

System Modelling and Design

A Simple ATM

Beyond Specification

Revision: 1.2, April 23, 2008

Ken Robinson

May 17, 2010

©Ken Robinson 2005-2010

[mailto::k.robinson@unsw.edu.au](mailto:k.robinson@unsw.edu.au)

Contents

1 Objectives of this Lecture	1
2 ATM0: A Simplistic Model of an ATM	2
2.1 ATM0	2
2.2 Improving the Model	4
2.3 ATMR0	8
2.4 Password Encryption	11

1 Objectives of this Lecture

Please note: This is an article version of the ATM lecture. At the moment there is a problem producing slides from the \LaTeX version of the machines produced by the Rodin \LaTeX generator.

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

2 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.

2.1 ATM0

ATM Context

CONTEXT ATM_ctx

SETS ACCOUNT The set of account IDs RESPONSES Set of responses

CONSTANTS OK REFUSED RESPONSE Possible responses

AXIOMS

axm1: $finite(ACCOUNT)$

axm4: $RESPONSES = \{OK, REFUSED\}$

axm5: $OK \neq REFUSED$

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

END

Password context

CONTEXT Password

SETS PASSWORD

END

MACHINE ATM0

SEES ATM_ctx, Password

VARIABLES accounts password balance customer response

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: $card(customer) \leq 1$
inv7: $response \in RESPONSE$

EVENTS

Initialisation

begin

act1: $accounts := \emptyset$
act2: $password := \emptyset$
act3: $balance := \emptyset$
act4: $customer := \emptyset$
act5: $response := \emptyset$

end

Event *InsertCard* $\hat{=}$

any *account* *pass*

when

grd1: $account \in ACCOUNT$
grd2: $pass \in PASSWORD$
grd3: $customer = \emptyset$
grd4: $response = \emptyset$

then

act1: $response, customer : |$
 $(account \in accounts \wedge pass = password(account)$
 $\Rightarrow response' = \{OK\} \wedge customer' = \{account\})$
 $\wedge ((account \notin accounts \vee pass \neq password(account))$
 $\Rightarrow response' = \{REFUSED\} \wedge customer' = \emptyset)$

end

Event *Withdraw* $\hat{=}$

any *amount* *account*

when

grd1: $response = \emptyset$

grd2: $customer \neq \emptyset$

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

grd4: $\{account\} = customer$

then

act1: $response : |$
 $(balance(account) \geq amount$
 $\Rightarrow response' = \{OK\})$
 $\wedge (balance(account) < amount$
 $\Rightarrow response' = \{REFUSED\})$

act2: $balance : |$
 $(balance(account) \geq amount$
 $\Rightarrow balance' = balance$
 $\Leftarrow \{account \mapsto balance(account) - amount\})$
 $\wedge (balance(account) < amount$
 $\Rightarrow balance' = balance)$

end

Event $ResetResponse \hat{=}$

Resets response

when

grd1: $response \neq \emptyset$

then

act1: $response := \emptyset$

end

END

2.2 Improving the Model

This *ATMO* 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.

One way of thinking about how you should model a machine like an ATM is to imagine watching another person using an ATM, from a little distance. Your model should describe what you see, not what you might imagine is going on inside the machine.

ATM0 could never be the result of such an exercise.

We are emphasising *what* rather than *how*, perhaps more than in any previous exercise.

ATM machine

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.

INVARIANTS

inv1: $customer \subseteq ACCOUNT$

inv2: $card(customer) \leq 1$

inv3: $response \in RESPONSE$

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

inv5: $finite(balance)$

inv6: $card(balance) \leq 1$

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

inv8: $finite(money)$

inv9: $card(money) \leq 1$

EVENTS

Initialisation

begin

act1: $customer := \emptyset$

act2: $response := \emptyset$

act3: $balance := \emptyset$

act4: $money := \emptyset$

end

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

Event *Withdraw* $\hat{=}$

Make withdrawal from ATM

any *amount*

when

grd1: $customer \neq \emptyset$

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

then

act1: $response, money, balance : |$
 $response' \in \{\{OK\}, \{REFUSED\}\}$
 $\wedge balance' \subseteq \mathbb{Z} \wedge finite(balance') \wedge (response' = \{OK\}$
 $\Rightarrow money' = \{amount\} \wedge balance' \in \mathbb{P}(\mathbb{Z}))$
 $\wedge card(balance') \leq 1$
 $\wedge (response' = \{REFUSED\}$
 $\Rightarrow money' = \emptyset \wedge balance' = \emptyset)$

end

Event *RemoveCard* $\hat{=}$

Customer terminates session

when

grd1: $customer \neq \emptyset$

then

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

act2: $customer := \emptyset$

end

Event *ResetResponse* $\hat{=}$

Reset response when no customer using ATM

when

grd1: $customer = \emptyset$

grd2: $response \neq \emptyset$

then

act1: $response := \emptyset$

end

Event *ResetUI* $\hat{=}$

Reset User Interface

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

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: $GENSCARD \in ACCOUNT \mapsto SCARD$

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.

2.3 ATMRO

We show two stages in refinement of the ATM. The first attempt, ATMRO, 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 ATMRO

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 or contain a single value. *money*

INVARIANTS

inv1: $customer \in \mathbb{P}(validaccounts)$

EVENTS

Initialisation

begin

act1: $customer := \emptyset$

act2: $response := \emptyset$

act3: $balance := \emptyset$

act4: $money := \emptyset$

end

Event *InsertCard_ok* $\hat{=}$

Insert service card into ATM

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*

grd6: *pass* = *password(account)*

then

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

act2: *customer* := {*account*}

end

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

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

Event *Withdraw_nok* $\hat{=}$

refines *Withdraw*

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: $customer := \emptyset$

Customer terminates session

end

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

Event *ResetUI* $\hat{=}$

Reset User Interface

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

2.4 Password Encryption

In *ATMR0* 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: $ENCRYPT \in PASSWORD \rightarrow CRYPT$

END

CONTEXT CardStatus1

Adds encrypted passwords

EXTENDS CardStatus

CONSTANTS *cryptpass* We will store encrypted passwords, not plain passwords

AXIOMS

axm1: $cryptpass \in validaccounts \rightarrow CRYPT$

axm2: $\forall acc \cdot acc \in validaccounts \Rightarrow cryptpass(acc) = ENCRYPT(password(acc))$

THEOREMS

thm1: $\forall acc, pass \cdot acc \in validaccounts$
 $\Rightarrow (pass = password(acc))$
 $\Rightarrow ENCRYPT(pass) = cryptpass(acc)$

END

MACHINE ATMR1

REFINES ATMRO

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: $customer := \emptyset$

act2: $response := \emptyset$

act3: $balance := \emptyset$

act4 : $money := \emptyset$

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  $\in \{\emptyset\} \cup \{n \cdot n \in \mathbb{Z} | \{n\}\}$ 
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}

```

Customer terminates session

act2 : *customer* := \emptyset

end

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

Event *ResetUI* $\hat{=}$

Reset User Interface

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