Automated translation of VDM-SL to JML annotated Java

Technical Report

Peter W. V. Tran-Jørgensen

Aarhus University - Department of Engineering

AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

Table of Contents

Introduction

ATM Example

This appendix includes the complete version of the VDM-SL ATM example used to demonstrate the Java Modeling Language (JML) translation presented in [1] as well as this technical report.

```
module ATM

imports from IO all
exports all

definitions

state St of
 validCards : set of Card
 currentCard : [Card]
 pinOk : bool
 accounts : map AccountId to Account
 init St == St = mk_St({},nil,false,{|->})
 inv mk_St(v,c,p,a) ==
  (p or c <> nil => c in set v)
  and
  forall id1, id2 in set dom a &
   id1 <> id2 =>
   a(id1).cards inter a(id2).cards = {}
end

types

Card ::
 id : nat
 pin : Pin;

Pin = nat
inv p == 0 <= p and p <= 9999;

AccountId = nat
inv id == id > 0;

Account ::
 cards : set of Card
 balance : real
 inv a == a.balance >= -1000;
```

```
Amount = nat1
inv a == a < 2000;

functions

TotalBalance : set of Account -> real
TotalBalance (acs) ==
 if acs = {} then
   0
 else
  let a in set acs
  in
    a.balance + TotalBalance(acs \ {a})
measure TotalBalanceMes;

TotalBalanceMes: set of Account +> nat
TotalBalanceMes(acs) == card acs;

operations

GetStatus : () ==> bool * seq of char
GetStatus () ==
if currentCard <> nil then
  if pinOk then
    return mk_(false, "transaction in progress.")
  else
    return mk_(false, "debit card inserted. Awaiting pin code.")
else
  return mk_(true,"no debit card is currently inserted into the machine.");

OpenAccount : set of Card * AccountId ==> ()
OpenAccount (cards,id) ==
 accounts := accounts munion {id |-> mk_Account(cards,0.0)}
pre id not in set dom accounts
post id in set dom accounts and
     accounts(id).balance = 0;

AddCard : Card ==> ()
AddCard (c) ==
 validCards := validCards union {c}
pre c not in set validCards
post c in set validCards;

RemoveCard : Card ==> ()
RemoveCard (c) ==
 validCards := validCards \ {c}
pre c in set validCards
post c not in set validCards;

InsertCard : Card ==> <Accept>|<Busy>|<Reject>
InsertCard (c) ==
if c in set validCards then
(
 currentCard := c;
 return <Accept>;
)
elseif currentCard <> nil then
  return <Busy>
else
  return <Reject>
```

```
pre currentCard = nil
post
 if RESULT = <Accept> then
  currentCard = c
 else if RESULT = <Busy> then
   currentCard = currentCard~
 else currentCard = nil;

Display : seq of char ==> ()
Display (msg) ==
  IO`println(msg);

NotifyUser : <Accept>|<Busy>|<Reject> ==> ()
NotifyUser (outcome) ==
if outcome = <Accept> then
  Display("Card accepted")
elseif outcome = <Busy> then
  Display("Another card has already been inserted")
else if outcome = <Reject> then
  Display("Unknown card")
else
  error;

EnterPin : Pin ==> ()
EnterPin (pin) ==
 pinOk := (currentCard.pin = pin)
pre currentCard <> nil;

ReturnCard : () ==> ()
ReturnCard () ==
atomic
(
 currentCard := nil;
 pinOk := false;
)
pre currentCard <> nil
post currentCard = nil and not pinOk;

Withdraw : AccountId * Amount ==> real
Withdraw (id, amount) ==
let newBalance = accounts(id).balance - amount
in
(
 accounts(id).balance := newBalance;
 return newBalance;
)
pre currentCard in set validCards and pinOk and
    currentCard in set accounts(id).cards and
    id in set dom accounts
post
let accountPre = accounts~(id),
    accountPost = accounts(id)
in
 accountPre.balance = accountPost.balance + amount and
 accountPost.balance = RESULT;

Deposit : AccountId * Amount ==> real
Deposit (id, amount) ==
let newBalance = accounts(id).balance + amount
in
```

```
(
 accounts(id).balance := newBalance;
 return newBalance;
)
pre pre_Withdraw(id,amount,St)
post
let accountPre = accounts~(id),
    accountPost = accounts(id)
in
 accountPre.balance + amount = accountPost.balance and
 accountPost.balance = RESULT;

PrintAccount: AccountId ==> ()
PrintAccount(id) ==
let balance = accounts(id).balance
in
 IO`printf("Balance is for account %s is %s\n", [id,balance]);

GetCurrentCardId : () ==> [nat]
GetCurrentCardId () ==
 if currentCard <> nil then
   return currentCard.id
 else
   return nil;

--
-- Test operations
--

TestCurrentCardId : () ==> [nat]
TestCurrentCardId () ==
let id = GetCurrentCardId()
in
  return id;

TestStatus : () ==> real
TestStatus () ==
let accId = 1,
   c1 = mk_Card(1,1234)
in
(

 AddCard(c1);
 OpenAccount({mk_Card(1,1234)},accId);

 let status = GetStatus(),
     awaitingCard = status.#1,
     msg = status.#2
  in
  (
    IO`println("Message: " ^ msg);
    if awaitingCard and <Accept> = InsertCard(c1) then
    (
      NotifyUser(<Accept>);
      EnterPin(1234);
      Deposit(accId,100);
    );
  );

  return 0;
```

```
);

TestWithdraw : () ==> real
TestWithdraw () ==
let accId = 1,
    cardId = 1,
    pin = 1234,
   c1 = mk_Card(cardId,pin)
in
(

 AddCard(c1);
 OpenAccount({mk_Card(1,1234)},accId);

 if InsertCard(c1) = <Accept> then
 (
   EnterPin(pin);
   let expense = 600,
     profit = 100
   in
     let amount : nat1 = expense - profit
     in
       Withdraw(accId, amount);
 );

 error;
);

TestTotalBalance : () ==> real
TestTotalBalance () ==
let card1 = mk_Card(1,1234),
    card2 = mk_Card(2,5678),
    ac1 = mk_Account({card1}, 1000),
    ac2 = mk_Account({card2}, 500)
in
  TotalBalance({ac1,ac2});

TestScenario : () ==> ()
TestScenario() ==
let accId1 : AccountId = 1,
    pin1 = 1234,
    card1 = mk_Card(1, pin1),
    pin2 = 2345,
    card2 = mk_Card(2, pin2)
in
(
 AddCard(card1);
 AddCard(card2);
 OpenAccount({card1, card2},accId1);
 let - = InsertCard(card2) in skip;
 PrintAccount(accId1);
 EnterPin(2345);
 let - = Deposit(accId1, 200) in skip;

 PrintAccount(accId1);

 ReturnCard();
 RemoveCard(card1);
 RemoveCard(card2);
);
```

```
end ATM
```

Bibliography

[1] Tran-Jørgensen, P.W.V., Larsen, P.G., Leavens, G.T.: Automated translation of VDM to JML annotated Java (Jan 2016 Submitted to the International Journal on Software Tools for Technology Transfer (STTT))