SYSTEM SOFTWARE ARCHITECTURE(CS-586)
PROJECT FINAL PHASE
SPRING 2019

1. MDA-EFSM model for the Vending Machine Components.

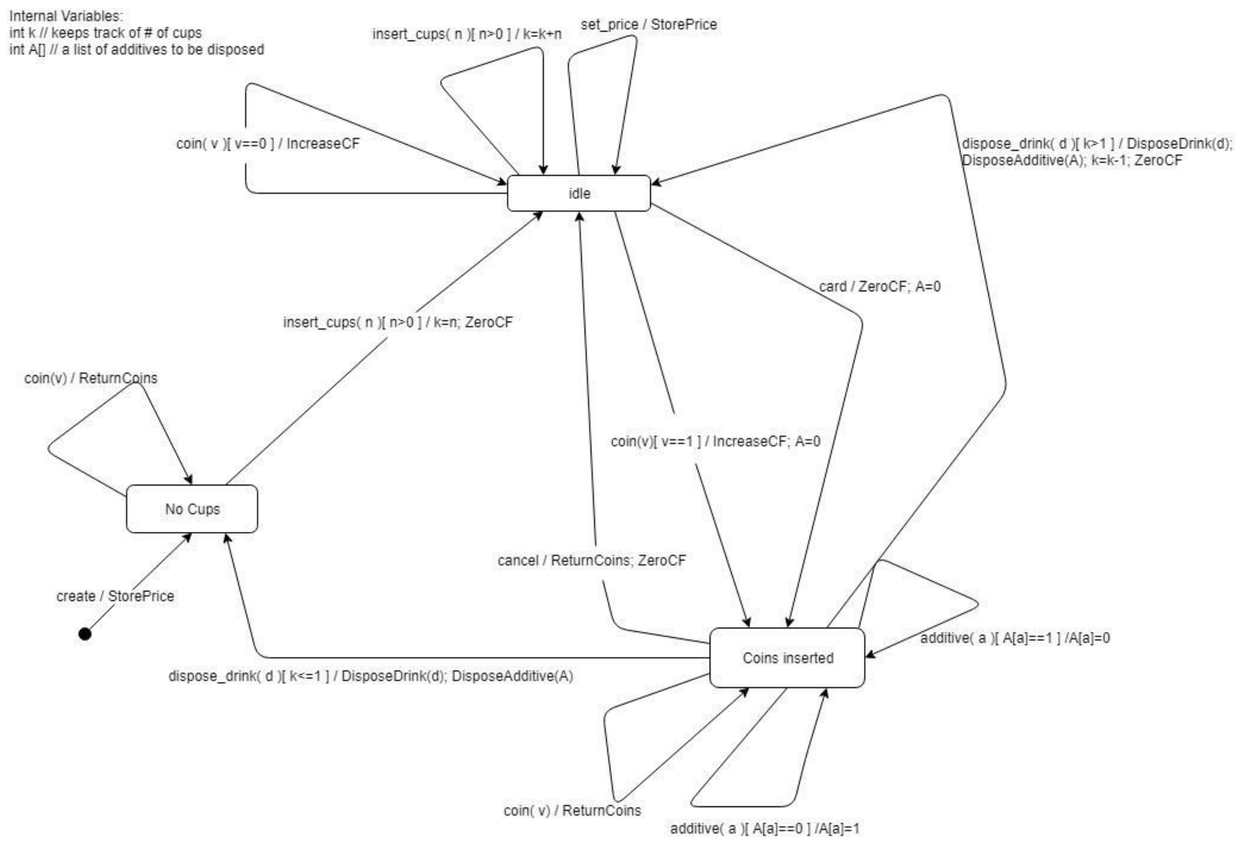
a) MDA-EFSM Events:

1. create()	
2. insert_cups(int n)	// n represents number of cups
3. coin(int v)	// v=1: sufficient funds inserted for a drink
	// v=0: In-sufficient funds for a drink
4. card()	
5. cancel()	
6. set_price()	
7. dispose_drink(int d)	// d represents a drink id
8. additive(int a)	// a represents additive id

b) MDA-EFSM Actions:

1. StorePrice()	//store the initial price for the drink
2. ZeroCF()	// zero Cumulative Fund cp
3. IncreaseCF()	// increase Cumulative Fund cp
4. ReturnCoins()	// return coins inserted for a drink
5. DisposeDrink(int d)	// dispose a drink with d id
6. DisposeAdditive(int A[])	//dispose marked additives in A list,
	// where additive with i id is disposed when A[i]=1

c) STATE DIAGRAM(MDA-EFSM)


Internal Variables:
int k // keeps track of # of cups
int A[] // a list of additives to be disposed
insert_cups(n)[n>0] / k=k+n
set_price / StorePrice
coin(v)[v==0] / IncreaseCF
idle
dispose_drink(d)[k>1] / DisposeDrink(d); DisposeAdditive(A); k=k-1; ZeroCF
card / ZeroCF; A=0
insert_cups(n)[n>0] / k=n; ZeroCF
coin(v) / ReturnCoins
coin(v)[v==1] / IncreaseCF; A=0
No Cups
cancel / ReturnCoins; ZeroCF
create / StorePrice
additive(a)[A[a]==1] /A[a]=0
Coins inserted
dispose_drink(d)[k<=1] / DisposeDrink(d); DisposeAdditive(A)
coin(v) / ReturnCoins
additive(a)[A[a]==0] /A[a]=1

d) PSEUDOCODE:

Operations of the Input Processor: Vending Machine-1

Note:

s: pointer to the MDA-EFSM
d: pointer to the data store DS-1
In the data store:
cp: represents a cumulative fund
price: represents a price for a drink

```
create(int p) {
d->p=p;
s->create(); }

coin(int v) {
d->v=v;
if (d->cp+v>=d->price)
s->coin(1);
else
 s->coin(0); }

card(float x) {
if (x>=d->price)
s->card(); }

sugar() {
s->additive(1); }

tea() {
s->dispose_drink(1); }

chocolate() {
s->dispose_drink(2); }

insert_cups(int n) {
s->insert_cups(n); }

set_price(int p) {
d->p=p;
s->set_price() }

 cancel() {
s->cancel(); }
```

Operations of the Input Processor: <u>Vending Machine-2</u>

<u>Note:</u>
s: pointer to the MDA-EFSM
d: pointer to the data store DS-2
In the data store:
cp: represents a cumulative fund
price: represents a price for a drink

```
CREATE(float p) {
d->p=p;
s->create(); }

COIN(float v) {
d->v=v;
if (d->cp+v>=d->price)
s->coin(1);
else
s->coin(0); }

SUGAR() {
s->additive(1); }

CREAM() {
s->additive(2); }

COFFEE() {
s->dispose_drink(1); }

InsertCups(int n) {
s->insert_cups(n); }

SetPrice(float p) {
d->p=p;
s->set_price() }

CANCEL() {
s->cancel();}
```

2. Class diagram of the MDA of the Vending Machine components.

-Attached a JPG file along due to the large picture of class diagram.
-State Class is the MDA-EFSM class.
-StateMachine Class is the State of MDA_EFSM while using State design Pattern.

3.a) The purpose of each class, i.e., responsibilities.

Class VendingMachine1:
This class is an input processor for Vending Machine 1. It checks for initial conditions and calls corresponding methods from MDA-EFSM.

Class VendingMachine2:
This class is an input processor for Vending Machine 2. It checks for initial conditions and calls corresponding methods from MDA-EFSM.

Class State: (MDA-EFSM class)
Here this is the MDA-EFSM in Decentralized state design pattern
Here state classes are responsible for performing: Actions and State transitions.

Class StateMachine: (State class of MDA-EFSM for State design pattern)

This class is the abstract State superclass in the De-centralized State Design Pattern. Normally, each operation defined in this class should be abstract. However, it was decided to instead default each operation to print "not allowed" message to save coding and memory space. In this design, the methods that do not get overridden will print a "Not Allowed" message if they are called from a state that does not allow them to be performed.

Class Start:
Initial State in the EFSM model. The actions and state transitions are done for the methods belonging to this class accordingly.

Class NoCups:
State NoCups in the EFSM model. The actions and state transitions are done for the methods belonging to this class accordingly.

Class Idle:
State Idle in the EFSM model. The actions and state transitions are done for the methods belonging to this class accordingly.

Class Coins_Inserted:
State Coins_Inserted in the EFSM model. The actions and state transitions are done for the methods belonging to this class accordingly.

Class OutputProcessor:
This class acts as the "Client" class in the strategy design pattern. This class is the output processor for the vending machine system. It must be initialized with the proper action implementations for the specific vending machine chosen. This is done thorough the Abstract Factory design pattern.

Class AbstractFactory:
This class groups all Concrete Factory classes in 1 abstract superclass. It defines the methods that return the Vending Machine specific action event components which all Concrete Factories need to implement

Class ConcreteVending1:
This class is the factory that produces the necessary driver objects for VendingMachine1. OutputProcessor object will be instantiated with an object of this class when it needs to display output for VendingMachine1. Output processor will call the methods provided by this class to bind VendingMachine1 specific actions. Instantiates the action strategies with the shared data structure.

Class ConcreteVending2:
This class is the factory that produces the necessary driver objects for VendingMachine2. OutputProcessor object will be instantiated with an object of this class when it needs to display output for VendingMachine2. Output processor will call the methods provided by this class to

bind VendingMachine2 specific actions. Instantiates the action strategies with the shared data structure.

Class DataStore:
This class groups all Data classes in 1 abstract superclass which are used as data structure for each Vending Machine component.

Class DS1:
Vending Machine1 data storage object for storing data that must be shared between system components. Instead of getters and setters method, we have accessed the fields directly.

Class DS2:
Vending Machine2 data storage object for storing data that must be shared between system components. Instead of getters and setters method, we have accessed the fields directly.

Class StorePrice:
Abstract StorePrice action strategy. Groups all "Store Price" actions under 1 abstract superclass.

Class StorePrice1:
Vending Machine 1 StorePrice action responsible for storing the price parameter specified by method create() of the InputProcessor for Vending Machine1.

Class StorePrice2:
Vending Machine 2 StorePrice action responsible for storing the price parameter specified by method create() of the InputProcessor for Vending Machine2.

Class ZeroCF:
Abstract ZeroCF action strategy. Groups all "ZeroCF" actions under 1 abstract superclass.

Class ZeroCF1:
Vending Machine1 ZeroCF action responsible for setting current funds to 0.

Class ZeroCF2:
Vending Machine2 ZeroCF action responsible for setting current funds to 0.

Class IncreaseCF:
Abstract IncreaseCF action strategy. Groups all "IncreaseCF" actions under 1 abstract superclass.

Class IncreaseCF1:
Vending Machine 1 IncreaseCF action responsible adding up extra coins added.

Class IncreaseCF2:
Vending Machine 2 IncreaseCF action responsible adding up extra coins added.

Class ReturnCoins:
Abstract ReturnCoins action strategy. Groups all "ReturnCoins" actions under 1 abstract superclass.

Class ReturnCoins1:
Vending Machine 1 ReturnCoins action responsible for returning the remaining coins.

Class ReturnCoins2:
Vending Machine 2 ReturnCoins action responsible for returning the remaining coins.

Class DisposeDrink:
Abstract DisposeDrink() action strategy. Groups all "DisposeDrink" actions under 1 abstract superclass.

Class DisposeDrink1:
Vending machine1 DisposeDrink action responsible for disposing the drink selected from method tea() or chocolate() of the InputProcessor for Vending Machine 1.

Class DisposeDrink2:
Vending machine2 DisposeDrink action responsible for disposing the drink selected from method COFFEE() of the InputProcessor for Vending Machine 2.

Class DisposeAdditive:
Abstract DisposeAdditive action strategy.Groups all "Dispose Additive" actions under 1 abstract superclass strategy. Each Vending machine gets its own DisposeAdditive action class to allow for easy modification in the future without having to program new classes.

Class DisposeAdditive1:
Vending Machine 1 DisposeAdditive action responsible for Disposing an additive.

Class DisposeAdditive2:
Vending Machine 2 DisposeAdditive action responsible for Disposing an additive(s).

3.b) The responsibility of each operation supported by each class.

Class VendingMachine1:

create(int p):
Checks the input parameters for correctness and call the create() from the MDA-EFSM. Also check for wrong input. p= price of drink v=value of the coin x=value of card n= number of cups.

coin(int v):
Calls coin(v) method from MDA-EFSM. v=1 means sufficient funds inserted according to MDA-EFSM , v=0 means Insufficient funds according to MDA-EFSM.

card(float x):
Calls card() from MDA-EFSM. x=value in card.

sugar():
Calls additive(int a) from MDA-EFSM . additive(1) selects the additive sugar for the drink also print the selected additive message to confirm the choice.

tea():
Calls dispose_drink(int d) from MDA-EFSM. dispose_drink(1) is a function with value passed as 1. This selects the drink type as tea Also print the selected drink confirmation message.

chocolate():
Calls dispose_drink(int d) from MDA-EFSM . dispose_drink(2) is a function with value passed as 2. This selects the drink type as Chocolate and print the selected drink confirmation message.

insert_cups(int n):
Calls insert_cups(int n) from MDA-EFSM. n= number of cups print the number of cups inserted.

set_price(int p):
Calls set_price() from MDA-EFSM. p is the price to be set.

cancel():
Calls cancel() from MDA-EFSM. Prints the cancelled msg.

Class VendingMachine2:

CREATE(float p):
Checks the input parameters for correctness and call the create() from the MDA-EFSM. Also check for wrong input. p= price of drink, v=value of the coin, n= number of cups.

COIN(float v):
Calls coin(v) method from MDA-EFSM. v=1 means sufficient funds inserted according to MDA-EFSM , v=0 means Insufficient funds according to MDA-EFSM.

SUGAR():
Calls additive(int a) from MDA-EFSM . additive(1) selects the additive sugar for the drink also print the selected additive message to confirm the choice.

CREAM():
Calls additive(int a) from MDA-EFSM . additive(2) selects the additive CREAM for the drink also print the selected additive message to confirm the choice.

COFFEE():
Calls dispose_drink(int d) from MDA-EFSM. dispose_drink(1) is a function with value passed as 1. This selects the drink type as Coffee. Also print the selected drink confirmation message.

InsertCups():
Calls insert_cups(int n) from MDA-EFSM. n= number of cups print the number of cups inserted.

SetPrice():
Calls set_price() from MDA-EFSM. p is the price to be set.

CANCEL():
Calls cancel() from MDA-EFSM. Prints the cancelled msg.

Class State: (MDA-EFSM class)
The states are initialized with id to recognize them. All the operations(create(), insert_cups(int n), coin(int v), card(), cancel(), set_price(), dispose_drink(int d), additive(int a)) called are forwarded to the StateMachine Class(State class for the MDA-EFSM).

Class StateMachine: (State class for the MDA-EFSM)
Each operation(create(), insert_cups(int n), coin(int v), card(), cancel(), set_price(), dispose_drink(int d), additive(int a)) defined in this class should be abstract. However, it was decided to instead default each operation to print "not allowed" message to save coding and memory space. In this design, the methods that do not get overridden will print a "Not Allowed" message if they are called from a state that does not allow them to be called.

Class Start:
create():
Transition to State Nocups and call the StorePrice() meta-action.

Class NoCups:

coin(int v):
No Transition in this state but calls ReturnCoins() meta-action when v=0.

insert_cups(int n):
Transition to Idle State, storing cups in a parameter and calling ZeroCF() meta-action.

Class Idle:
coin(int v):
Transition to State Coins_Inserted if v==1, and calling IncreaseCF() meta-action else no transition and calls IncreaseCF() meta-action.

insert_cups(int n):
No transition but storing the number of cups.

set_price():
No Transition and calls StorePrice() meta-action.

card():
Transition to Coins_Inserted State and calls ZeroCF() meta-action.

Class Coins_Inserted:

coin(int v):
No transition and calls ReturnCoins() meta-action when v=0.

cancel():
Transition to Idle State and calls ReturnCoins() and ZeroCF() meta-actions.

dispose_drink(int d):
If Number of cups is > 1, then Transition to Idle state calling DisposeDrink() meta-action if only drink is selected. If additive too is selected then, DisposeAdditive() meta-action too is called.

additive(int a):
No transition and function select or de-selects the additive.

Class OutputProcessor:
This class initializes AbstractFactory object. Implements all meta-action operations(StorePrice(), ZeroCF(), IncreaseCF(), ReturnCoins(), DisposeDrink(int d), DisposeAdditive(int A[])) using Strategy pattern.

Class AbstractFactory:
This is an abstract superclass for the Concrete classes. It defines the operations that return the Vending Machine specific action event components which all Concrete Factories need to implement.
The abstract operations are:
getds(), getsp(), getcf(), geticf(), getrc(), getdd(), getda().

Class ConcreteVending1:

getds():
Returns the shared data structure appropriate for Vending Machine-1.

getsp():
Returns the price stored for Vending Machine-1. The StorePrice class is returned already instantiated with the shared data structure it needs to read data from.

getcf():
Returns the zero Cumulative Fund cp for Vending Machine-1. The ZeroCF class is returned already instantiated with the shared data structure it needs to read data from.

geticf():
Returns the increase Cumulative Fund cp for Vending Machine-1. The IncreaseCF class is returned already instantiated with the shared data structure it needs to read data from.

getrc():
Returns the coins returned for Vending Machine-1. The ReturnsCoins class is returned already instantiated with the shared data structure it needs to read data from.

getdd():
Returns the drink to be disposed for Vending Machine-1. The DisposeDrink class is returned already instantiated with the shared data structure it needs to read data from.

getda():
Returns the additive for the Vending Machine-1. The DisposeDrink class is returned already instantiated with the shared data structure it needs to read data from.

Class ConcreteVending2:
getds():
Returns the shared data structure appropriate for Vending Machine-2.

getsp():
Returns the price stored for Vending Machine-2. The StorePrice class is returned already instantiated with the shared data structure it needs to read data from.

getcf():
Returns the zero Cumulative Fund cp for Vending Machine-2. The ZeroCF class is returned already instantiated with the shared data structure it needs to read data from.

geticf():
Returns the increase Cumulative Fund cp for Vending Machine-2. The IncreaseCF class is returned already instantiated with the shared data structure it needs to read data from.

getrc():
Returns the coins returned for Vending Machine-2. The ReturnsCoins class is returned already instantiated with the shared data structure it needs to read data from.

getdd():
Returns the drink to be disposed for Vending Machine-2. The DisposeDrink class is returned already instantiated with the shared data structure it needs to read data from.

getda():
Returns the additive for the Vending Machine-2. The DisposeDrink class is returned already instantiated with the shared data structure it needs to read data from.

Class DataStore:
This class groups all Data classes in 1 abstract superclass which are used as data structure for each Vending Machine component.

Class DS1:
Instead of getters and setters method, we have accessed the fields directly.

Class DS2:
Instead of getters and setters method, we have accessed the fields directly.

Class StorePrice:
storeprice(): operation is abstract.

Class StorePrice1:
storeprice():
Read the temporary variable p and initialize the initial drink price.

Class StorePrice2:
storeprice():
Read the temporary variable p and initialize the initial drink price.

Class ZeroCF:
zerocf(): operation is abstract.

Class ZeroCF1:
zerocf():
function sets the current funds to 0.

Class ZeroCF2:
zerocf():
function sets the current funds to 0.

Class IncreaseCF:
increasecf(): operation is abstract.

Class IncreaseCF1:
increasecf():
function calculates and stores the added coins

Class IncreaseCF2:
increasecf():
function calculates and stores the added coins

Class ReturnCoins:
returncoins():operation is abstract.

Class ReturnCoins1:
returncoins():
Function returns the coins and sets the current funds to 0.

Class ReturnCoins2:
returncoins():
Function returns the coins and sets the current funds to 0.

Class DisposeDrink:
disposedrink(int d): operation is abstract.

Class DisposeDrink1:
disposedrink(int d):
This function checks for the id selected and disposes the drink accordingly.

Class DisposeDrink2:
disposedrink(int d):
This function checks for the id selected and disposes the drink accordingly.

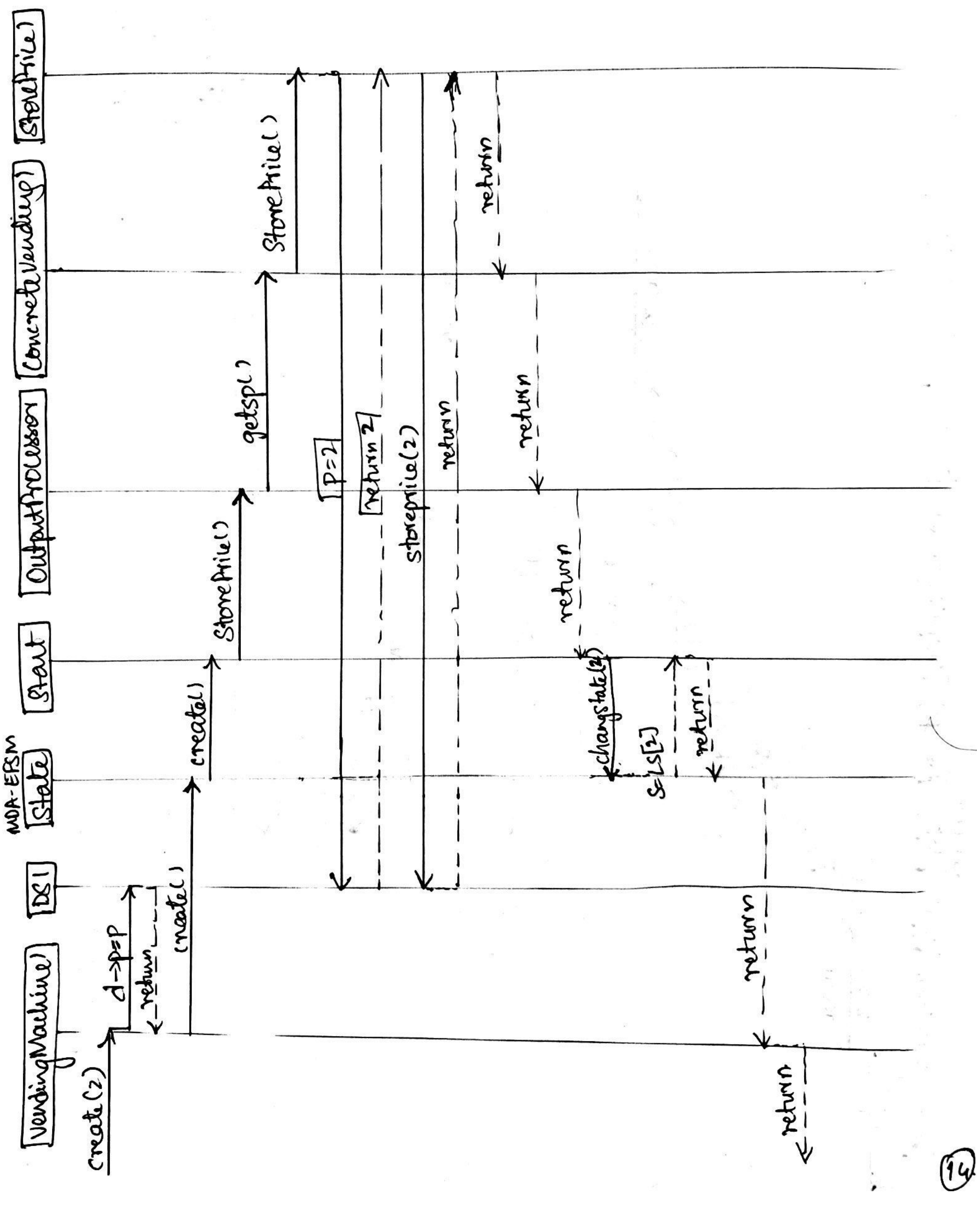
Class DisposeAdditive:
disposeadditive(int A[]): operation is abstract.

Class DisposeAdditive1:
disposeadditive(int A[]):
This function checks if any additive is selected and if the condition is true, disposes the additive.

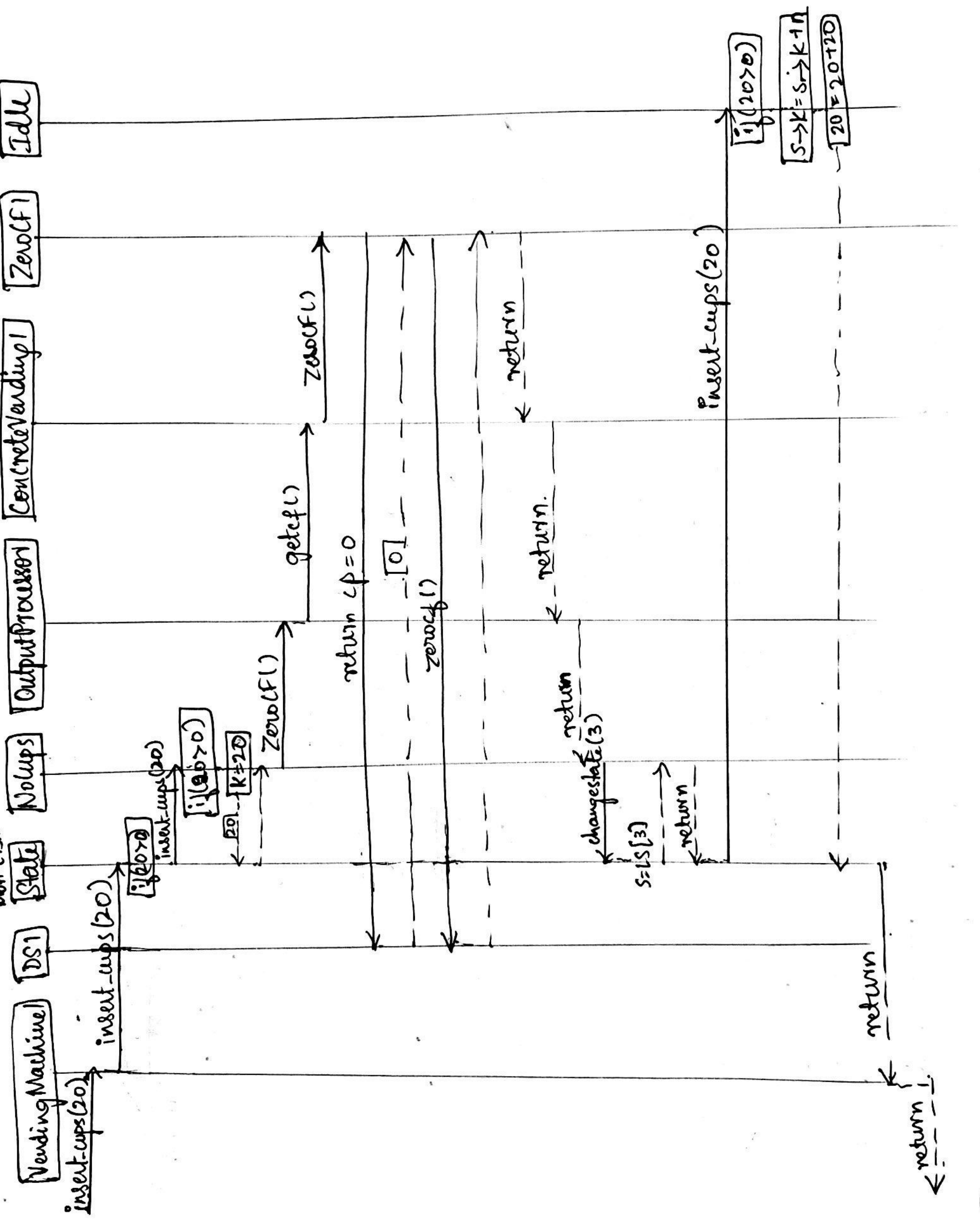
Class DisposeAdditive2:
disposeadditive(int A[]):
This function checks if any additive is selected and if the condition is true, disposes the additive.

4a) Sequence diagrams for VendingMachine-1.

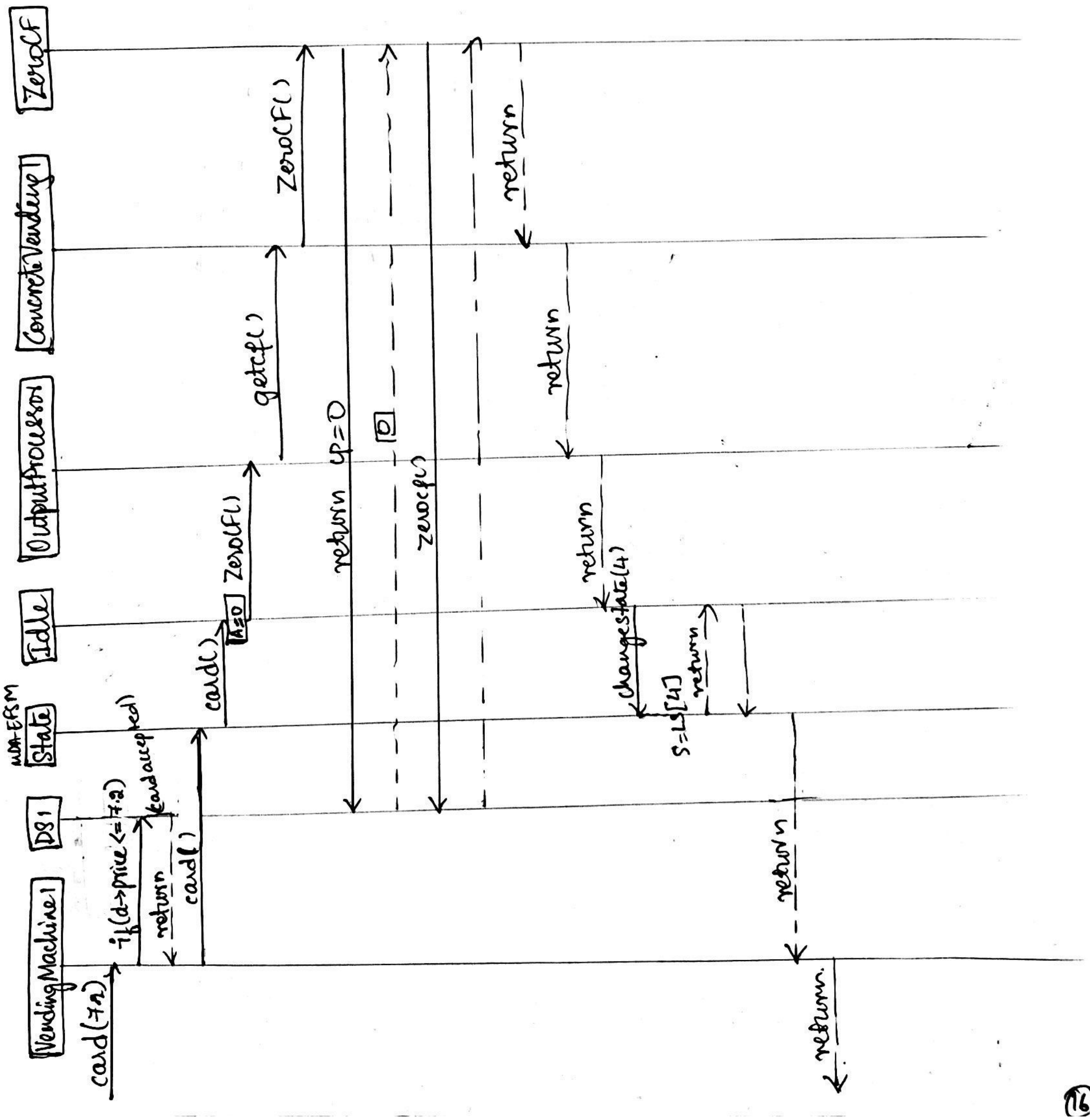
create(2)



insert_cups(20)

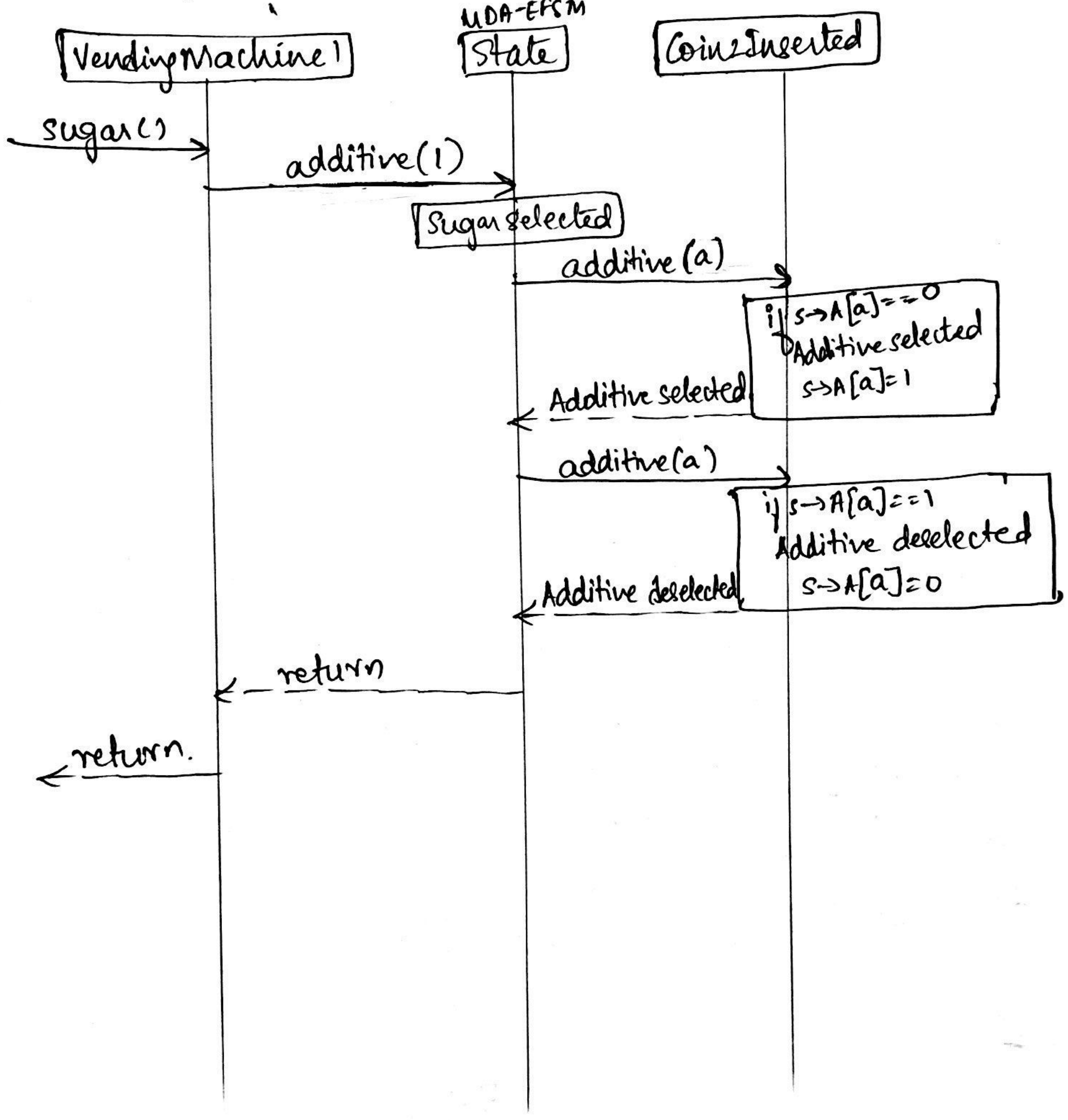


card (7.2)



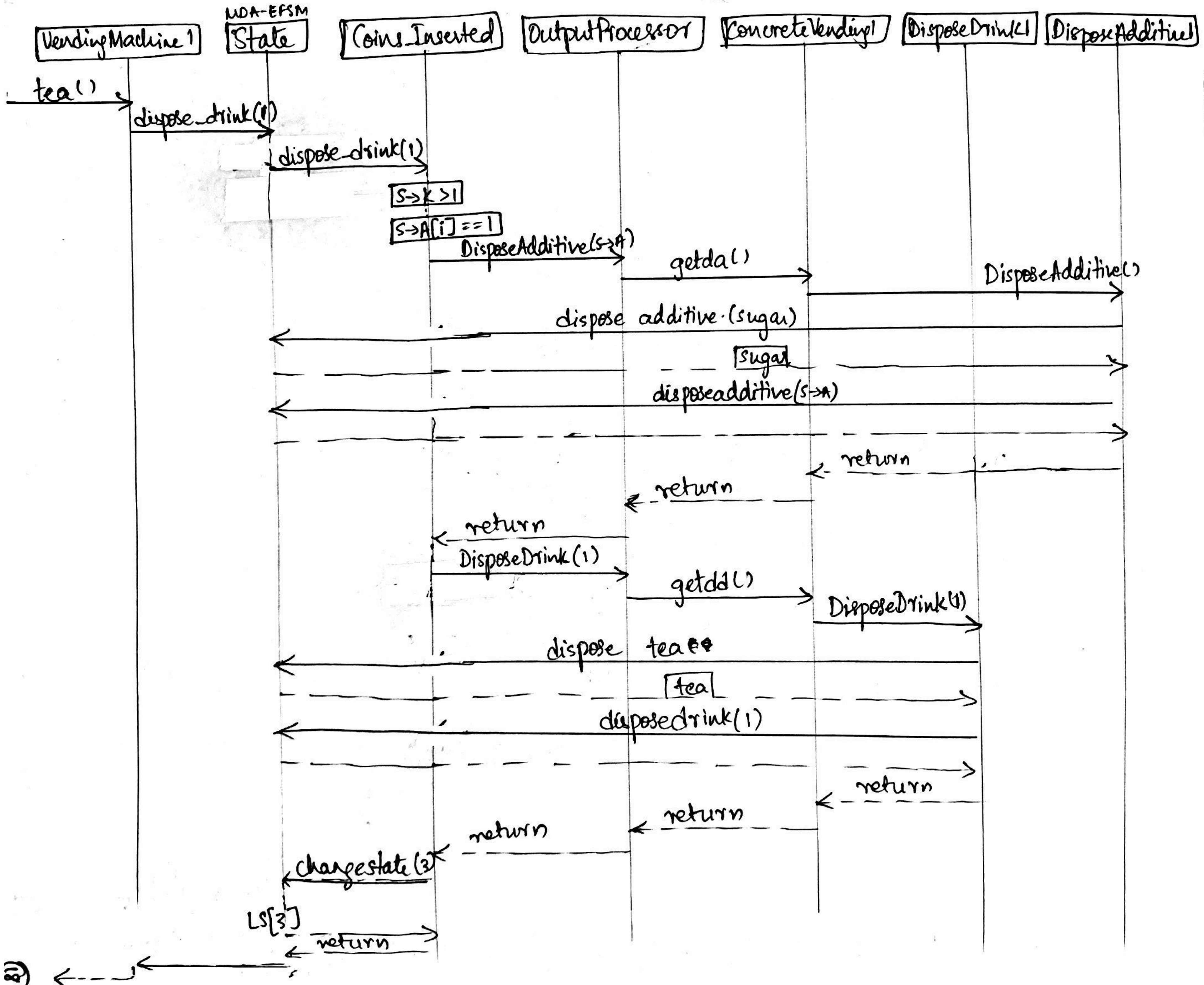
VendingMachine1
DS1
MDA-EFSM State
Idle
OutputProcessor
ConcreteVending1
ZeroCF
card(7.2)
if(d->price <= 7.2)
cardaccepted
return
card()
card()
A=0
ZeroCF()
getcf()
ZeroCF()
return cf=0
0
zerocf()
return
return
return
changestate(4)
S=LS[4]
return
return
return

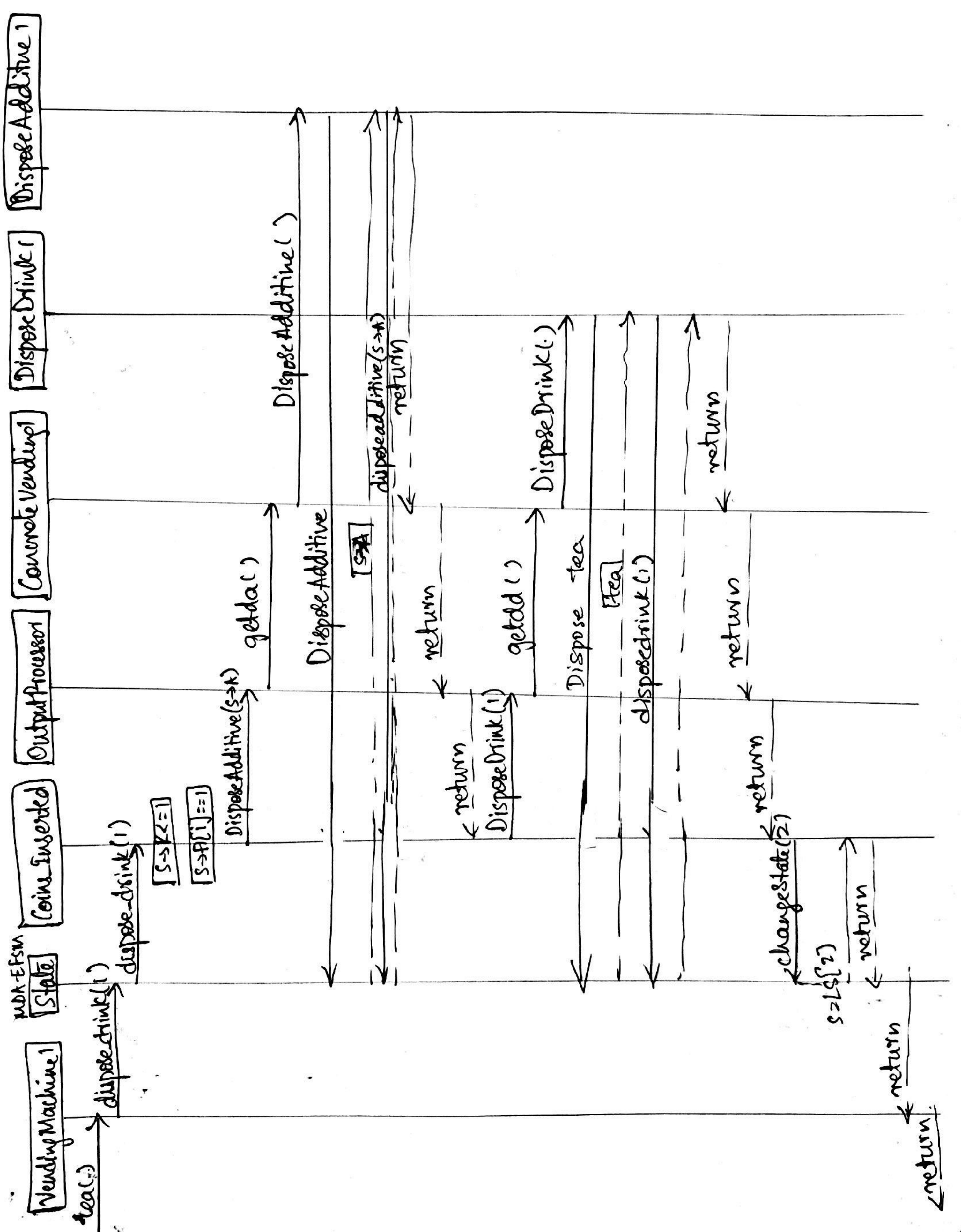
sugar()



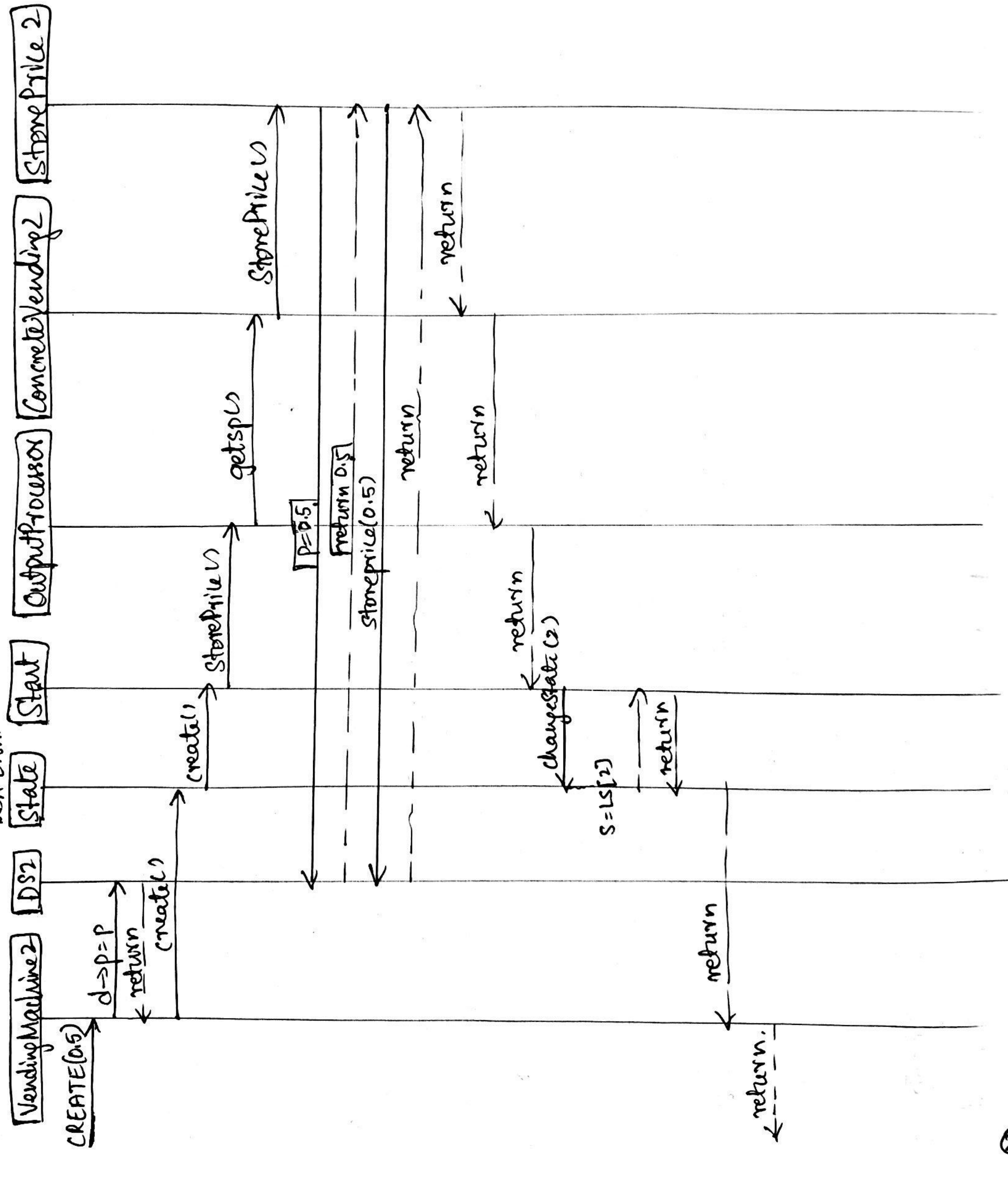
MDA-EFSM
VendingMachine1
State
Coin2Inserted
sugar()
additive(1)
Sugarselected
additive(a)
if s->A[a]==0
Additive selected
s->A[a]=1
Additive selected
additive(a)
if s->A[a]==1
Additive deselected
s->A[a]=0
Additive deselected
return
return.

tea()


MDA-EFSM
VendingMachine1
State
Coins_Inserted
OutputProcessor
ConcreteVending1
DisposeDrink1
DisposeAdditive1
tea()
dispose_drink(1)
dispose_drink(1)
S->k>1
S->A[i]==1
DisposeAdditive(S->A)
getda()
DisposeAdditive()
dispose additive·(sugar)
sugar
disposeadditive(S->A)
return
return
return
DisposeDrink(1)
getdd()
DisposeDrink(1)
dispose tea
tea
disposedrink(1)
return
return
return
changestate(3)
LS[3]
return

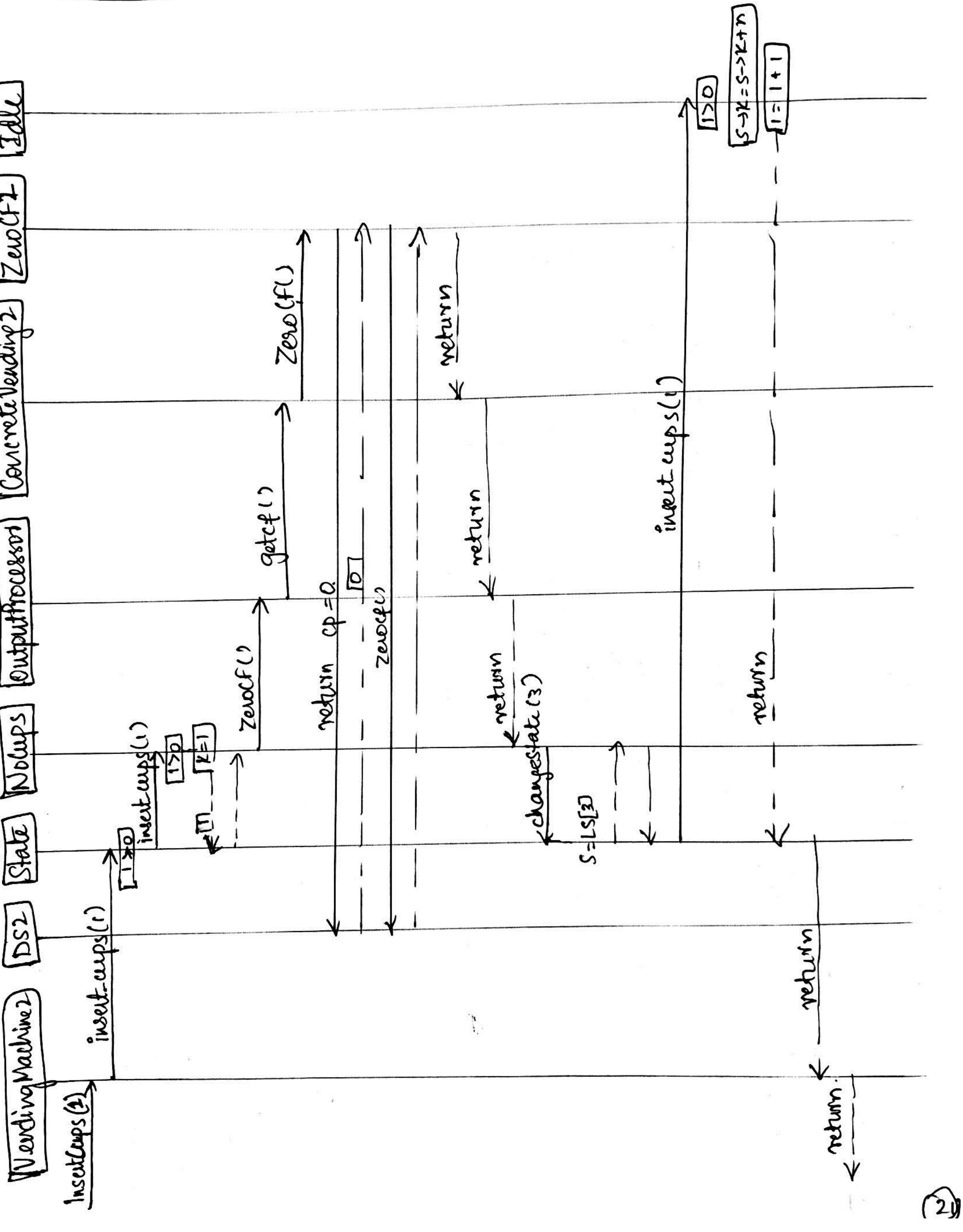

VendingMachine1
MDA-EFSM State
CoinsInserted
OutputProcessor
ConcreteVending1
DisposeDrink1
DisposeAdditive1
tea( )
disposedrink(1)
disposedrink(1)
dispose_drink(1)
S->K<=1
S->A[1]==1
DisposeAdditive(S->A)
getda( )
DisposeAdditive( )
DisposeAdditive
S->A
disposeadditive(S->A)
return
return
return
DisposeDrink(1)
getdd( )
DisposeDrink(.)
Dispose tea
Tea
disposedrink(1)
return
return
return
changestate(S[2])
s=S[2]
return
return
return

4b) sequence diagrams for VendingMachine-2.

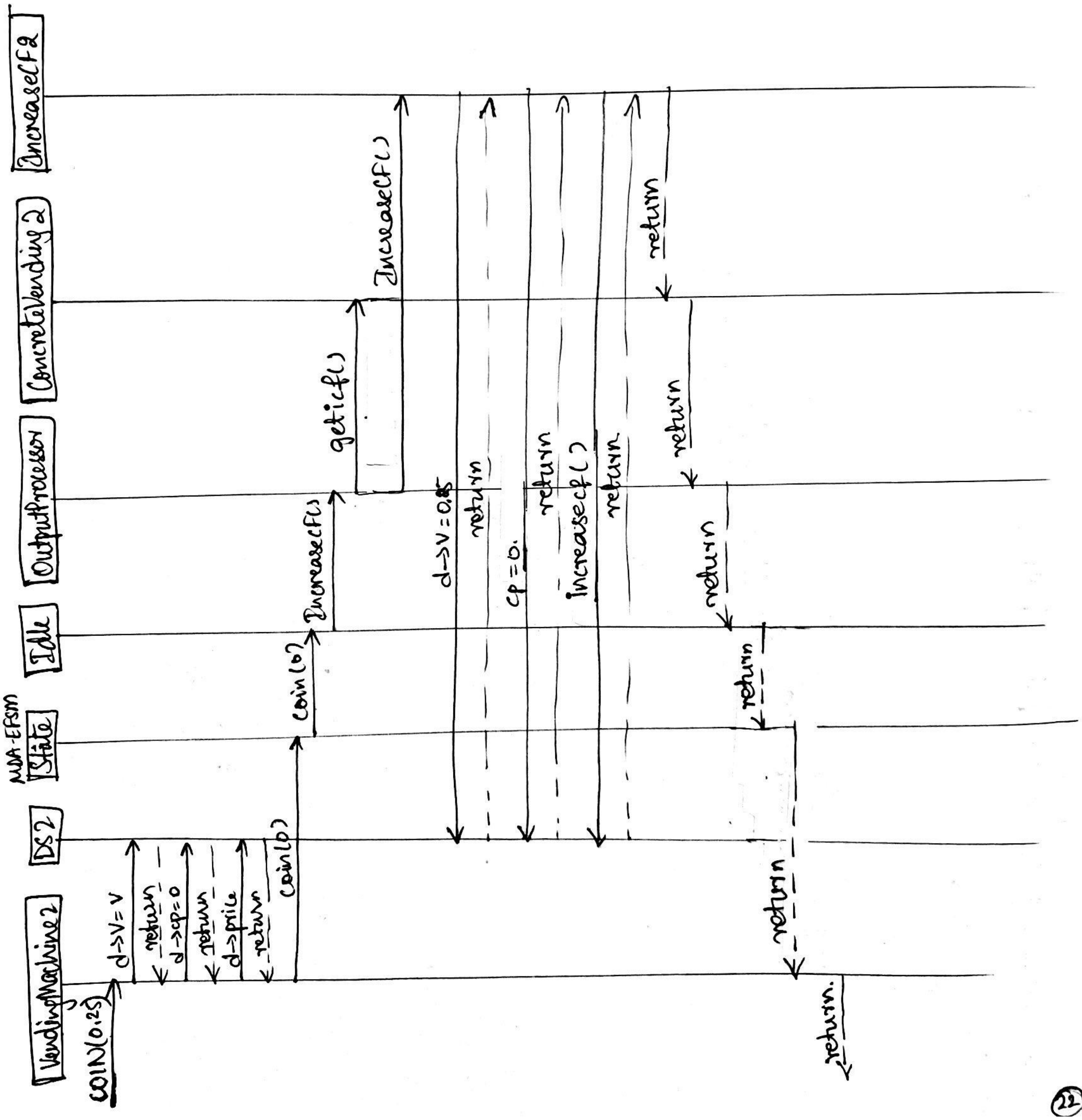


Scanned with CamScanner

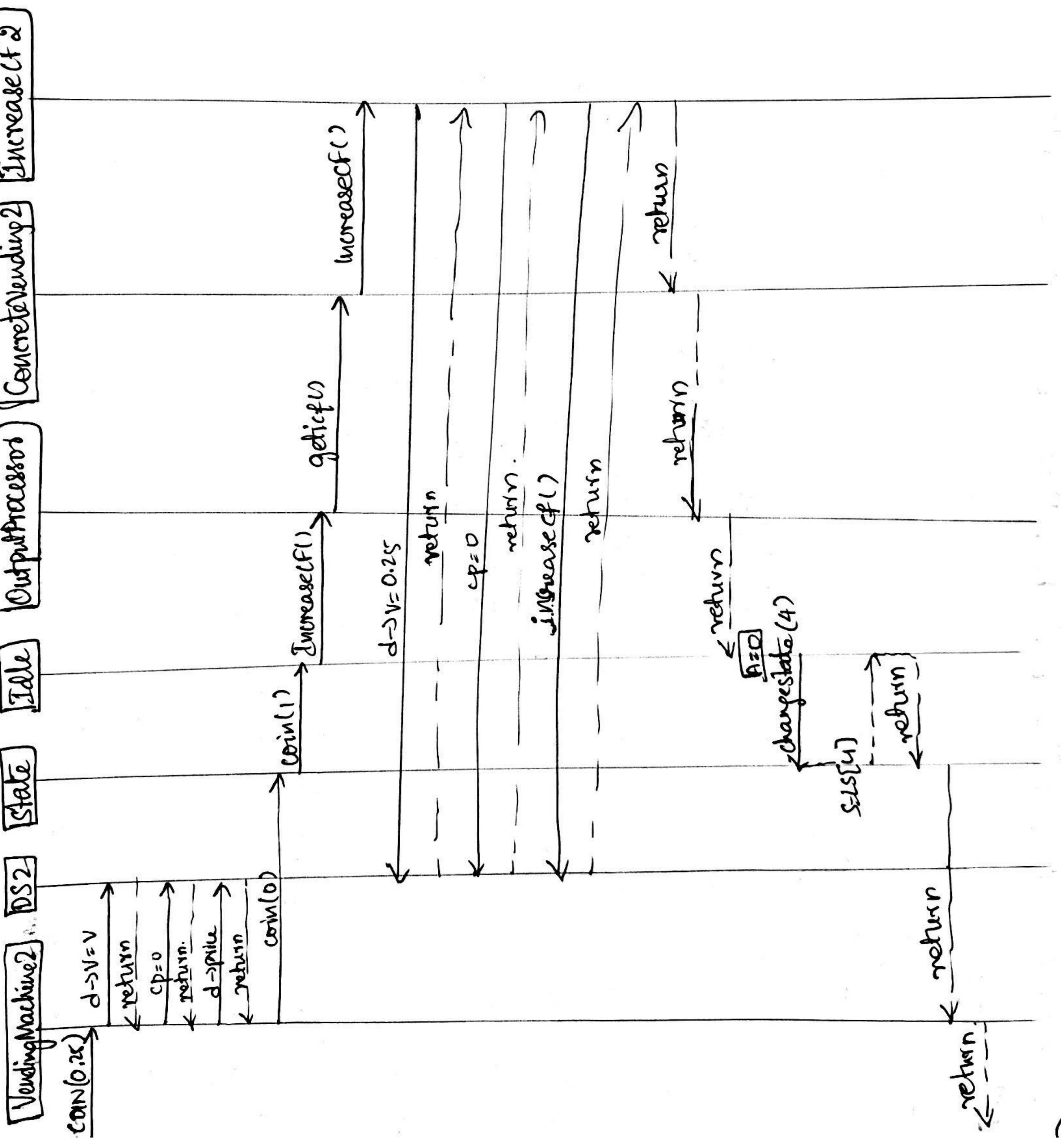
InsertCups(1)



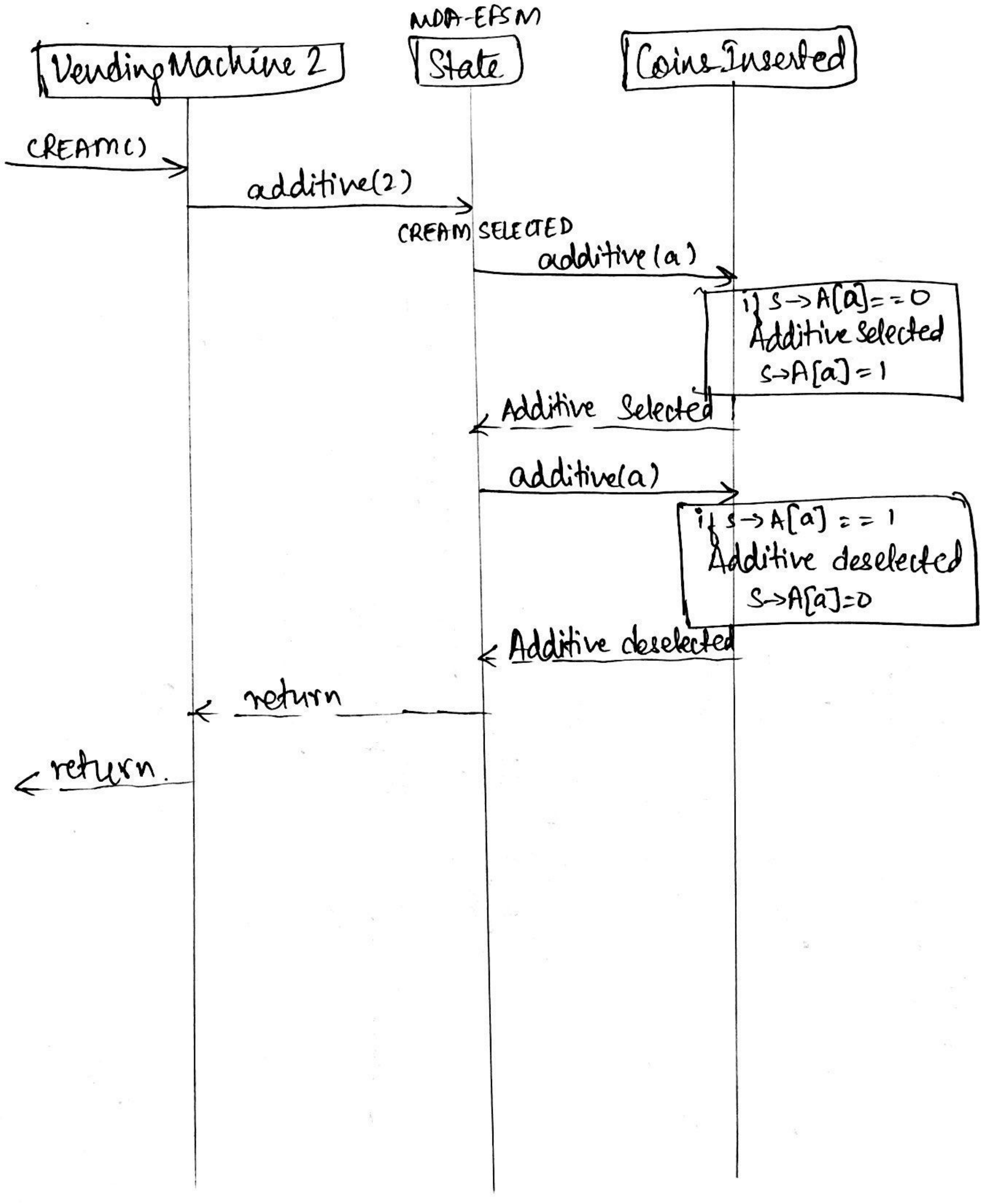
COIN(0.25)


VendingMachine2
DS2
MDA-EFSM
State
Idle
OutputProcessor
ConcreteVending2
IncreaseCF2
COIN(0.25)
d->V=V
return
d->cp=0
return
d->price
return
coin(0)
coin(0)
IncreaseCF()
getICF()
IncreaseCF()
d->V=0.85
return
cp=0.
return
IncreaseCF()
return
return
return
return
return
return
return.

COIN(0.25)

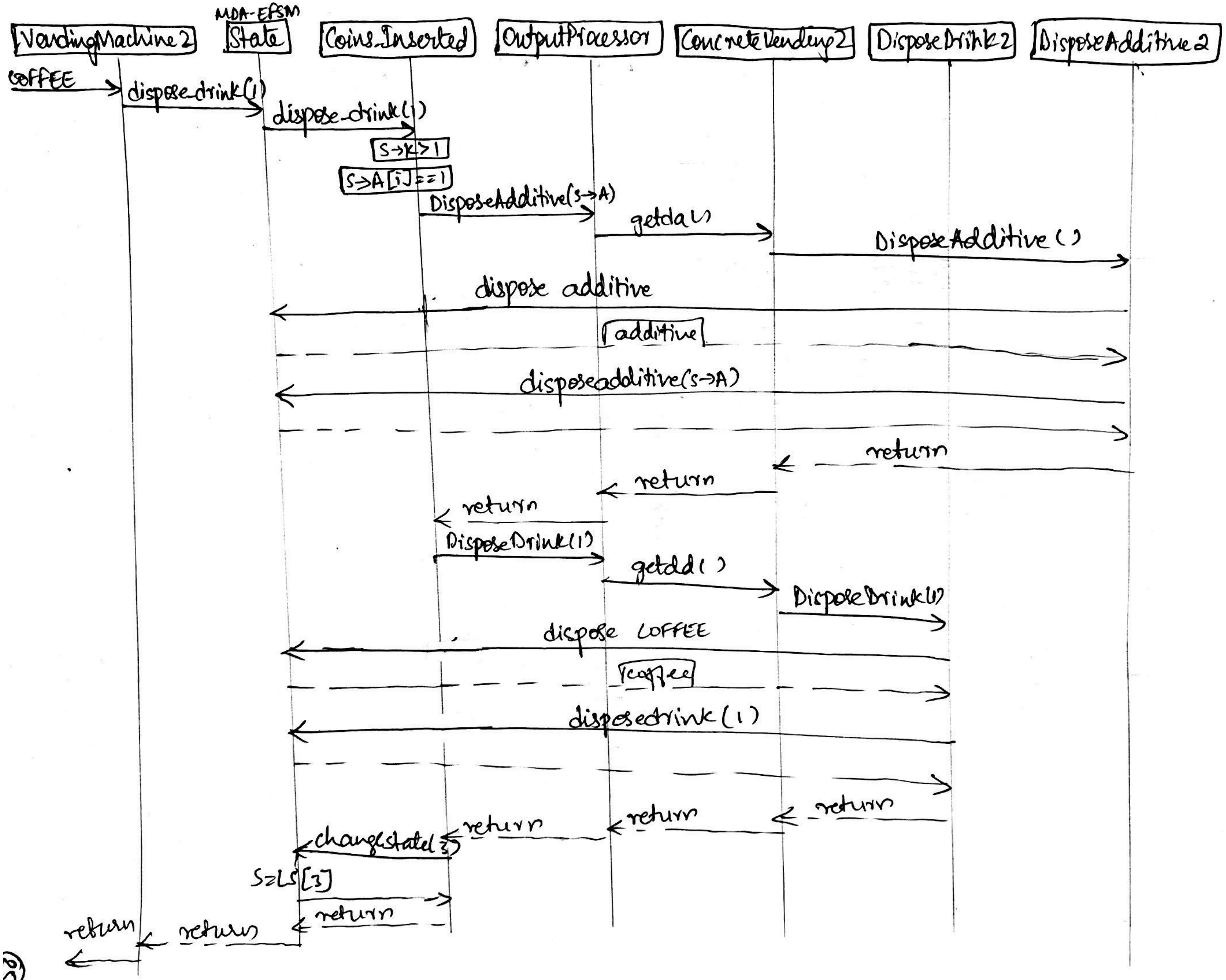

VendingMachine2
DS2
MDA-EFSM
State
Idle
OutputProcessor
ConcreteVending2
IncreaseCf2
COIN(0.25)
d->v=v
return
cp=0
return.
d->price
return
coin(0)
coin(1)
IncreaseCf()
getcf()
increaseCf()
d->v=0.25
return
cp=0
return.
increaseCf()
return
return
return
return
A=0
changestate(4)
s=S[4]
return
return
return.

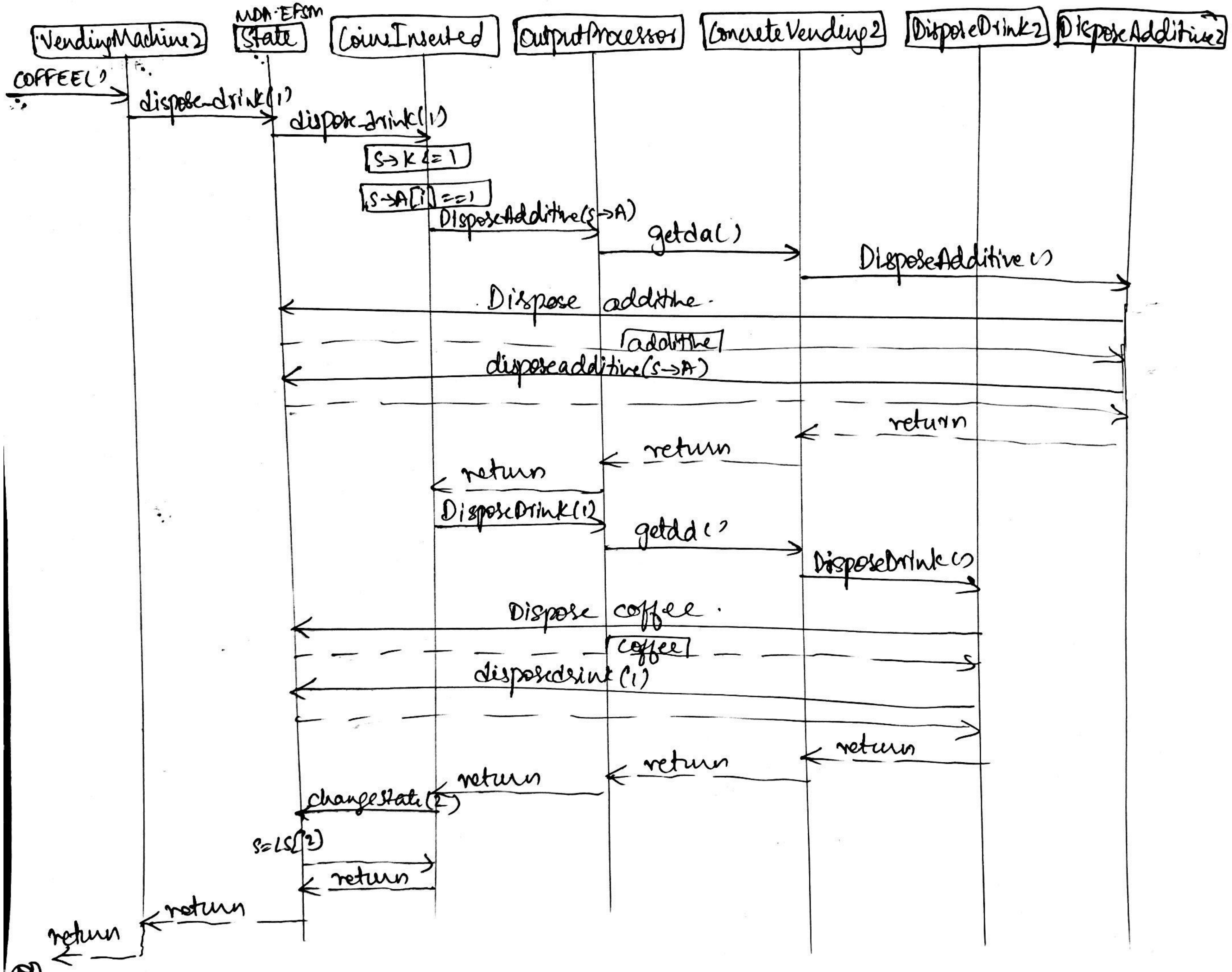
CREAM( )



Scanned with CamScanner

COFFEE( )


MDA-EFSM
VendingMachine2
State
Coins_Inserted
OutputProcessor
ConcreteVending2
DisposeDrink2
DisposeAdditive2
COFFEE
dispose-drink(1)
dispose-drink(1)
S->k>1
S->A[i]==1
DisposeAdditive(s->A)
getda()
DisposeAdditive( )
dispose additive
additive
disposeadditive(s->A)
return
return
return
DisposeDrink(1)
getdd( )
DisposeDrink(1)
dispose COFFEE
coffee
disposedrink(1)
return
return
return
changestate(3)
S=LS[3]
return
return
return


MDA-EFSM
VendingMachine2
State
CoinsInserted
OutputProcessor
ConcreteVending2
DisposeDrink2
DisposeAdditive2
COFFEE()
dispose-drink(1)
dispose-drink(1)
S->K<=1
S->A[i]==1
DisposeAdditive(S->A)
getda()
DisposeAdditive()
Dispose additive.
additive
disposeadditive(S->A)
return
return
return
DisposeDrink(1)
getdd()
DisposeDrink()
Dispose coffee.
coffee
disposedrink(1)
return
return
return
changeState(2)
S=LS[2]
return
return
return