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.
Objectives of this Lecture

• to demonstrate that nondeterminism can be closer to reality than determinism.
• to illustrate the above using a simple ATM example.
Objectives of this Lecture

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

axm1: \( \text{finite}(\text{ACCOUNT}) \)

axm4: \( \text{RESPONSES} = \{\text{OK}, \text{REFUSED}\} \)

axm5: \( \text{OK} \neq \text{REFUSED} \)

axm6: \( \text{RESPONSE} = \{\{\text{OK}\}, \{\text{REFUSED}\}, \emptyset\} \)

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 ⊆ ACCOUNT
inv2: finite(accounts)
inv3: password ∈ accounts → PASSWORD
inv4: balance ∈ accounts → ℤ
inv5: customer ⊆ accounts
inv6: card(customer) ≤ 1
inv7: response ∈ RESPONSE
EVENTS

Initialisation

begin

act1:  accounts := ∅
act2:  password := ∅
act3:  balance := ∅
act4:  customer := ∅
act5:  response := ∅

end
ATM0: A Simplistic Model of an ATM

Event \( \text{InsertCard} \triangleq \)

any account

pass

when

\( \text{grd1}: \) account \( \in \) ACCOUNT

\( \text{grd2}: \) pass \( \in \) PASSWORD

\( \text{grd3}: \) customer \( = \) \( \emptyset \)

\( \text{grd4}: \) response \( = \) \( \emptyset \)
then
act1: 

response, customer : |
(account ∈ accounts ∧ pass = password(account) ⇒ response' = \{OK\} ∧ customer' = \{account\}) ∧ ((account ∉ accounts ∨ pass ≠ password(account)) ⇒ response' = \{REFUSED\} ∧ customer' = ∅)
end
Event $WithDraw \triangleq$

any $amount$

account

when

$grd1$: $response = \emptyset$

$grd2$: $customer \neq \emptyset$

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

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

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

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

end
Event  \( ResetResponse \triangleq \)

when

\( grd1: \)  \( response \neq \emptyset \)

then

\( act1: \)  \( response \leftarrow \emptyset \)

end

END

Resets response
Improving the Model

This $ATM_0$ 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.
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.
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: \( \text{customer} \subseteq \text{ACCOUNT} \)
inv2: \( \text{card(customer)} \leq 1 \)
inv3: \( \text{response} \in \text{RESPONSE} \)
inv4: \( \text{balance} \subseteq \mathbb{Z} \)
inv5: \( \text{finite(balance)} \)
inv6: \( \text{card(balance)} \leq 1 \)
inv7: \( \text{money} \subseteq \mathbb{N} \)
inv8: \( \text{finite(money)} \)
inv9: \( \text{card(money)} \leq 1 \)
ATM machine III

EVENTS

Initialisation

begin

act1: \textit{customer} := \emptyset

act2: \textit{response} := \emptyset

act3: \textit{balance} := \emptyset

act4: \textit{money} := \emptyset

end
ATM machine IV

Event $\text{InsertCard} \triangleq$ Insert service card into ATM

any \hspace{1em} scard

pass

when

grd1: customer = ∅
grd2: response = ∅
grd3: scard $\in$ SCARD
grd4: pass $\in$ PASSWORD

then

act1: response, customer : |

response' $\in$ \{\{OK\}, \{REFUSED\}\} \land \text{customer}' $\in$ $\mathbb{P}$ (ACCOUNT) $\land$ (response' = \{OK\})

$\Rightarrow$ customer' = \{GENSCARD$^{-1}$(scard)\}

$\land$ (response' = \{REFUSED\}) $\Rightarrow$ customer' = ∅)

end
ATM machine V

Event $Withdraw \triangleq$

any $amount$

when

$grd1$: $customer \neq \emptyset$

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

then

$act1$: $response, money, balance : |

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

end
ATM machine VI

Event \( \text{RemoveCard} \) \( \Rightarrow \) Customer terminates session

when

\( \text{grd1: } \) customer \( \neq \emptyset \)

then

\( \text{act1: } \) response := \( \{ \text{OK} \} \)

\( \text{act2: } \) customer := \( \emptyset \)

end
Event $ResetResponse \triangleq$ Reset response when no customer using ATM

when

$grd1$: $customer = \emptyset$

$grd2$: $response \neq \emptyset$

then

$act1$: $response := \emptyset$

end
ATM machine VIII

Event $\text{ResetUI} \triangleq$

when

grd1: $\text{customer} \neq \emptyset \Rightarrow \text{money} \neq \emptyset$
grd2: $\text{customer} \neq \emptyset \Rightarrow \text{balance} \neq \emptyset$
grd3: $\text{customer} \neq \emptyset \Rightarrow \text{response} \neq \emptyset$

then

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

end

END
## CardStatus context

**CONTEXT** CardStatus  
**EXTENDS** ServiceCards  

**SETS**  
`CARDSTATUS`

**CONSTANTS**  
`validaccounts`, `currentbalance`, `withdrawlimit`, `password`, `CARDOK`, `CARDNOK`

**AXIOMS**  
1. `validaccounts` $\subseteq$ `ACCOUNT`  
2. `currentbalance` $\in$ `validaccounts` $\rightarrow$ $\mathbb{Z}$  
3. `withdrawlimit` $\in$ `validaccounts` $\rightarrow$ $\mathbb{N}$  
4. `CARDSTATUS` $=$ `{CARDOK, CARDNOK}`  
5. `CARDOK` $\neq$ `CARDNOK`  
6. `password` $\in$ `validaccounts` $\rightarrow$ `PASSWORD`

END
ServicCard 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.
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.
ATM: A Simplistic Model of an ATM

INvariants

inv1: \( customer \in \mathcal{P}(validaccounts) \)
EVENTS

Initialisation

begin

act1: \textit{customer} := \emptyset \\
act2: \textit{response} := \emptyset \\
act3: \textit{balance} := \emptyset \\
act4: \textit{money} := \emptyset \\
end
Event $\textit{InsertCard}_{\text{ok}} \equiv \text{Insert service card into ATM}$
refines $\textit{InsertCard}$

any $\textit{scard}$

$\textit{pass}$

$\textit{account}$

when

$\textit{grd1}: \textit{customer} = \emptyset$

$\textit{grd2}: \textit{response} = \emptyset$

$\textit{grd3}: \textit{scard} \in \textit{SCARD}$

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

$\textit{grd5}: \textit{account} \in \textit{validaccounts}$

$\textit{grd6}: \textit{pass} = \text{password}(\textit{account})$

then

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

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

end
ATMR0 V

Event $\text{InsertCard}_{\text{\text{\_nok}}} \triangleq$

refines $\text{InsertCard}$

any $\text{scard}$

$\text{pass}$

$\text{account}$

when

$\text{grd}1$: $\text{customer} = \emptyset$

$\text{grd}2$: $\text{response} = \emptyset$

$\text{grd}3$: $\text{scard} \in \text{SCARD}$

$\text{grd}4$: $\text{account} = \text{GENSCARD}^{-1}(\text{scard})$

$\text{grd}5$: $\text{account} \in \text{validaccounts} \Rightarrow \text{pass} \neq \text{password}(\text{account})$

then

$\text{act}1$: $\text{response} := \{\text{REFUSED}\}$

end
ATMR0 VI

Event \( \text{Withdraw}_{-} \text{ok} \) \( \doteq \) Make withdrawal from ATM

refines \( \text{Withdraw} \)

any \( \text{amount} \)

account

when

\( \text{grd1:}\) \( \text{customer} \neq \emptyset \)

\( \text{grd2:}\) \( \text{amount} \in \mathbb{N} \)

\( \text{grd3:}\) \( \text{customer} = \{\text{account}\} \)

\( \text{grd4:}\) \( \text{amount} \leq \text{withdrawlimit}(\text{account}) \)

then

\( \text{act1:}\) \( \text{response} := \{\text{OK}\} \)

\( \text{act2:}\) \( \text{balance} \in \{\emptyset\} \cup \{n \cdot n \in \mathbb{Z}|\{n\}\} \)

\( \text{act3:}\) \( \text{money} := \{\text{amount}\} \)

end
ATMR0 VII

Event \( \text{Withdraw\_nok} \equiv \)
refines \( \text{Withdraw} \)
  any \( \text{amount} \)
  \( \text{account} \)
when
  \( \text{grd1}: \text{customer} \neq \emptyset \)
  \( \text{grd2}: \text{amount} \in \mathbb{N} \)
  \( \text{grd3}: \text{customer} = \{ \text{account} \} \)
  \( \text{grd4}: \text{amount} > \text{withdrawlimit(} \text{account} \text{)} \)
then
  \( \text{act1}: \text{response} := \{ \text{REFUSED} \} \)
  \( \text{act2}: \text{balance} := \emptyset \)
  \( \text{act3}: \text{money} := \emptyset \)
end
Event $RemoveCard \Rightarrow$ Customer terminates session
refines $RemoveCard$
when
$grd1: \text{customer} \neq \emptyset$
then
$act1: \text{response} := \{\text{OK}\}$
$act2: \text{customer} := \emptyset$
end
Event $ResetResponse \overset{\trianglelefteq}{=} \text{Reset response when no customer using ATM}$

refines $ResetResponse$

when

$grd1$: $\text{customer} = \emptyset$

$grd2$: $\text{response} \neq \emptyset$

then

$act1$: $\text{response} := \emptyset$

end
Event \( \text{ResetUI} \) refines \( \text{ResetUI} \)

when

\( \text{grd1}: \quad \text{customer} \neq \emptyset \Rightarrow \text{money} \neq \emptyset \)

\( \text{grd2}: \quad \text{customer} \neq \emptyset \Rightarrow \text{balance} \neq \emptyset \)

\( \text{grd3}: \quad \text{customer} \neq \emptyset \Rightarrow \text{response} \neq \emptyset \)

then

\( \text{act1}: \quad \text{money} := \emptyset \)

\( \text{act2}: \quad \text{balance} := \emptyset \)

\( \text{act3}: \quad \text{response} := \emptyset \)

end

END
Password Encryption

In *ATMR0* we model the mapping from account to password with a function \( \text{accounts} \rightarrow \text{PASSWORD} \).

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 \( \text{CheckPassword} \) 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 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.
Password Encryption

In \textit{ATM}0, we model the mapping from account to password with a function \textit{accounts} $\rightarrow$ \textit{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 \textit{Encryption}.

We also specify the operation \textit{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 *ATMR0* we model the mapping from account to password with a function $\textit{accounts} \rightarrow \textit{PASSWORD}$.

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 $\textit{Encryption}$.

We also specify the operation $\textit{CheckPassword}$ 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
$axm1: \quad ENCRYPT \in PASSWORD \rightarrow CRYPT$
END
CONTEXT CardStatus1
EXTENDS CardStatus
CONSTANTS
  cryptpass
  We will store encrypted passwords, not plain passwords

AXIOMS
  axm1: cryptpass ∈ validaccounts → CRYPT
  axm2: ∀acc·acc ∈ validaccounts ⇒ cryptpass(acc) = ENCRYPT(password(acc))

THEOREMS
  thm1: ∀acc, pass·acc ∈ validaccounts
  ⇒ (pass = password(acc)
  ⇒ ENCRYPT(pass) = cryptpass(acc))
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: \textit{customer} := \emptyset

act2: \textit{response} := \emptyset

act3: \textit{balance} := \emptyset

act4: \textit{money} := \emptyset

end
Event \( \text{InsertCard}_\text{ok} \) \( \supseteq \) Insert service card into ATM

refines \( \text{InsertCard}_\text{ok} \)

any scard, pass, account

when

grd1: \( \text{customer} = \emptyset \)
grd2: \( \text{response} = \emptyset \)
grd3: \( \text{scard} \in \text{SCARD} \)
grd4: \( \text{account} = \text{GENSCARD}^{-1}(\text{scard}) \)
grd5: \( \text{account} \in \text{validaccounts} \)
grd6: \( \text{ENCRYPT}(\text{pass}) = \text{cryptpass}(\text{account}) \)

then

act1: \( \text{response} := \{\text{OK}\} \)
act2: \( \text{customer} := \{\text{account}\} \)

end
Event \( \text{InsertCard\_nok} \) \( \triangleq \) refines \( \text{InsertCard\_nok} \)

any \( \text{scard} \)

pass

account

when

\( \text{grd1} : \text{customer} = \emptyset \)

\( \text{grd2} : \text{response} = \emptyset \)

\( \text{grd3} : \text{scard} \in \text{SCARD} \)

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

\( \text{grd5} : \text{account} \in \text{validaccounts} \Rightarrow \text{ENCRYPT}(\text{pass}) \neq \text{cryptpass}(\text{account}) \)

then

\( \text{act1} : \text{response} := \{\text{REFUSED}\} \)

end
Event $\text{Withdraw\_ok} \cong$ Make withdrawal from ATM

refines $\text{Withdraw\_ok}$

any $\text{amount}$

$\text{account}$

when

$\text{grd1: } \text{customer} \neq \emptyset$

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

$\text{grd3: } \text{customer} = \{\text{account}\}$

$\text{grd4: } \text{amount} \leq \text{withdrawlimit}(\text{account})$

then

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

$\text{act2: } \text{balance} \in \{\emptyset\} \cup \{n \cdot n \in \mathbb{Z} | \{n\}\}$

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

end
Event $\text{Withdraw\_nok} \triangleright$
refines $\text{Withdraw\_nok}$

any $\text{amount}$

account

when

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

then

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

end
Event $\text{RemoveCard} \supseteq$ Customer terminates session
refines $\text{RemoveCard}$
when
$\text{grd1}: \text{customer} \neq \emptyset$
then
$\text{act1}: \text{response} := \{OK\}$
$\text{act2}: \text{customer} := \emptyset$
end
Event  \( \text{ResetResponse} \triangleq \) Reset response when no customer using ATM

refines  \( \text{ResetResponse} \)

when

\( \text{grd1} : \ customer = \emptyset \)

\( \text{grd2} : \ response \neq \emptyset \)

then

\( \text{act1} : \ response := \emptyset \)

end
Event $\textit{ResetUI} \models$

refines $\textit{ResetUI}$

when

grd1: $\textit{customer} \neq \emptyset \Rightarrow \textit{money} \neq \emptyset$

grd2: $\textit{customer} \neq \emptyset \Rightarrow \textit{balance} \neq \emptyset$

grd3: $\textit{customer} \neq \emptyset \Rightarrow \textit{response} \neq \emptyset$

then

act1: $\textit{money} := \emptyset$

act2: $\textit{balance} := \emptyset$

act3: $\textit{response} := \emptyset$

end

END