The ATM case study
in AsmetaL

ATM (Cash machine) (Egon Boerger and Robert Staerk. Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, March 11, 2003) : The Problem

Design the control for an ATM, where via a GUI the customer can perform the following operations:

- Op1. Enter the ID (the PIN number). Only one attempt is allowed per session; upon failure the card is withdrawn.

- Op2. Ask for the balance of the account. This operation is allowed only once and only before attempting to withdraw money.

- Op3. Withdraw money from the account. Only one attempt is allowed per session. A warning is issued if the amount required exceeds the balance of the account.

ATM – other requirements

Acc. The central system is supposed to be designed separately.

- It receives the information about every withdrawal and updates the account balance correspondingly.

- The ATM becomes inaccessible for the customer for any other transaction until this update has become effective.

Ref. Extend the ATM to go out-of-service when not enough money is left.

ATM use case description

Caso d'uso (stato ATM)

evento
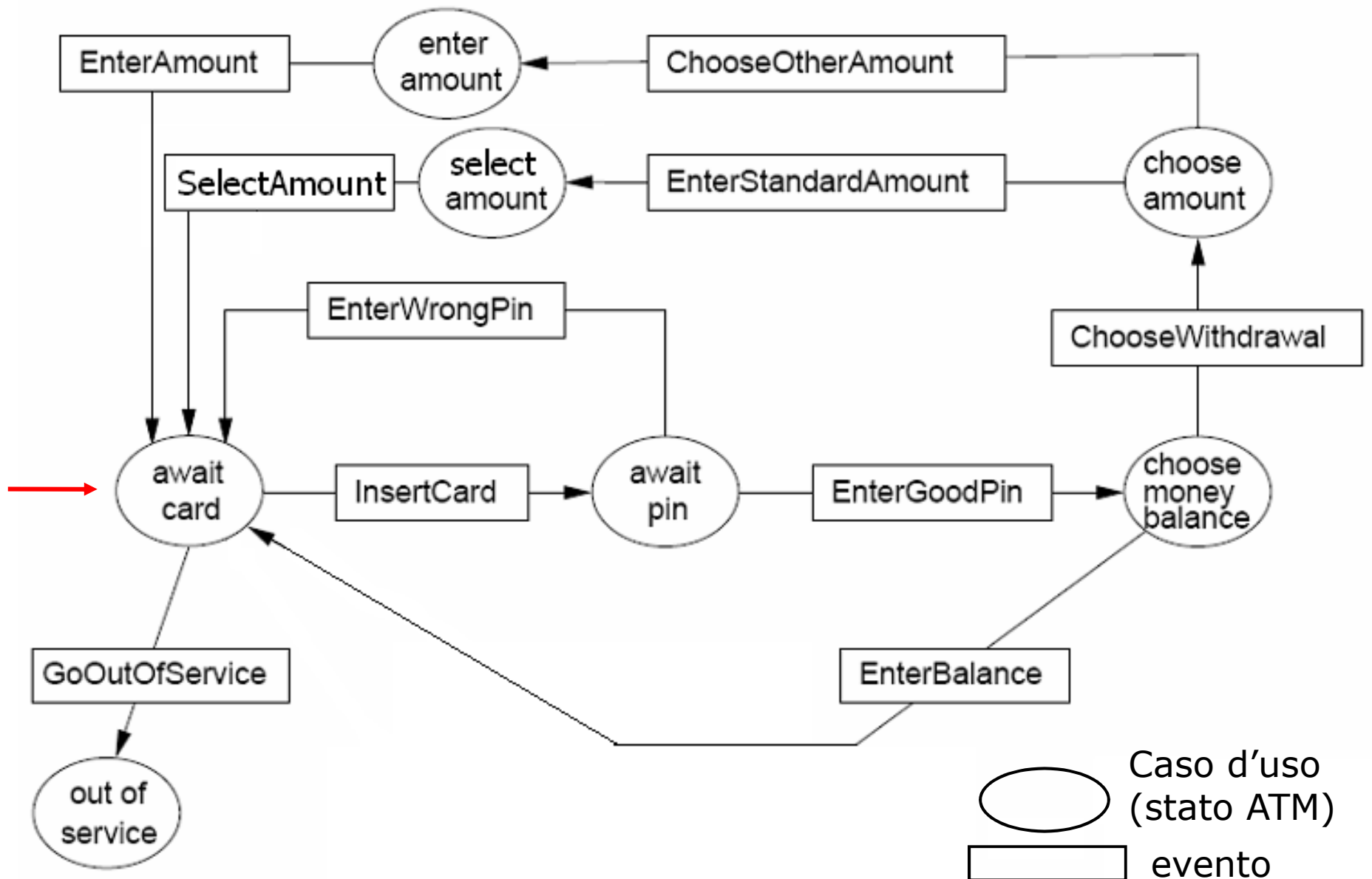
ASM signature

Domains:

- **abstract domain** NumCard

- **enum domain** State =
{ AWAITCARD | AWAITPIN | CHOOSE | OUTOFSERVICE |
CHOOSEAMOUNT | STANDARDAMOUNTSELECTION |
OTHERAMOUNTSELECTION}
States of the ATM

- **enum domain** Service = {BALANCE | WITHDRAWAL |
EXIT}
The customer can: ask for the balance or withdraw money
or exit

- **domain** MoneySize **subsetof** Integer
  e.g. = {10, 20, 40, 50, 100, 150, 200}

- **enum domain** MoneySizeSelection = {STANDARD |
OTHER}

Functions

- **dynamic controlled** currCard: NumCard
the currently inserted card
- **dynamic controlled** atmState: State
records the state of the ATM
- **dynamic controlled** outMess: Any
an output function whose values abstractly represent the messages to be displayed on the screen
- **static** pin: NumCard -> Integer the PIN of a card
- **dynamic controlled** balance: NumCard -> Integer
the account's balance
- **dynamic controlled** accessible: NumCard -> Boolean
indicates whether or not a previous customer ATM operation is still pending in the central system. By setting accessible(CurrCard) to false (see the rule guards for entering a pin number) prevent further transactions until the central system changes the accessibility back to true.

Other functions

- **dynamic monitored** insertedCard: NumCard
inserted card
- **dynamic monitored** insertedPin: Integer
inserted PIN
- **dynamic monitored** selectedService: Service
selected service
- **dynamic monitored** standardOrOther: MoneySizeSelection
selected money size: STANDARD or OTHER
- **dynamic monitored** insertMoneySize: Integer
selected money size (in case of OTHER)
- **dynamic controlled** moneyLeft: Integer
 ATM cash
- **derived** allowed: Prod(NumCard, Integer) -> Boolean
withdrawal iff the balance is >= to the requested money

function allowed($c in NumCard, $m in Integer) =
        balance($c) >= $m

ASM transition rules

Insert a card:

By requirement Op1, the insertion of a card (preceding entering an ID) can be formalized as follows

```
rule r_insertcard =
  if (atmState=AWAITCARD) then
    if (exist $c in NumCard with $c=insertedCard) then
                    par
                        currCard := insertedCard
                        atmState := AWAITPIN
                        outMess := "Enter pin"
                    endpar
            endif
        endif
```

ASM transition rules

Enter the PIN: By Op1 the inserted PIN must be correct and by the requirement Acc, access should be granted only if the account of the current card is *accessible*

```
rule r_enterPin =
if (atmState=AWAITPIN) then
  if (insertedPin=pin(currCard) and accessible(currCard))
    then par
  outMess := "Choose service"
  atmState := CHOOSE
endpar
    else  //wrong PIN or account inaccesible: the card is returned
          //by setting atmState := AWAITCARD
          par
  atmState := AWAITCARD
          if (insertedPin!=pin(currCard))
          then outMess := "Wrong pin" endif
    if (not(accessible(currCard)) and insertedPin=pin(currCard))
          then outMess := "Account non accessible" endif
endpar
  endif
endif
```

Choose service:

By Op2 and Op3: ask for balance, or for money, or exit

```
rule r_chooseService =
  if (atmState=CHOOSE)
  then par
  if (selectedService=BALANCE) //display the balance
  then outMess := balance(currCard) endif
        if (selectedService=WITHDRAWAL)
         then par
       atmState := CHOOSEAMOUNT // standard or other
              outMess := "Choose Standard or Other"
          endpar endif
         if (selectedService=EXIT)
         then par
       atmState := AWAITCARD // choice: EXIT
       outMess := "Goodbye"
              endpar endif
  endpar
   endif
```

Choose amount: By Op3

```
rule r_chooseAmount =
  if (atmState=CHOOSEAMOUNT) then
par
  if (standardOrOther=STANDARD) then
    par
            atmState := STANDARDAMOUNTSELECTION
      outMess := "Select a money size"
    endpar
  endif
  if (standardOrOther=OTHER) then
    par
        atmState := OTHERAMOUNTSELECTION
        outMess := "Enter money size"
    endpar
  endif
endpar
  endif
```

Withdraw money: By Op3 — ASM transition rules

rule r_withdraw =
  par
   if (atmState=STANDARDAMOUNTSELECTION)
   then if (exist $m in MoneySize with $m=insertMoneySize)
        then if (insertMoneySize<=moneyLeft)
             then r_processMoneyRequest [insertMoneySize]
       else outMess := "No enough cash in the ATM"
             endif
endif
  endif
   if (atmState=OTHERAMOUNTSELECTION)
     then if (mod(insertMoneySize, 10)=0)
           then if (insertMoneySize<=moneyLeft)
                then r_processMoneyRequest [insertMoneySize]
         else outMess := " No enough cash in the ATM "
           endif
   else outMess := "Money size not available"
   endif
     endif
  endpar

ASM transition rules

Process money request: By Op3

```
rule r_processMoneyRequest ($m in Integer) =
    if (allowed(currCard, $m))
    then  r_grantMoney[$m]
    else outMess := "Not enough money in your account"
    endif
```

Grant money: By Op3

```
rule r_grantMoney($m in Integer) =
par
    r_subtractFrom[currCard, $m] //update the account balance
    moneyLeft := moneyLeft - insertMoneySize //ATM cash decreases
    seq
      accessible(currCard) := false //set the account inaccessible
      accessible(currCard) := true //Another agent should unblock
                                    //the account!   Così inutile
    endseq
    atmState := AWAITCARD //the card is returned to the customer
    outMess := "Goodbye"
endpar


  rule r_subtractFrom ($c in NumCard, $m in Integer) =
        balance($c) := balance($c) - $m
```

ASM transition rules

Go out of service: By ref.

```
macro rule r_goOutOfService =
    if (moneyLeft < minMoney) then
        par
            atmState := OUTOFSERVICE
            outMess := "Out of Service"
        endpar
    endif
```

where (a new function is added to the signature):

static minMoney: Integer
Minimum amount of money to permit the ATM to work

Go out of service: By ref.

```
macro rule r_goOutOfService =
        if (moneyLeft < minMoney) then
                par
                        atmState := OUTOFSERVICE
                        outMess := "Out of Service"
                endpar
        endif
```

where:

static minMoney: Integer
Minimum amount of money to permit the ATM to work

static maxPrelievo: Integer
Maximum amount of money one can withdraw

ASM transition rules

```
main rule r_Main =
     seq
           r_goOutOfService[]
           par
                r_insertcard[]
                r_enterPin[]
                r_chooseService[]
                r_chooseAmount[]
                r_prelievo[]
           endpar
     endseq
```