JML & ESC/Java case study:
specifying the JavaCard APDU protocol in JML

Erik Poll
The JavaCard platform

JavaCard platform (JCRE) = JavaCard language (JCVM) + JavaCard API

The JavaCard API provides

- some base classes and interfaces, eg. Applet
- some OS-like functionality, including the APDU class for communication with the smartcard terminal.
ISO 7816 and APDU’s

Smartcard and terminal communicate by exchanging APDU’s – sequences of bytes – as specified in ISO7816-4:

1. terminal sends a command APDU to smartcard

   \[
   \begin{array}{ccccccc}
   \text{CLA} & \text{INS} & \text{P1} & \text{P2} & \text{LE} & \text{D}_1 & \ldots & \text{D}_\text{LE} \\
   \end{array}
   \]

2. smartcard sends a response APDU back to terminal

3. back to 1.

Two variants, T=0 and T=1: byte and block transmission
ISO 7816 and APDU’s for applets

The JCRE mediates between applets and terminal:

JCRE passes an APDU-object to the selected applet, by invoking its process(apdu) method.

An APDU-object is essentially a buffer, with methods for reading/writing/etc. in it.
The APDU class

So an applet receives an APDU, on which it can invoke

```java
public static byte[] getBytes()
public static short getInBlockSize()
public static short getOutBlockSize()
```

and

```java
public short setIncomingAndReceive()
public short receiveBytes(short bOff)
public short setOutgoing()
  public void setOutgoingLength(short len)
  public void sendBytes(short bOff, short len)
```

in a certain order!
Informal (javadoc) spec

receiveBytes

public short receiveBytes(short bOff) throws APDUException

    Gets as many data bytes as will fit without APDU buffer overflow, at the
    specified offset bOff. Gets all the remaining bytes if they fit.

Parameters: bOff - the offset into APDU buffer.
Returns: number of bytes read. Returns 0 if no bytes are available.
Throws: APDUException - with the following reason codes:

- APDUException.ILLEGAL_USE if setIncomingAndReceive() not called or if
  setOutgoing() or setOutgoingNoChaining() previously invoked.
- APDUException BUFFER_BOUNDS if not enough buffer space for incoming block size.
- APDUException.T1_IFD_ABORT if T=1 protocol is in use and the CAD sends in an
  ABORT S-Block command to abort the data transfer.
Reference implementation

The reference implementation of APDU uses 7 flags

incomingFlag, outgoingFlag, outgoingLenSetFlag, lrIs256Flag, sendInProgressFlag, noChainingFlag, noGetResponseFlag

to enforce invocation order, eg.

public short receiveBytes(short bOff) throws APDUException
{
    if (!getIncomingFlag() || getOutgoingFlag() )
        APDUException.throwIt( APDUException.ILLEGAL_USE );
...

but protocol has $2^7$ states!
Using a model field and FSM to specify APDU

```java
//@ public model int _APDU_state;

//@ requires _APDU_state == 1 && ... ;
//@ ensures _APDU_state == 2 && ... ;
//@
public short setIncomingAndReceive()

//@ requires _APDU_state == 2 && ... ;
//@ ensures _APDU_state == 2 && ... ;
//@
public short receiveBytes(short bOff)
```
Relating reference implementation to formal spec

**Invariants** relating the abstract state to its concrete representation, eg:

```c
/*@ invariant
@   _APDU_state == 2
@   <=>
@   getIncomingFlag() && !getOutgoingFlag();
@*/
```
More detailed JML spec of `receiveBytes(short bOff)`

```jml
/*@ requires _APDU_state == 2 && 
@ 0 <= bOff && 
@ bOff + getInBlockSize() <= BUFFERSIZE; 
@
@ assignable _APDU_state, _Lc, buffer[bOff..bOff+\result-1]; 
@
@ ensures _APDU_state == 2 && 
@ 0 <= \result && \result <= old(_Lc) && 
@ _Lc == old(_Lc) - \result && 
@ bOff + \result <= BUFFERSIZE && 
@ (* data received in buffer[bOff..bOff+\result-1] *); 
@
@ signals (APDUException e) e.getReason() == APDUException.IO_ERROR 
@ || e.getReason() == APDUException.T1_IFD_ABORT 
@*/
```

Here model field `_Lc` is the length of incoming command.
Relating reference implementation to formal spec

Length of the incoming command in JML spec:

```java
//@ public model int _Lc;
//@ public invariant 0 <= _Lc && _Lc < 256;
```

Representation in the reference implementation:

```java
private byte getLc()
{
    return ramVars[LC];
}
```

NB. byte-int conversion yields getLc() ∈ [−128..127]. So

```java
//@ private invariant _Lc == (getLc()&0xFF)
```

and **not** _Lc == getLc().
Bug in reference impl. of receiveBytes

The reference implementation does NOT meet this spec, but requires a stronger precondition than

\[ b_{\text{off}} + \text{getInBlockSize()} \leq \text{BUFFERSIZE}, \]

namely

\[ b_{\text{off}} + \text{getInBlockSize()} < \text{BUFFERSIZE}. \]

This is probably a bug.
Conclusions

- (Still incomplete) formal specs for APDU protocol:
  400 lines code, 400 lines javadoc, 170 lines public JML spec

- State transition diagram nice way to specify the APDU protocol. Why isn’t it used anywhere in the existing documentation??

- Whenever possible, our specs do not say "exception $E_i$ is thrown, if $P_i$ holds" , but insist on "¬$P_1$ & & . . . & & ¬$P_n$" as precondition.