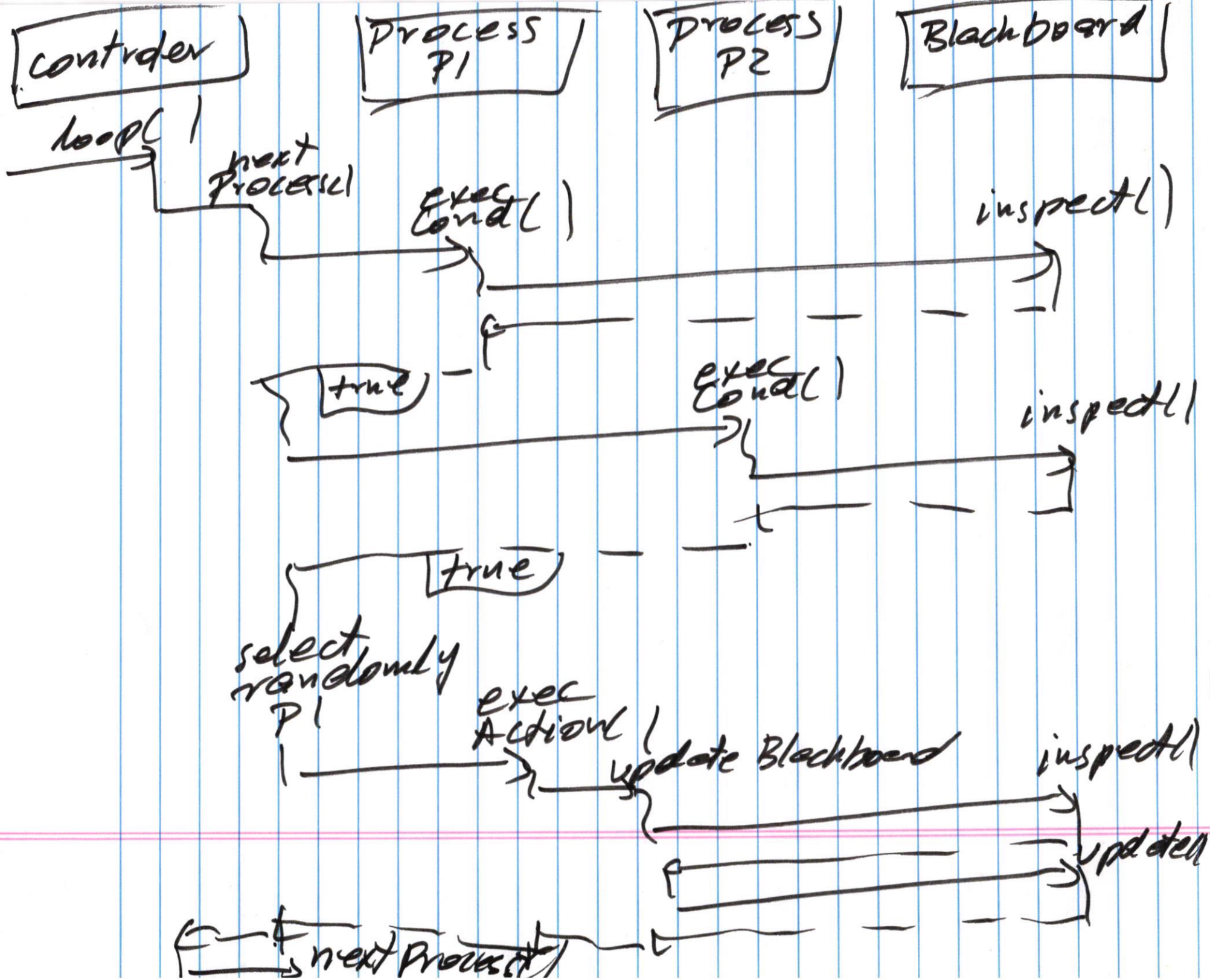
   (b) action.

3. Control selects processes for execution.

   (a) check conditions of processes.

   (b) select randomly which process to execute among processes that evaluate to "true"


Blackboard
data
control data
solutions
inspect()
update()
Process
execCond()
execAction()
updateBlackboard()
control
loop()
nextProcess()


controler
Process P1
Process P2
Blackboard
loop( )
next Process( )
execCond( )
inspect( )
true
execCond( )
inspect( )
true
select randomly P
execAction( )
update Blackboard
inspect( )
updated
next Process( )

Adv

* it is useful - when no deterministic approach is known.

Disadv

* maintenance can be very hard.
* repository is a bottleneck.

↓

any small change to repository

↓

many changes & in processes