

System Modelling and Design

A Simple ATM Beyond Specification

Revision: 1.2, April 23, 2008

Ken Robinson

School of Computer Science & Engineering
The University of New South Wales, Sydney Australia

May 17, 2010

© Ken Robinson 2005-2010

`mailto::k.robinson@unsw.edu.au`

Outline I

Objectives of this Lecture

ATM0: A Simplistic Model of an ATM

ATM0

Improving the Model

ATMR0

Password Encryption

Objectives of this Lecture

- to demonstrate that nondeterminism can be closer to reality than determinism.
- to illustrate the above using a simple ATM example.



ATM0: A Simplistic Model of an ATM

We want to produce a model of an ATM. The model will be kept reasonably simple, but also reasonably realistic.

Required ATM operations:

- an operation to insert the card and provide a password;
- an operation to withdraw money;

The initial attempt might be as shown in the *ATM0* machine. This is likely to be the type of specification produced by someone familiar only with machine level development.

ATM0: A Simplistic Model of an ATM

We want to produce a model of an ATM. The model will be kept reasonably simple, but also reasonably realistic.

Required ATM operations:

- an operation to insert the card and provide a password;
- an operation to withdraw money;

The initial attempt might be as shown in the *ATM0* machine. This is likely to be the type of specification produced by someone familiar only with machine level development.

ATM0: A Simplistic Model of an ATM

We want to produce a model of an ATM. The model will be kept reasonably simple, but also reasonably realistic.

Required ATM operations:

- an operation to insert the card and provide a password;
- an operation to withdraw money;

The initial attempt might be as shown in the *ATM0* machine. This is likely to be the type of specification produced by someone familiar only with machine level development.

ATM0: A Simplistic Model of an ATM

We want to produce a model of an ATM. The model will be kept reasonably simple, but also reasonably realistic.

Required ATM operations:

- an operation to insert the card and provide a password;
- an operation to withdraw money;

The initial attempt might be as shown in the *ATM0* machine. This is likely to be the type of specification produced by someone familiar only with machine level development.

ATM Context I

CONTEXT ATM_ctx

SETS

ACCOUNT

The set of account IDs

RESPONSES

Set of responses

CONSTANTS

OK

REFUSED

RESPONSE

Possible responses

ATM Context II

AXIOMS

axm1: finite(ACCOUNT)

axm4: RESPONSES = {OK, REFUSED}

axm5: OK \neq REFUSED

axm6: RESPONSE = {{OK}, {REFUSED}, \emptyset }

END

Password context I

CONTEXT Password

SETS

PASSWORD

END

ATM0 I

MACHINE ATM0

SEES ATM_ctx, Password

VARIABLES

accounts

password

balance

customer

response

ATM0 II

INVARIANTS

inv1: accounts \subseteq *ACCOUNT*

inv2: finite(accounts)

inv3: password \in *accounts* \rightarrow *PASSWORD*

inv4: balance \in *accounts* $\rightarrow \mathbb{Z}$

inv5: customer \subseteq *accounts*

inv6: card(customer) ≤ 1

inv7: response \in *RESPONSE*

ATM0 III

EVENTS

Initialisation

begin

act1: accounts := ∅

act2: password := ∅

act3: balance := ∅

act4: customer := ∅

act5: response := ∅

end

ATM0 IV

Event *InsertCard* $\hat{=}$

any *account*
pass

when

grd1: *account* \in *ACCOUNT*

grd2: *pass* \in *PASSWORD*

grd3: *customer* = \emptyset

grd4: *response* = \emptyset

ATM0 V

then

act1:

response, customer : |
(*account* ∈ *accounts* ∧ *pass* = *password*(*account*)
⇒ *response*' = {OK} ∧ *customer*' = {*account*})
∧ ((*account* ∉ *accounts* ∨ *pass* ≠ *password*(*account*))
⇒ *response*' = {REFUSED} ∧ *customer*' = ∅)

end

ATM0 VI

Event *Withdraw* $\hat{=}$

any *amount*
account

when

grd1: *response* = \emptyset

grd2: *customer* $\neq \emptyset$

grd3: *amount* $\in \mathbb{N}$

grd4: $\{\textit{account}\} = \textit{customer}$

ATM0 VII

then

act1: response : |
(balance(account) ≥ amount
⇒ response' = {OK})
∧ (balance(account) < amount
⇒ response' = {REFUSED})

act2: balance : |
(balance(account) ≥ amount
⇒ balance' = balance
⇐ {account ↦ balance(account) - amount})
∧ (balance(account) < amount
⇒ balance' = balance)

end

ATM0 VIII

```
Event ResetResponse  $\hat{=}$   
  when  
    grd1: response  $\neq \emptyset$   
  then  
    act1: response :=  $\emptyset$   
  end  
END
```

Resets response

Improving the Model

This *ATM0* model is seriously ill-conceived. It puts bank-like state inside the ATM. This is obviously wrong: ATMs have no banking knowledge, they are simply boxes in the wall that interact with a card user and communicate with a remote banking system.

We will attempt to build a more realistic model that separates the ATM and the remote banking system.

First, we need to specify the context information that is common to both the ATM and the remote banking system. This is shown in *CardStatus* and *Password* contexts. It's split into two machines because the account, service card and response modelling "belongs" to the banking system, but the modelling of passwords is global.

Improving the Model

This *ATM0* model is seriously ill-conceived. It puts bank-like state inside the ATM. This is obviously wrong: ATMs have no banking knowledge, they are simply boxes in the wall that interact with a card user and communicate with a remote banking system.

We will attempt to build a more realistic model that separates the ATM and the remote banking system.

First, we need to specify the context information that is common to both the ATM and the remote banking system. This is shown in *CardStatus* and *Password* contexts. It's split into two machines because the account, service card and response modelling “belongs” to the banking system, but the modelling of passwords is global.

Improving the Model

This *ATM0* model is seriously ill-conceived. It puts bank-like state inside the ATM. This is obviously wrong: ATMs have no banking knowledge, they are simply boxes in the wall that interact with a card user and communicate with a remote banking system.

We will attempt to build a more realistic model that separates the ATM and the remote banking system.

First, we need to specify the context information that is common to both the ATM and the remote banking system. This is shown in *CardStatus* and *Password* contexts. It's split into two machines because the account, service card and response modelling "belongs" to the banking system, but the modelling of passwords is global.

Improving the Model

This *ATM0* model is seriously ill-conceived. It puts bank-like state inside the ATM. This is obviously wrong: ATMs have no banking knowledge, they are simply boxes in the wall that interact with a card user and communicate with a remote banking system.

We will attempt to build a more realistic model that separates the ATM and the remote banking system.

First, we need to specify the context information that is common to both the ATM and the remote banking system. This is shown in *CardStatus* and *Password* contexts. It's split into two machines because the account, service card and response modelling “belongs” to the banking system, but the modelling of passwords is global.

Improving the Model

This *ATM0* model is seriously ill-conceived. It puts bank-like state inside the ATM. This is obviously wrong: ATMs have no banking knowledge, they are simply boxes in the wall that interact with a card user and communicate with a remote banking system.

We will attempt to build a more realistic model that separates the ATM and the remote banking system.

First, we need to specify the context information that is common to both the ATM and the remote banking system. This is shown in *CardStatus* and *Password* contexts. It's split into two machines because the account, service card and response modelling “belongs” to the banking system, but the modelling of passwords is global.

ATM machine I

MACHINE ATM

SEES ATM_ctx, Password

VARIABLES

response

The variables of this machine model

customer

what we may think of as a User Interface.

balance

Each variable is a set that may be either empty

money

or contain a single value.

ATM machine II

INVARIANTS

inv1: customer \subseteq *ACCOUNT*

inv2: card(customer) ≤ 1

inv3: response \in *RESPONSE*

inv4: balance $\subseteq \mathbb{Z}$

inv5: finite(balance)

inv6: card(balance) ≤ 1

inv7: money $\subseteq \mathbb{N}$

inv8: finite(money)

inv9: card(money) ≤ 1

ATM machine III

EVENTS

Initialisation

begin

act1: customer := ∅

act2: response := ∅

act3: balance := ∅

act4: money := ∅

end

ATM machine IV

Event *InsertCard* $\hat{=}$

Insert service card into ATM

any *scard*
pass

when

grd1: *customer* = \emptyset

grd2: *response* = \emptyset

grd3: *scard* \in SCARD

grd4: *pass* \in PASSWORD

then

act1: *response*, *customer* : |
response' \in { {OK}, {REFUSED} }
 \wedge *customer'* \in \mathbb{P} (ACCOUNT) \wedge (*response'* = {OK}
 \Rightarrow *customer'* = {GENSCARD⁻¹(*scard*)})
 \wedge (*response'* = {REFUSED} \Rightarrow *customer'* = \emptyset)

end

ATM machine V

Event *Withdraw* $\hat{=}$ Make withdrawal from ATM
 any *amount*
 when
grd1: *customer* $\neq \emptyset$
grd2: *amount* $\in \mathbb{N}$
 then
act1:
 response, money, balance : |
 response' $\in \{\{OK\}, \{REFUSED\}\}$
 \wedge *balance'* $\subseteq \mathbb{Z} \wedge$ *finite*(*balance'*) \wedge (*response'* = {OK})
 \Rightarrow *money'* = {*amount*} \wedge *balance'* $\in \mathbb{P}(\mathbb{Z})$
 \wedge *card*(*balance'*) ≤ 1
 \wedge (*response'* = {REFUSED})
 \Rightarrow *money'* = $\emptyset \wedge$ *balance'* = \emptyset
 end

ATM machine VI

Event *RemoveCard* $\hat{=}$

when

grd1: *customer* $\neq \emptyset$

then

act1: *response* := {OK}

act2: *customer* := \emptyset

end

Customer terminates session

ATM machine VII

Event *ResetResponse* $\hat{=}$ Reset response when no customer
using ATM

when

grd1: *customer* = \emptyset

grd2: *response* $\neq \emptyset$

then

act1: *response* := \emptyset

end

ATM machine VIII

Event *ResetUI* $\hat{=}$

when

grd1: $customer \neq \emptyset \Rightarrow money \neq \emptyset$

grd2: $customer \neq \emptyset \Rightarrow balance \neq \emptyset$

grd3: $customer \neq \emptyset \Rightarrow response \neq \emptyset$

then

act1: $money := \emptyset$

act2: $balance := \emptyset$

act3: $response := \emptyset$

end

END

Reset User Interface

CardStatus context

CONTEXT CardStatus
EXTENDS ServiceCards
SETS

CARDSTATUS

CONSTANTS

validaccounts
currentbalance
withdrawlimit
password
CARDOK
CARDNOK

AXIOMS

axm1: $validaccounts \subseteq ACCOUNT$
axm2: $currentbalance \in validaccounts \rightarrow \mathbb{Z}$
axm3: $withdrawlimit \in validaccounts \rightarrow \mathbb{N}$
axm4: $CARDSTATUS = \{CARDOK, CARDNOK\}$
axm5: $CARDOK \neq CARDNOK$
axm6: $password \in validaccounts \rightarrow PASSWORD$

END

ServiceCard context

CONTEXT ServiceCards

EXTENDS ATM_ctx

SETS

SCARD

The set of service cards

CONSTANTS

GENSCARD

An injective function that maps service cards to accounts

AXIOMS

axm1: finite(SCARD)

axm2: GENSCARD ∈ ACCOUNT ↦ SCARD

END

We are now modelling a service card, distinct from the account. We assume that the service card can be represented by information that is generated from the account, and that the account can be extracted from the service card.

ATMR0 I

We show two stages in refinement of the ATM. The first attempt, ATMR0, is nearly what we are aiming for, but it contains modelling of the login management that is really nothing to do with the pure interface view of an ATM.

MACHINE ATMR0

REFINES ATM

SEES CardStatus

VARIABLES

response

The variables of this machine model

customer

what we may think of as a User Interface.

balance

Each variable is a set that may be either empty

money

or contain a single value.

ATM0 V

Event *InsertCard_nok* $\hat{=}$

refines *InsertCard*

any *scard*
pass
account

when

grd1: *customer* = \emptyset

grd2: *response* = \emptyset

grd3: *scard* \in SCARD

grd4: *account* = GENSCARD⁻¹(*scard*)

grd5:

account \in validaccounts \Rightarrow *pass* \neq password(*account*)

then

act1: *response* := {REFUSED}

end

ATMR0 VI

Event $Withdraw_ok \hat{=}$

Make withdrawal from ATM

refines $Withdraw$

any $amount$
 $account$

when

$grd1: customer \neq \emptyset$

$grd2: amount \in \mathbb{N}$

$grd3: customer = \{account\}$

$grd4: amount \leq withdrawlimit(account)$

then

$act1: response := \{OK\}$

$act2: balance := \{\emptyset\} \cup \{n \cdot n \in \mathbb{Z} \mid \{n\}\}$

$act3: money := \{amount\}$

end

ATMRO VIII

Event *RemoveCard* $\hat{=}$

Customer terminates session

refines *RemoveCard*

when

grd1: *customer* $\neq \emptyset$

then

act1: *response* := {*OK*}

act2: *customer* := \emptyset

end

ATMRO IX

Event $ResetResponse \hat{=}$ Reset response when no customer
using ATM

refines $ResetResponse$

when

$grd1$: $customer = \emptyset$

$grd2$: $response \neq \emptyset$

then

$act1$: $response := \emptyset$

end

ATMR0 X

Event *ResetUI* $\hat{=}$

refines *ResetUI*

when

grd1: $customer \neq \emptyset \Rightarrow money \neq \emptyset$

grd2: $customer \neq \emptyset \Rightarrow balance \neq \emptyset$

grd3: $customer \neq \emptyset \Rightarrow response \neq \emptyset$

then

act1: $money := \emptyset$

act2: $balance := \emptyset$

act3: $response := \emptyset$

end

END

Reset User Interface

Password Encryption

In *ATMRO* we model the mapping from account to password with a function $accounts \rightarrow PASSWORD$.

Looking ahead to implementation, we recognise that it would be unwise to implement a mapping from account to a plaintext password. It would be more secure to encrypt the password. To provide facilities for this we introduce a new machine *Encryption*.

We also specify the operation *CheckPassword* as comparing encrypted passwords, rather than comparing plain passwords. Notice that we need to “think ahead” on this issue: if we specified the operation as comparing plain passwords, we could not later decide to implement the operation using comparison of encrypted passwords as this is weaker than comparing plain passwords and is hence not a refinement.

Password Encryption

In *ATM₀* we model the mapping from account to password with a function $accounts \rightarrow PASSWORD$.

Looking ahead to implementation, we recognise that it would be unwise to implement a mapping from account to a plaintext password. It would be more secure to encrypt the password. To provide facilities for this we introduce a new machine *Encryption*.

We also specify the operation *CheckPassword* as comparing encrypted passwords, rather than comparing plain passwords. Notice that we need to “think ahead” on this issue: if we specified the operation as comparing plain passwords, we could not later decide to implement the operation using comparison of encrypted passwords as this is weaker than comparing plain passwords and is hence not a refinement.

Password Encryption

In *ATMRO* we model the mapping from account to password with a function $accounts \rightarrow PASSWORD$.

Looking ahead to implementation, we recognise that it would be unwise to implement a mapping from account to a plaintext password. It would be more secure to encrypt the password. To provide facilities for this we introduce a new machine *Encryption*.

We also specify the operation *CheckPassword* as comparing encrypted passwords, rather than comparing plain passwords. Notice that we need to “think ahead” on this issue: if we specified the operation as comparing plain passwords, we could not later decide to implement the operation using comparison of encrypted passwords as this is weaker than comparing plain passwords and is hence not a refinement.

Password Encryption

In *ATMRO* we model the mapping from account to password with a function *accounts* \rightarrow *PASSWORD*.

Looking ahead to implementation, we recognise that it would be unwise to implement a mapping from account to a plaintext password. It would be more secure to encrypt the password. To provide facilities for this we introduce a new machine *Encryption*.

We also specify the operation *CheckPassword* as comparing encrypted passwords, rather than comparing plain passwords. Notice that we need to “think ahead” on this issue: if we specified the operation as comparing plain passwords, we could not later decide to implement the operation using comparison of encrypted passwords as this is weaker than comparing plain passwords and is hence not a refinement.



CONTEXT Encryption
EXTENDS Password
SETS
CRYPT
CONSTANTS
ENCRYPT
AXIOMS
axm1: ENCRYPT ∈ PASSWORD → CRYPT
END

MACHINE ATMR1

REFINES ATMR0

SEES CardStatus1

VARIABLES

response

The variables of this machine model

customer

what we may think of as a User Interface.

balance

Each variable is a set that may be either empty

money

or contain a single value.

EVENTS

Initialisation

begin

act1: *customer* := ∅

act2: *response* := ∅

act3: *balance* := ∅

act4: *money* := ∅

end



Event $InsertCard_ok_ \hat{=}$

Insert service card into ATM

refines $InsertCard_ok_$

any $scard$
 $pass$
 $account$

when

grd1: $customer = \emptyset$

grd2: $response = \emptyset$

grd3: $scard \in SCARD$

grd4: $account = GENSCARD^{-1}(scard)$

grd5: $account \in validaccounts$

grd6: $ENCRYPT(pass) = cryptpass(account)$

then

act1: $response := \{OK\}$

act2: $customer := \{account\}$

end



```

Event InsertCard_nok  $\hat{=}$ 
refines InsertCard_nok
    any scard
        pass
        account

    when

grd1: customer =  $\emptyset$ 
grd2: response =  $\emptyset$ 
grd3: scard  $\in$  SCARD
grd4: account = GENSCARD-1(scard)
grd5: account  $\in$  validaccounts  $\Rightarrow$  ENCRYPT(pass)  $\neq$ 
        cryptpass(account)

    then

act1: response := {REFUSED}
    end
  
```

Event *Withdraw_ok* $\hat{=}$ Make withdrawal from ATM
refines *Withdraw_ok*
 any *amount*
 account

 when
grd1 : *customer* $\neq \emptyset$
grd2 : *amount* $\in \mathbb{N}$
grd3 : *customer* = {*account*}
grd4 : *amount* \leq *withdrawlimit*(*account*)
 then
act1 : *response* := {OK}
act2 : *balance* \in { \emptyset } \cup {*n* · *n* $\in \mathbb{Z}$ | {*n*} }
act3 : *money* := {*amount*}
 end

```
Event Withdraw_nok  $\hat{=}$ 
refines Withdraw_nok
  any amount
     account

  when

  grd1 : customer  $\neq \emptyset$ 
  grd2 : amount  $\in \mathbb{N}$ 
  grd3 : customer = {account}
  grd4 : amount > withdrawlimit(account)

  then

  act1 : response := {REFUSED}
  act2 : balance :=  $\emptyset$ 
  act3 : money :=  $\emptyset$ 

  end
```



```

Event RemoveCard  $\hat{=}$ 
refines RemoveCard
  when
grd1: customer  $\neq \emptyset$ 
  then
act1: response := {OK}
act2: customer :=  $\emptyset$ 
  end

```

Customer terminates session

Reset User Interface

```
Event ResetUI  $\hat{=}$   
refines ResetUI  
  when  
    grd1: customer  $\neq \emptyset \Rightarrow$  money  $\neq \emptyset$   
    grd2: customer  $\neq \emptyset \Rightarrow$  balance  $\neq \emptyset$   
    grd3: customer  $\neq \emptyset \Rightarrow$  response  $\neq \emptyset$   
  then  
    act1: money :=  $\emptyset$   
    act2: balance :=  $\emptyset$   
    act3: response :=  $\emptyset$   
  end  
END
```