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
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
a concerted effort of the global scientific community to deliver

1. A comprehensive theory of programming
a concerted effort of the global scientific community to deliver

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

2. A coherent tool set
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

Issued by NatWest in 1996
First product certified to ITSEC Level E6
Sanitised documentation publicly available
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
A model

refinement

B model

refinement

C model

implementation

Java code

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

```java
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

ToPurse

idle

StartTo

Epv

Val

Endt

FromPurse

idle

StartFrom

Epr

Req

Epa

Ack

Endf
Architecture of a Java Card Application

The Mondex Case Study
**Val Purse Operation**

\[ \text{ValPurseOkay} \]

\[ \Delta \text{ConPurse} \]

\[ m?, m! : \text{Message} \]

\[ \text{AuthenticValMessage} \]

\[ \text{status} = \text{epv} \]

\[ \exists \text{ConPurseVal} \]

\[ \text{balance}' = \text{balance} + \text{pdAuth.value} \]

\[ \text{status}' = \text{esTo} \]

\[ m! = \text{ackpdAuth} \]
ASM Specification of the Val Operation

VAL#

if \( \text{msg} = \text{val(} \text{pdAuth(} \text{receiver} \text{)} \text{)} \land \neg \text{fail?} \) then

\( \text{balance(} \text{receiver} \text{)} := \)

\( \text{balance(} \text{receiver} \text{)} + \text{pdAuth(} \text{receiver} \text{)} \text{.value} \)

\( \text{state(} \text{receiver} \text{)} := \text{idle} \)

\( \text{outmsg} := \text{ack(} \text{pdAuth(} \text{receiver} \text{)} \text{)} \)

else

\( \text{outmsg} := \bot \)
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 ∈ ether ∧ auth(rec) in
if isStartTo(msg) ∧ state(rec) = idle then STARTO#
else if isStartFrom(msg) ∧ state(rec) = idle
    then STARTFROM#
else if isreq(msg) ∧ state(rec) = epr then REQ#
else if isval(msg) ∧ state(rec) = epv then VAL#
else if isack(msg) ∧ state(rec) = epa then ACK#
else ABORT#
seq ether := ether + +outmsg
BOP#

choose \( msg, fail?, rec \) with \( msg \in \text{ether} \land \text{auth}(\text{rec}) \) in

if \( \text{isStartTo}(msg) \land \text{state}(\text{rec}) = \text{idle} \) then \text{STARTO}#

else if \( \text{isStartFrom}(msg) \land \text{state}(\text{rec}) = \text{idle} \) then \text{STARTFROM}#

else if \( \text{isreq}(msg) \land \text{state}(\text{rec}) = \text{epr} \) then \text{REQ}#

else if \( \text{isval}(msg) \land \text{state}(\text{rec}) = \text{epv} \) then \text{VAL}#

else if \( \text{isack}(msg) \land \text{state}(\text{rec}) = \text{epa} \) then \text{ACK}#

else \text{ABORT}#

seq \( \text{ether} := \text{ether} + +\text{outmsg} \)
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)
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 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.
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.

The Mondex Case Study
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 $x$

\[
\text{Rel}(o, x):
\]
\[
(o.\ \text{transaction} == x.\ \text{transaction} \land\ o.\ \text{name} != x.\ \text{name})
\]
\[
\land (o.\ \text{status} == \text{Endf}) \Rightarrow
\]
\[
(x.\ \text{status} == \text{Endt})
\]
\[
\land (o.\ \text{status} == \text{Endt}) \Rightarrow
\]
\[
((x.\ \text{status} == \text{Epa}) \lor (x.\ \text{status} == \text{Endf}))
\]
\[
\land ((\text{status} == \text{Epa}) \Rightarrow
\]
\[
((x.\ \text{status} == \text{Epv}) \lor (x.\ \text{status} == \text{Endt}))
\]
\[
\land ((o.\ \text{status} == \text{Epv}) \Rightarrow
\]
\[
((x.\ \text{status} == \text{Idle}) \lor (x.\ \text{status} == \text{Epr}) \lor (x.\ \text{status} == \text{Epa}))
\]
\[
\land ((o.\ \text{status} == \text{Epr}) \Rightarrow
\]
\[
((x.\ \text{status} == \text{Idle}) \lor (x.\ \text{status} == \text{Epv}))
\]
**Helper Functions**

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

\[
\text{o.loss}() = \begin{cases}
  o\text{.transaction.value} & \text{if } (o\text{.status} == \text{Epa}) \text{ or } (o\text{.status} == \text{Endf}) \\
  \text{and } (x\text{.status} == \text{Epa}) \text{ or } (x\text{.status} == \text{Endf}) \\
  0 & \text{otherwise}
\end{cases}
\]
Constraint on bookedValue()

ConPurseJC:

/*@ public constraint
    @ ((\old(balance) != balance) ==> 
        @ ((balance -\old(balance)) 
        @      == bookedValue()));
    @*/
We need to show for every purse \( o \) and its counterpart \( x \)
All Values Accounted Property

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

\[ \text{Rel}(o,x) \]

\[ \implies o.\text{bookedValue}() + x.\text{bookedValue}() + o.\text{loss} = 0 \]
All Values Accounted Property

We need to show for every purse $o$ and its counterpart $x$

$$\text{Rel}(o, x)$$

$$\Rightarrow$$

$$o.\text{bookedValue}() + x.\text{bookedValue}() + o.\text{loss} = 0$$

whenever the process method terminates, normally or abruptly.
## Proof Statistics

<table>
<thead>
<tr>
<th>Method</th>
<th>Nodes</th>
<th>Branches</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Using Contracts</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>process</td>
<td>4,731</td>
<td>54</td>
<td>10</td>
</tr>
<tr>
<td>showProperties</td>
<td>6,565</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td><strong>Using Implementation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>startFrom</td>
<td>3,818</td>
<td>102</td>
<td>5</td>
</tr>
<tr>
<td>startTo</td>
<td>3,975</td>
<td>105</td>
<td>5</td>
</tr>
<tr>
<td>req</td>
<td>3,482</td>
<td>95</td>
<td>5</td>
</tr>
<tr>
<td>val</td>
<td>3,525</td>
<td>91</td>
<td>5</td>
</tr>
<tr>
<td>ack</td>
<td>2,370</td>
<td>69</td>
<td>5</td>
</tr>
<tr>
<td>clear_ex_log</td>
<td>1,352</td>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>read_ex_log</td>
<td>28,292</td>
<td>490</td>
<td>35</td>
</tr>
<tr>
<td>abort_if_necessary</td>
<td>2,427</td>
<td>57</td>
<td>5</td>
</tr>
</tbody>
</table>
### Proof Statistics

**Continued**

<table>
<thead>
<tr>
<th>Method</th>
<th>Nodes</th>
<th>Branches</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strong Invariant</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>startFrom</td>
<td>19,084</td>
<td>44</td>
<td>10</td>
</tr>
<tr>
<td>startTo</td>
<td>19,015</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>req</td>
<td>23,165</td>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>val</td>
<td>18,689</td>
<td>51</td>
<td>15</td>
</tr>
<tr>
<td>ack</td>
<td>14,199</td>
<td>32</td>
<td>10</td>
</tr>
<tr>
<td>clear_ex_log</td>
<td>7,588</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>abort_if_necessary</td>
<td>8,761</td>
<td>33</td>
<td>5</td>
</tr>
</tbody>
</table>
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
- 185 lines of JML specification
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.
one operation on the model level (e.g., exception 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., execution 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., exception 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 programming 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., exception 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 $\mathcal{Z}$,
  refinement proofs by hand and using $\mathcal{Z}$/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

- Specification using RSL (Raise Specification Language),
  refinement verification with PVS and SAL
  C. George, A. E. Haxthausen.
The Mondex Case Study

Previous contributions to the Grand Challenge repository

- Specification using $\text{Z}$, refinement proofs by hand and using $\text{Z/Eves}$.
  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 $\text{Z}$, refinement proofs by hand and using $\text{Z/Eves}$.
  S. Stepney, D. Cooper, and J. Woodcock.
  Oxford University Computing Laboratory, 2000.

- Specification using $\text{ASM (Abstract State Machines)}$, refinement verification with $\text{KIV}$
  G. Schellhorn, H. Grundy, D. Haneberg, W. Reif.
  Universität Augsburg, 2006.

- Specification using $\text{Alloy}$, verification with $\text{Alloy model finder}$
The Mondex Case Study

Previous contributions to the Grand Challenge repository

▶ Specification using $\mathbb{Z}$, 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

▶ Specification using RSL (Raise Specification Language), refinement verification with PVS and SAL
C. George, A. E. Haxthausen.