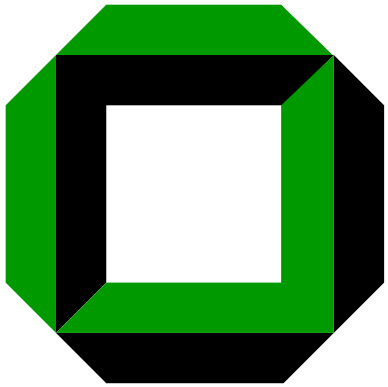
The Mondex Case Study
Verifying a Java Implementation

Peter H. Schmitt, Isabel Tonin

Institute for Theoretical Computer Science
Department of Computer Science
Universität Karlsruhe (TH)



KeY Symposium, June, 2007

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs
2. A coherent tool set

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs
2. A coherent tool set

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs
2. A coherent tool set
   automating the theory and scaling up to large code

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs
2. A coherent tool set
   automating the theory and scaling up to large code
3. A repository of verified programs.
   Contains at this time mostly contributions to the Mondex Case Study.

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs
2. A coherent tool set
   automating the theory and scaling up to large code
3. A repository of verified programs.
   Contains at this time mostly contributions to the Mondex Case Study.

Verified Software Grand Challenge

a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
   covering all features needed to built practical and reliable programs
2. A coherent tool set
   automating the theory and scaling up to large code
3. A repository of verified programs.
   Contains at this time mostly contributions to the Mondex Case Study.

You can't say any more it can't be done.
Here, we've done it!

The Mondex Card

- Smart card for electronic financial transactions

The Mondex Card

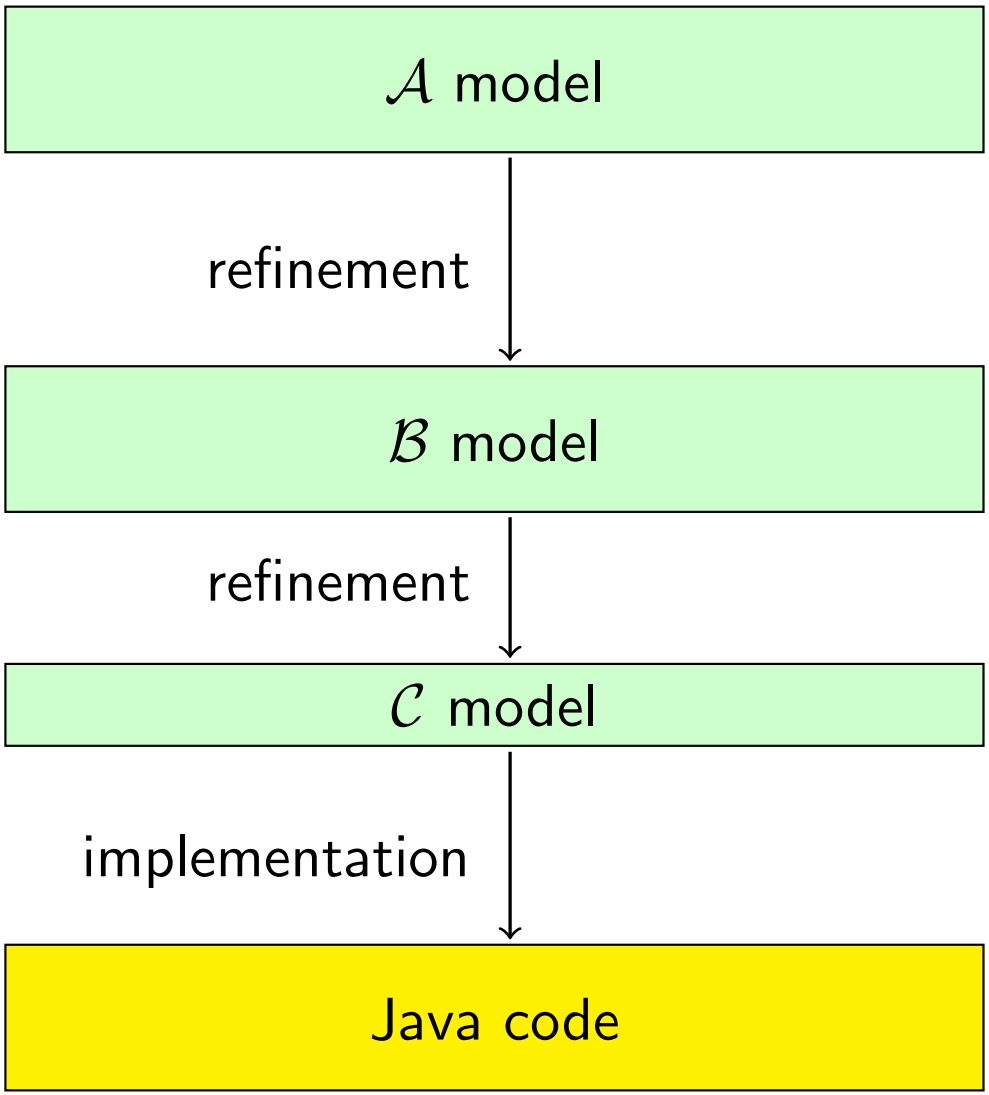
- Smart card for electronic financial transactions
- Issued by Natwest in 1996

The Mondex Card

- Smart card for electronic financial transactions
- Issued by Natwest in 1996
- First product certified to ITSEC Level E6

The Mondex Card

- Smart card for electronic financial transactions
- Issued by Natwest in 1996
- First product certified to ITSEC Level E6
- Sanitised documentation publicly available



Previous Work using **Z**, ASM, RSL, Alloy

Our Contribution using JML

Our Contribution

- Reference Implementation in Java Card

Our Contribution

- Reference Implementation in Java Card
- Specification using Design by Contract paradigm

Our Contribution

- Reference Implementation in Java Card
- Specification using Design by Contract paradigm
- Annotation using Java Modeling Language (JML)

Our Contribution

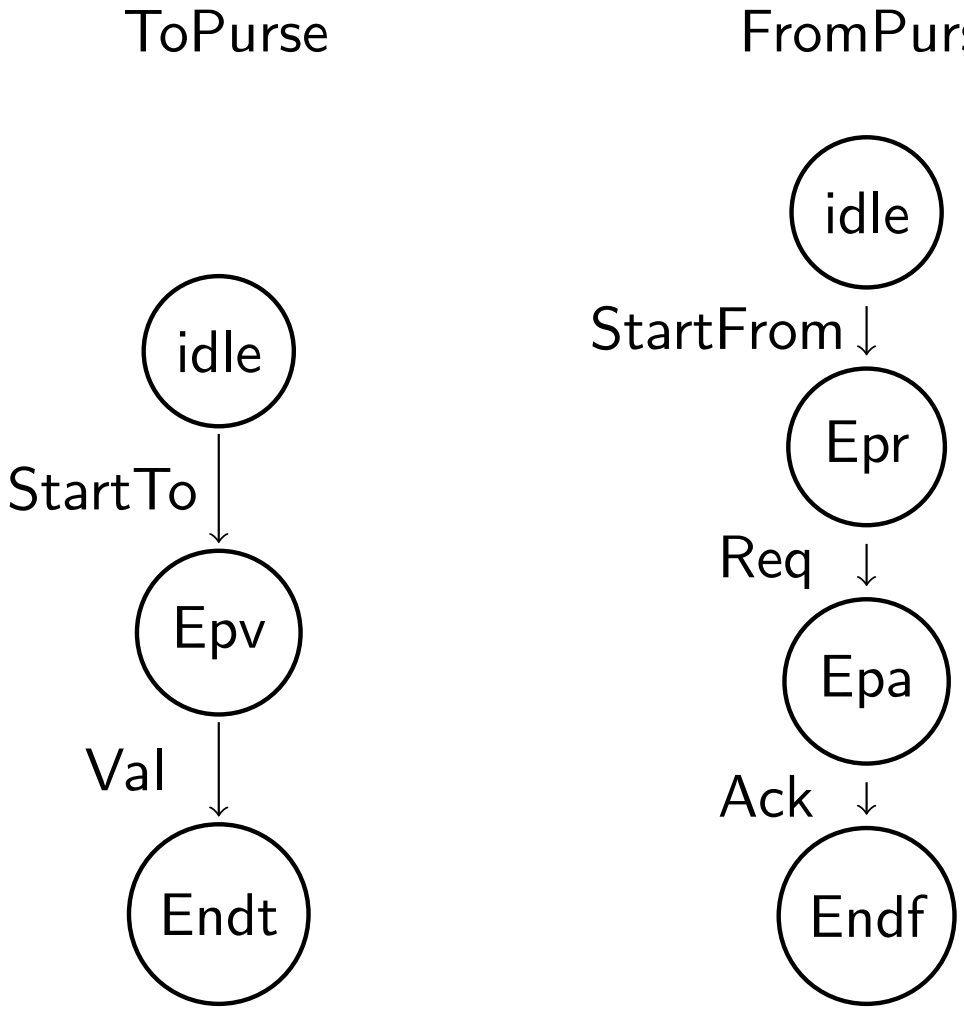
- Reference Implementation in Java Card
- Specification using Design by Contract paradigm
- Annotation using Java Modeling Language (JML)
- Full verification using the KeY prover

The Principal Classes of Mondex Card

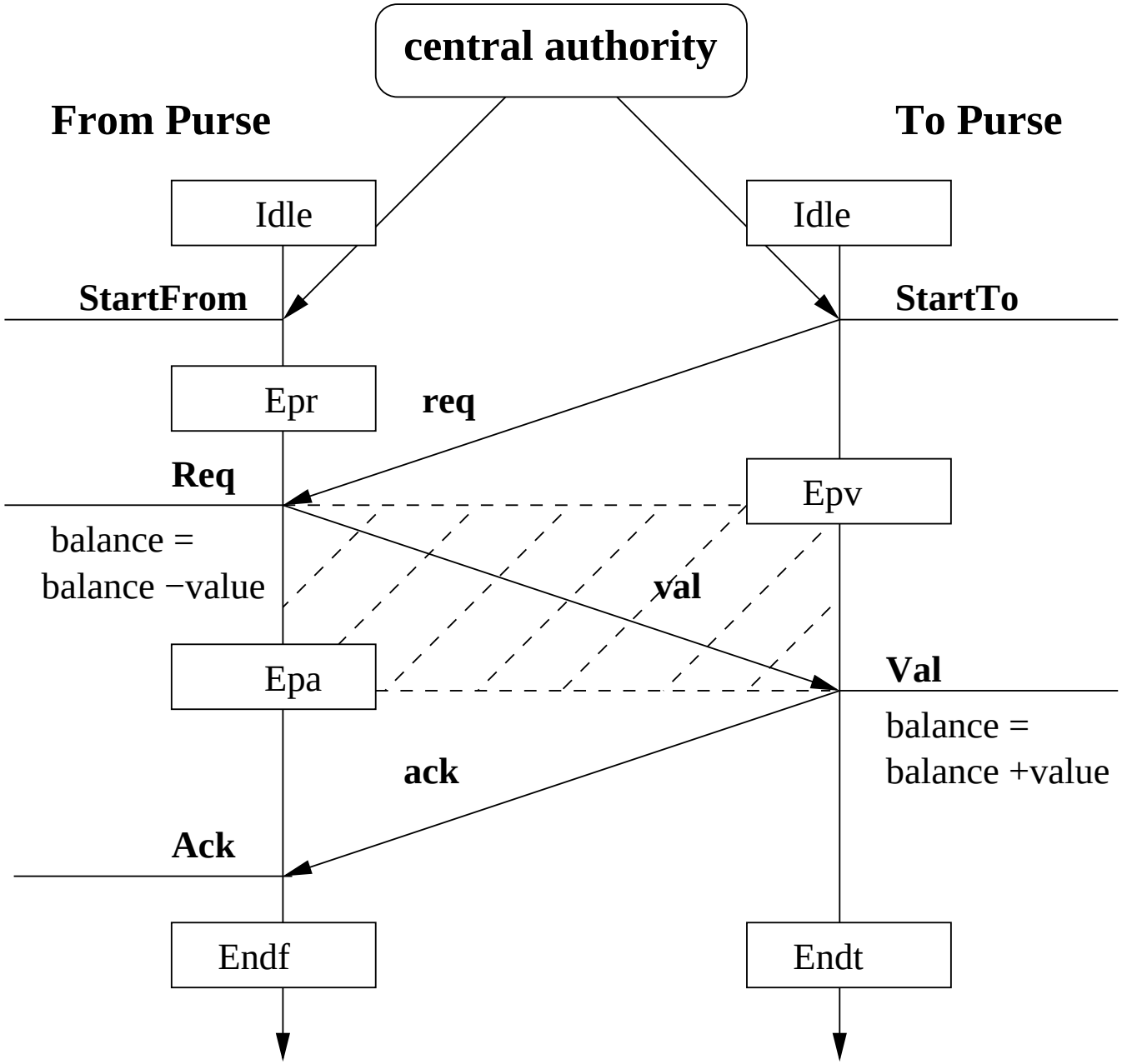
```
public class ConPurseJC extends Applet
{ private short name;
  private short balance;
  private byte status;
  private PayDetails transaction;
  private short nextSeq;
  private PayDetails [] exLog;
  private byte logIdx;
... }

public class PayDetails
{ short fromName;
  short toName;
  short value;
  short fromSeq;
  short toSeq;
... }
```

Mondex Protocol

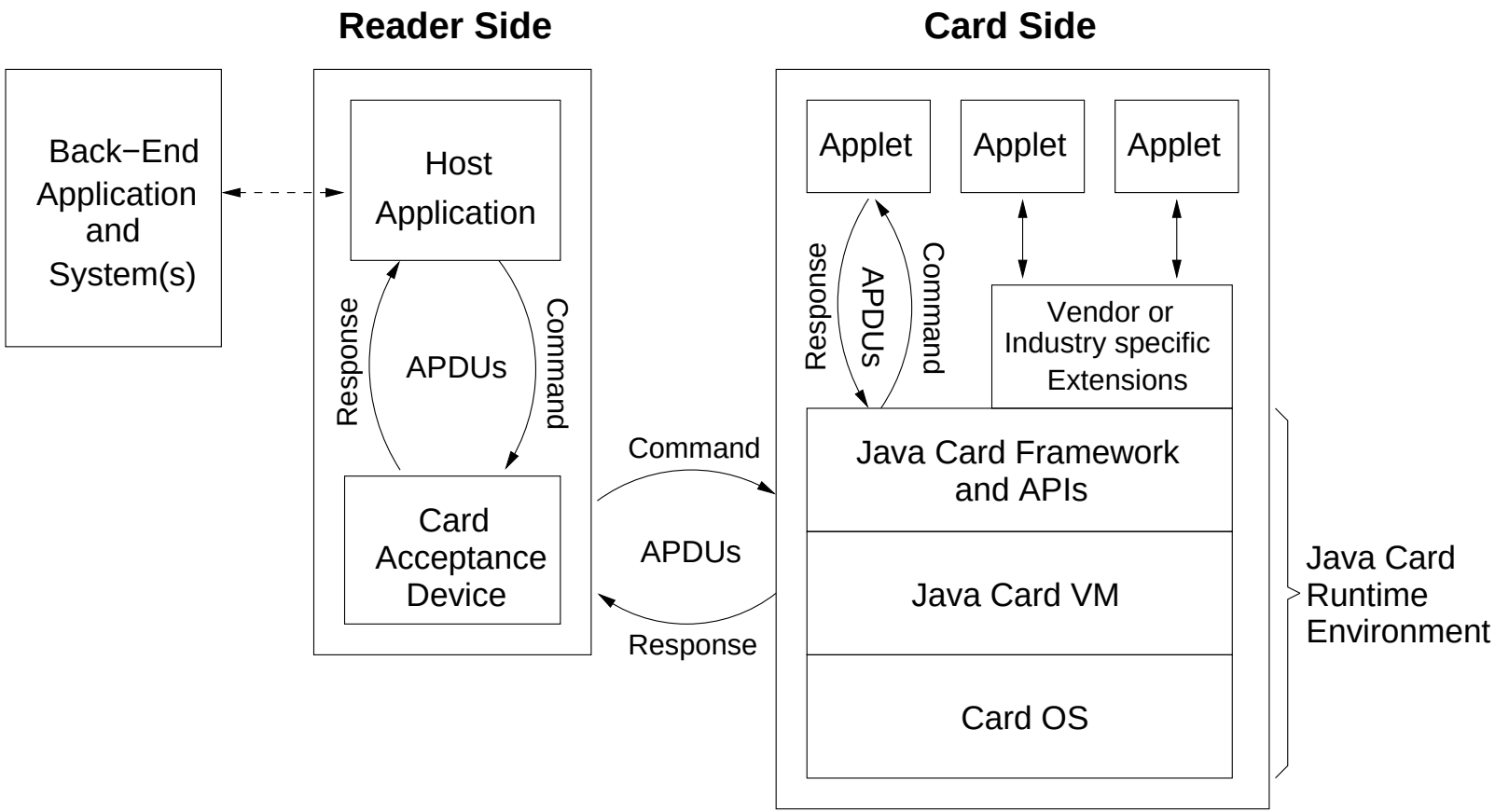
Automata View



The Protocol (Modified)



Architecture of a Java Card Application



Z Specification of the Val Operation

ValPurseOkay

$\Delta ConPurse$

$m?, m! : Message$

$AuthenticValMessage$

$status = epv$

$\Xi ConPurseVal$

$balance' = balance + pdAuth.value$

$status' = esTo$

$m! = ackpdAuth$

ASM Specification of the Val Operation

VAL#

if $msg = val(pdAuth(receiver)) \wedge \neg\ fail?$

then $balance(receiver) :=$
$\quad balance(receiver) + pdAuth(receicer).value$
$state(receiver) := idle$
$outmsg := ack(pdAuth(receiver))$

else $outmsg := \perp$

JML Specification of the Val Operation

```
/*@ public behavior
1 @ requires apdu != null;
2 @ assignable balance, status;
  @ ensures
3 @ (balance == \old(balance)
  @                + transaction.value) &&
  @ (\old(status) == Epv) && (status == Endt);
  @ signals_only ISOException;
  @ signals (ISOException e)
4 @ ((balance == \old(balance))
  @   && (status == \old(status)));
  @*/
  private void val_operation(APDU apdu)
                  throws ISOException
```

JML keyword in red.

Top Level ASM Specification

BOP#

choose $msg, fail?, rec$ **with** $msg \in ether \wedge auth(rec)$ **in**
  if $isStartTo(msg) \wedge state(rec) = idle$ **then** $STARTO\#$
  else if $isStartFrom(msg) \wedge state(rec) = idle$
    then $STARTFROM\#$
  else if $isreq(msg) \wedge state(rec) = epr$ **then** $REQ\#$
  else if $isval(msg) \wedge state(rec) = epv$ **then** $VAL\#$
  else if $isack(msg) \wedge state(rec) = epa$ **then** $ACK\#$
  else $ABORT\#$
  seq $ether := ether + + outmsg$

Top Level ASM Specification

BOP#

choose $msg, fail?, rec$ **with** $msg \in ether \wedge auth(rec)$ **in**
  if $isStartTo(msg) \wedge state(rec) = idle$ **then** $STARTO\#$
  else if $isStartFrom(msg) \wedge state(rec) = idle$
    then $STARTFROM\#$
  else if $isreq(msg) \wedge state(rec) = epr$ **then** $REQ\#$
  else if $isval(msg) \wedge state(rec) = epv$ **then** $VAL\#$
  else if $isack(msg) \wedge state(rec) = epa$ **then** $ACK\#$
  else $ABORT\#$
  seq $ether := ether + +outmsg$

Top Level JML Specification

First Installment

```
/*@ public behavior
  @ requires apdu != null;
  @ assignable ...
  @ ensures
  @ ((\old(logIdx) != logIdx) ==>
  @   ((logIdx==0) &&
  @    (status==Idle) &&
  @    (\old(status)==Idle)))
  @  &&
  @ ((\old(status)==status) ==>
  @   (\old(balance)==balance) &&
  @     (\old(nextSeq)==nextSeq))
  @  &&
```

Top Level JML Specification

Second Installment

```
&&
 @    ((\old(status)!=status) ==>
 @
 @   \old(apdu._buffer[I.OFFSET_INS])
 @     ==    apdu._buffer[I.OFFSET_INS]

 @    && (\old(status)==Epa ==>
 @    (status==Endf &&
 @   apdu._buffer[I.OFFSET_INS]==Ack
 @      && balance==\old(balance)))
 @     &&
```

Top Level JML Specification

Third Installment

```
@ signals_only ISOException;
 @ signals (ISOException e) (
 @   \old(balance)==balance &&
 @   \old(status)==status &&
 @   \old(logIdx)==logIdx &&
 @   \old(nextSeq) == nextSeq);
 @*/
 public void process(APDU apdu)
```

Top Level Z Specification

Security Property 1 No value creation: no value may be created in the system. The sum of all purses' balance does not increase.

Top Level Z Specification

Security Property 1 No value creation: no value may be created in the system. The sum of all purses' balance does not increase.

Security Property 2.1 All value accounted: all values must be accounted in the system. The sum of all purses' balance and lost components does not change.

Top Level Z Specification

Security Property 1 No value creation: no value may be created in the system. The sum of all purses' balance does not increase.

Security Property 2.1 All value accounted: all values must be accounted in the system. The sum of all purses' balance and lost components does not change.

Security Property 2.2 Exception Logging: if a purse aborts a transfer at a point where value could be lost, then the purse logs the details.

Top Level Z Specification

Security Property 1 No value creation: no value may be created in the system. The sum of all purses' balance does not increase.

Security Property 2.1 All value accounted: all values must be accounted in the system. The sum of all purses' balance and lost components does not change.

Security Property 2.2 Exception Logging: if a purse aborts a transfer at a point where value could be lost, then the purse logs the details.

Security Property 3 Authentic purses: a transfer can only occur between authentic purses.

Top Level Z Specification

Security Property 1 No value creation: no value may be created in the system. The sum of all purses' balance does not increase.

Security Property 2.1 All value accounted: all values must be accounted in the system. The sum of all purses' balance and lost components does not change.

Security Property 2.2 Exception Logging: if a purse aborts a transfer at a point where value could be lost, then the purse logs the details.

Security Property 3 Authentic purses: a transfer can only occur between authentic purses.

Security Property 4 Sufficient Funds: a transfer can occur only if there are sufficient funds in the from purse.

JML Invariants

ensuring the sufficient funds property

```
public class ConPurseJC extends Applet
{/*@ public invariant
  @ (exLog != null) && (exLog.length>0)
  @    && ...
  @ (balance>=0) && (balance<=ShortMaxValue)
  @    && ...
  @   ((status == Epr) ==>
  @   (transaction.value <= balance))  &&
  @ ((status==Epv) ==>
  @   (transaction.value<=
  @    (ShortMaxValue - balance))) &&
  @ (\forall byte i;i>=0 && i<exLog.length;
  @          exLog[i] != null);
  @*/
... }
```

Relationship between Purse and Counterpurse

Purse o, Counterpurse x

```
Rel(o,x):
 (o.transaction == x.transaction
 && o.name != x.name)
 && ((o.status == Endf) ==>
  (x.status == Endt))
 && ((o.status == Endt) ==>
  ((x.status == Epa) || (x.status == Endf)))
 && ((status == Epa) ==>
  ((x.status == Epv) || (x.status == Endt)))
 && ((o.status == Epv) ==>
  ((x.status == Idle) || (x.status == Epr) ||
  (x.status == Epa)))
 && ((o.status == Epr) ==>
  ((x.status == Idle) || (x.status == Epv)))
```

Helper Functions

$$o.bookedValue() = \begin{cases} -o.transaction.value & \text{if} \\ \quad (o.status == Epa) \text{ or} \\ \quad (o.status == Endf) \\ +o.transaction.value & \text{if} \\ \quad o.status == Endt \\ 0 \quad \text{otherwise} \end{cases}$$

$$o.loss() = \begin{cases} o.transaction.value & \text{if} \\ \quad (o.status == Epa) \text{ or} \\ \quad\quad (o.status == Endf) \\ \text{and} \\ \quad (x.status == Epa) \text{ or} \\ \quad\quad (x.status == Endf) \\ 0 \quad \text{otherwise} \end{cases}$$

Constraint on `bookedValue()`

ConPurseJC:

```
/*@ public constraint
  @ ((\old(balance) != balance) ==>
  @  ((balance -\old(balance))
  @       ==bookedValue()));
  @*/
```

All Values Accounted Property

We need to show for every purse o and its ounterpurse x

All Values Accounted Property

We need to show for every purse o and its ounterpurse x

Rel(o,x)
==>
`o.bookedValue() + x.bookedValue() + o.loss = 0`

All Values Accounted Property

We need to show for every purse o and its ounterpurse x

Rel(o,x)
==>
`o.bookedValue() + x.bookedValue() + o.loss = 0`

whenever the `process` method terminates, normally or abruptly.

Proof Statistics

Method	Nodes	Branches	Time (min)
Using Contracts			
process	4,731	54	10
showProperties	6,565	50	10
Using Implementation			
startFrom	3,818	102	5
startTo	3,975	105	5
req	3,482	95	5
val	3,525	91	5
ack	2,370	69	5
clear_ex_log	1,352	37	5
read_ex_log	28,292	490	35
abort_if_necessary	2,427	57	5

Proof Statistics

Continued

Method	Nodes	Branches	Time (min)
Strong Invariant			
startFrom	19,084	44	10
startTo	19,015	40	10
req	23,165	64	15
val	18,689	51	15
ack	14,199	32	10
clear_ex_log	7,588	18	5
abort_if_necessary	8,761	33	5

Further Statistics

- 63 pages of relevant Z specification

Further Statistics

- 63 pages of relevant Z specification
- 327 lines of Java Card code

Further Statistics

- 63 pages of relevant Z specification
- 327 lines of Java Card code
  - 2 classes

Further Statistics

- 63 pages of relevant Z specification
- 327 lines of Java Card code
  - 2 classes
  - 19 methods

Further Statistics

- 63 pages of relevant Z specification
- 327 lines of Java Card code
  - 2 classes
  - 19 methods
  - not counting API classes and methods

Further Statistics

- 63 pages of relevant Z specification
- 327 lines of Java Card code
  - 2 classes
  - 19 methods
  - not counting API classes and methods
- 185 lines of JML specification

Quote on Z

> ***Z*** *is mainly used at the specification level. Some data and operation refinement towards an implementation is possible in* ***Z****, but at some point a jump to code must be made, typically informally.*

by Jonathan Bowen,
in Software Specification Methods, Chapter 1
H.Habri and M.Frappier (eds), ISTE 2006.

Critical Issues

during jump to code

- one operation on the model level (e.g., exeception logging) might have to be realised as the combined effect of several operations of the implementation,

Critical Issues

during jump to code

- one operation on the model level (e.g., exeception logging) might have to be realised as the combined effect of several operations of the implementation,
- deployment of the implemented system on different platforms has heavy influence on the verification conditions,

Critical Issues

during jump to code

- one operation on the model level (e.g., exeception logging) might have to be realised as the combined effect of several operations of the implementation,
- deployment of the implemented system on different platforms has heavy influence on the verification conditions,
- replacing abstract data structures by programing language data types is not a refinement step,

Critical Issues

during jump to code

- one operation on the model level (e.g., exeception logging) might have to be realised as the combined effect of several operations of the implementation,
- deployment of the implemented system on different platforms has heavy influence on the verification conditions,
- replacing abstract data structures by programing language data types is not a refinement step,
- issues that require a lot of verification effort at the model level may no have a counter part in the implementation.

Critical Issues

during jump to code

- one operation on the model level (e.g., exeception logging) might have to be realised as the combined effect of several operations of the implementation,
- deployment of the implemented system on different platforms has heavy influence on the verification conditions,
- replacing abstract data structures by programing language data types is not a refinement step,
- issues that require a lot of verification effort at the model level may no have a counter part in the implementation.
- JML (and other OO specification languages) lack support for system invariants.

THE END

The Mondex Case Study

Previous contributions to the Grand Challenge repository

- Specification using **Z**,
  refinement proofs by hand and using Z/Eves.

  S. Stepney, D. Cooper, and J. Woodcock.
  Oxford University Computing Laboratory, 2000.

The Mondex Case Study

Previous contributions to the Grand Challenge repository

- Specification using **Z**,
  refinement proofs by hand and using Z/Eves.

  S. Stepney, D. Cooper, and J. Woodcock.
  Oxford University Computing Laboratory, 2000.
- Specification using ASM (Abstract State Machines),
  refinement verification with KIV

  G. Schellhorn, H. Grundy, D. Haneberg, W.Reif.
  Universität Augsburg, 2006.

The Mondex Case Study

Previous contributions to the Grand Challenge repository

- Specification using **Z**,
  refinement proofs by hand and using Z/Eves.

  S. Stepney, D. Cooper, and J. Woodcock.
  Oxford University Computing Laboratory, 2000.
- Specification using ASM (Abstract State Machines),
  refinement verification with KIV

  G. Schellhorn, H. Grundy, D. Haneberg, W.Reif.
  Universität Augsburg, 2006.
- Specification using Alloy, verification with Alloy model finder

  T. Ramananandro. École Normale Supérieure, Paris, 2006.

The Mondex Case Study

Previous contributions to the Grand Challenge repository

- Specification using **Z**,
  refinement proofs by hand and using Z/Eves.

  S. Stepney, D. Cooper, and J. Woodcock.
  Oxford University Computing Laboratory, 2000.
- Specification using ASM (Abstract State Machines),
  refinement verification with KIV

  G. Schellhorn, H. Grundy, D. Haneberg, W.Reif.
  Universität Augsburg, 2006.
- Specification using Alloy, verification with Alloy model finder

  T. Ramananandro. École Normale Supérieure, Paris, 2006.
- Specification using RSL (Raise Specification Language),
  refinement verification with PVS and SAL

  C. George, A. E. Haxthausen.
  United Nations University, Macau, 2007.