Transactions JavaCard avec Krakatoa

Nicolas Rousset
Axalto/LRI

GECCOO

10 mars 2006

Plan

- Spécificités du langage JavaCard
- Modélisation avec Krakatoa
- Un exemple : vérification du code PIN

Le langage JavaCard

- JavaCard : plate-forme pour les cartes à puce
- Le langage JavaCard
  - Sous-ensemble de Java (pas de chaînes de caractères, pas de nombres flottants, tableaux unidimensionnels, pas de thread...)
  - API spécialisée => sur-ensemble de ce sous-ensemble !
    - Mémoire organisée différemment : persistante / volatile
    - Mécanisme de Transactions

La mémoire dans JavaCard

- Mémoire Persistante
  - stockée en EEPROM
  - valeurs mémorisées durant toute la durée de vie de la carte
- Mémoire Volatile (ou 'transient')
  - stockée en RAM
  - effacée après chaque session

Les Transactions JavaCard

- Problème : arrachage de la carte ou panne de courant
- Idée : faire d'une suite d'écritures en mémoire persistante une opération atomique.
- Solution : un mécanisme de transactions via l'API
  - 3 méthodes (begin, commit, abort)
  - Pas de transactions imbriquées
  - En cas d'Interruption :
    - mémoire persistante revient à son état avant transaction
    - mémoire volatile non affectée (abort) ou effacée (arrachage)

Exemple : les différents cas du abort (1)

```
byte bi, bl;
byte bt[];  //@ invariant JCSystem.isTransient(bt) != JCSystem.NOT_A_TRANSIENT_OBJECT;
byte bp[];  //@ invariant JCSystem.isTransient(bp) == JCSystem.NOT_A_TRANSIENT_OBJECT;

/*@  requires JCSystem.getTransactionDepth() == 0;
  @   ensures bi == \old(bi) && bl == 1 && bt[0] == 1 && bp[0] == \old(bp[0]); @*/
public void test() {
   byte local = 0;
   JCSystem.beginTransaction();
   bi = 1;
   local = 1;
   bt[0] = 1;
   bp[0] = 1;
   JCSystem.abortTransaction();
   bl = local;
}
```

variable d'instance : persistante
=> restaurée par abort()

Exemple : les différents cas du abort (2)

```
byte bi, bl;
byte bt[];  //@ invariant JCSystem.isTransient(bt) != JCSystem.NOT_A_TRANSIENT_OBJECT;
byte bp[];  //@ invariant JCSystem.isTransient(bp) == JCSystem.NOT_A_TRANSIENT_OBJECT;

/*@  requires JCSystem.getTransactionDepth() == 0;
  @   ensures bi == \old(bi) && bl == 1 && bt[0] == 1 && bp[0] == \old(bp[0]); @*/
public void test() {
  byte local = 0;
  JCSystem.beginTransaction();
  bi = 1;
  local = 1;
  bt[0] = 1;
  bp[0] = 1;
  JCSystem.abortTransaction();
  bl = local;
}
```

variable locale : volatile
=> inchangée par abort()

Exemple : les différents cas du abort (3)

```
byte bi, bl;
byte bt[];  //@ invariant JCSystem.isTransient(bt) != JCSystem.NOT_A_TRANSIENT_OBJECT;
byte bp[];  //@ invariant JCSystem.isTransient(bp) == JCSystem.NOT_A_TRANSIENT_OBJECT;

/*@  requires JCSystem.getTransactionDepth() == 0;
  @   ensures bi == \old(bi) && bl == 1 && bt[0] == 1 && bp[0] == \old(bp[0]); @*/
public void test() {
   byte local = 0;
   JCSystem.beginTransaction();
   bi = 1;
   local = 1;
   bt[0] = 1;
   bp[0] = 1;
   JCSystem.abortTransaction();
   bl = local;
}
```

tableau transient : contenu volatile
=> contenu inchangé par abort()

Exemple : les différents cas du abort (4)

```
byte bi, bl;
byte bt[];  //@ invariant JCSystem.isTransient(bt) != JCSystem.NOT_A_TRANSIENT_OBJECT;
byte bp[];  //@ invariant JCSystem.isTransient(bp) == JCSystem.NOT_A_TRANSIENT_OBJECT;

/*@  requires JCSystem.getTransactionDepth() == 0;
  @   ensures bi == \old(bi) && bl == 1 && bt[0] == 1 && bp[0] == \old(bp[0]); @*/
public void test() {
   byte local = 0;
   JCSystem.beginTransaction();
   bi = 1;
   local = 1;
   bt[0] = 1;
   bp[0] = 1;
   JCSystem.abortTransaction();
   bl = local;
}
```

tableau non transient : contenu persistant
=> contenu restauré par abort()

Allocation de la mémoire

```
byte bt[];  //@ invariant JCSystem.isTransient(bt) != JCSystem.NOT_A_TRANSIENT_OBJECT;
byte bp[];  //@ invariant JCSystem.isTransient(bp) == JCSystem.NOT_A_TRANSIENT_OBJECT;

public TestTransaction() {
        bt = JCSystem.makeTransientByteArray((short) 8);
        bp = new byte[8];
    }
```

- Allocation en mémoire persistante
  - l'opérateur **new**
- Allocation en mémoire volatile
  - méthodes **makeTransient...Array** de l'API

Implantation des transactions

- 2 méthodes possibles
  - Table de restauration
    - commit immédiat
  - Table de commit
    - abort immédiat
- Hypothèse : peu de transactions sont interrompues par le programme
  => la première méthode est souvent choisie

Modélisation pour la preuve de programmes

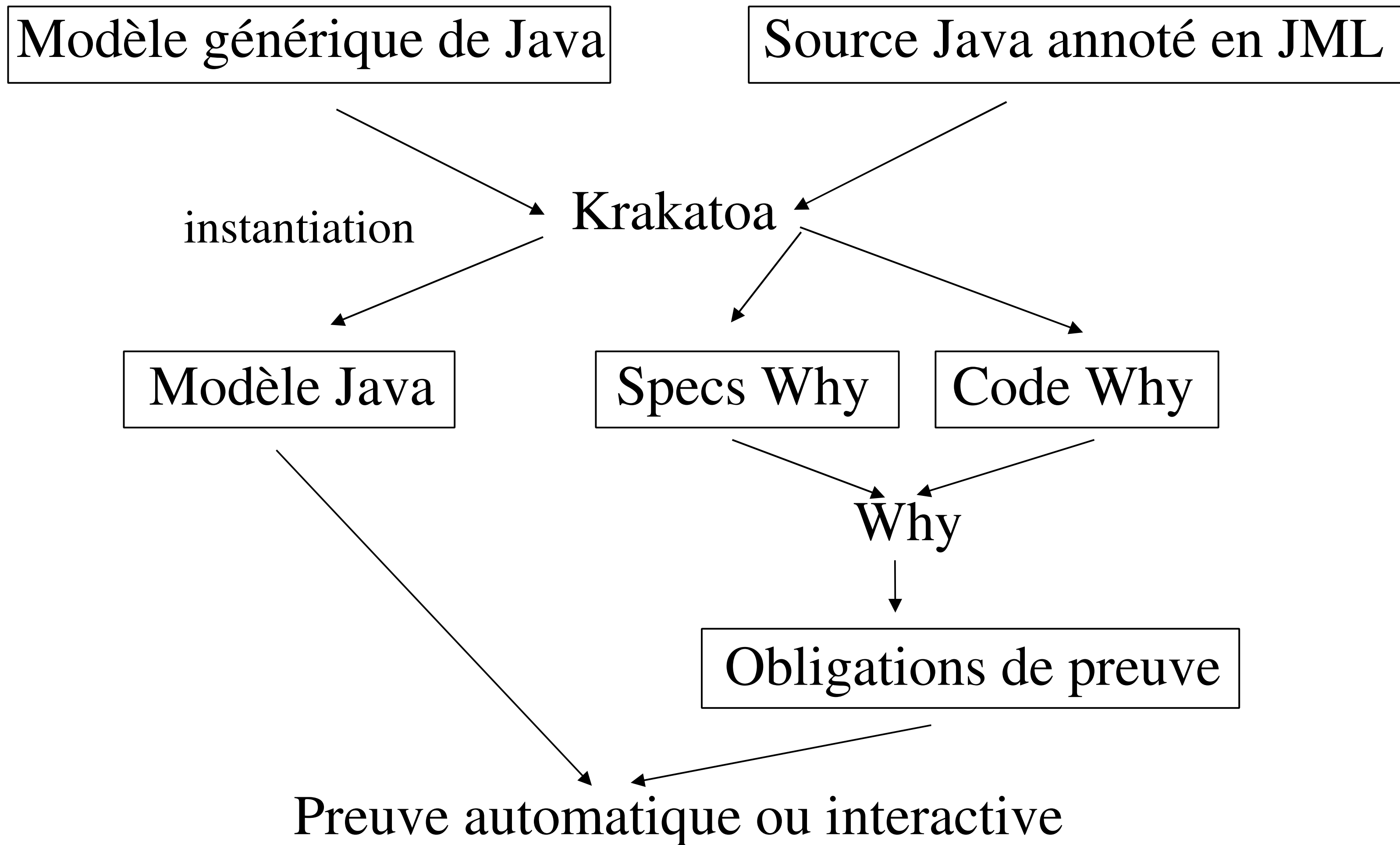
- JML est insuffisant
  - ok pour la non-imbrication des transactions

    M. Pavlova, G. Barthe, L. Burdy, M. Huisman, and J.-L. Lanet, Enforcing high-level security properties for applets, CARDIS'04.
  - mais abortTransaction() non traité
- KeY
  - Interprétation dans la Logique Dynamique

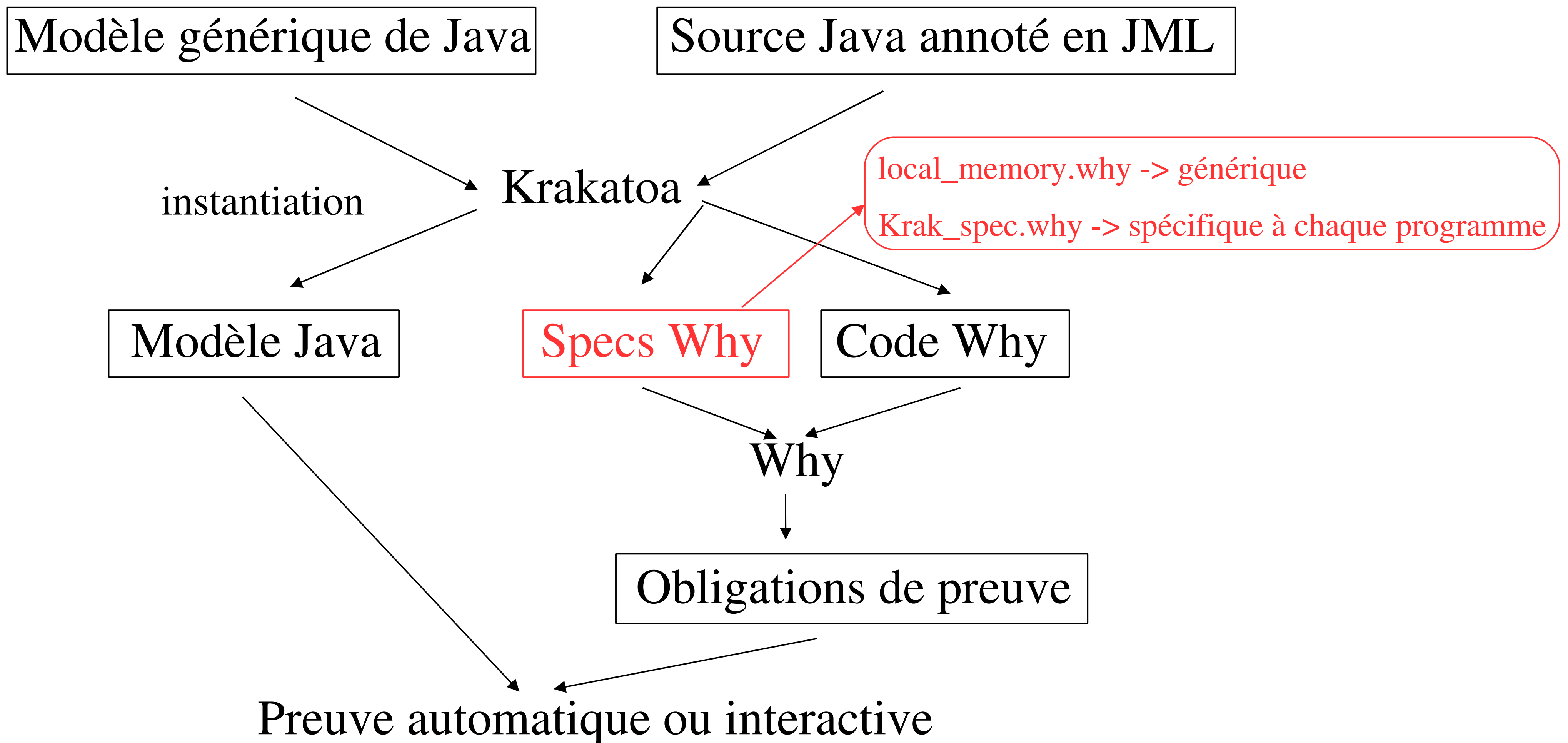
    Reiner Hähnle and Wojciech Mostowski, Verification of Safety Properties in the Presence of Transactions, CASSIS'04.
- LOOP
  - Transformation du source en implémentant les transactions en Java pur

    Engelbert Hubbers and Erik Poll, Reasoning about Card Tears and Transactions in Java Card, FASE'04.

Architecture de Krakatoa



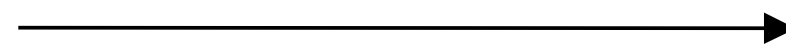
Architecture de Krakatoa



Modélisation avec Krakatoa (1)

```
/*@ requires JCSystem.getTransactionDepth() == 0;
  @ ensures bi == \old(bi);
  @ ensures bl == 1;
  @ ensures bt[0] == 1;
  @ ensures bp[0] == \old(bp[0]); @*/
public void test() {
   byte local = 0;
   JCSystem.beginTransaction();
   bi = 1;
   local = 1;
   bt[0] = 1;
   bp[0] = 1;
   JCSystem.abortTransaction();
   bl = local;
}
```

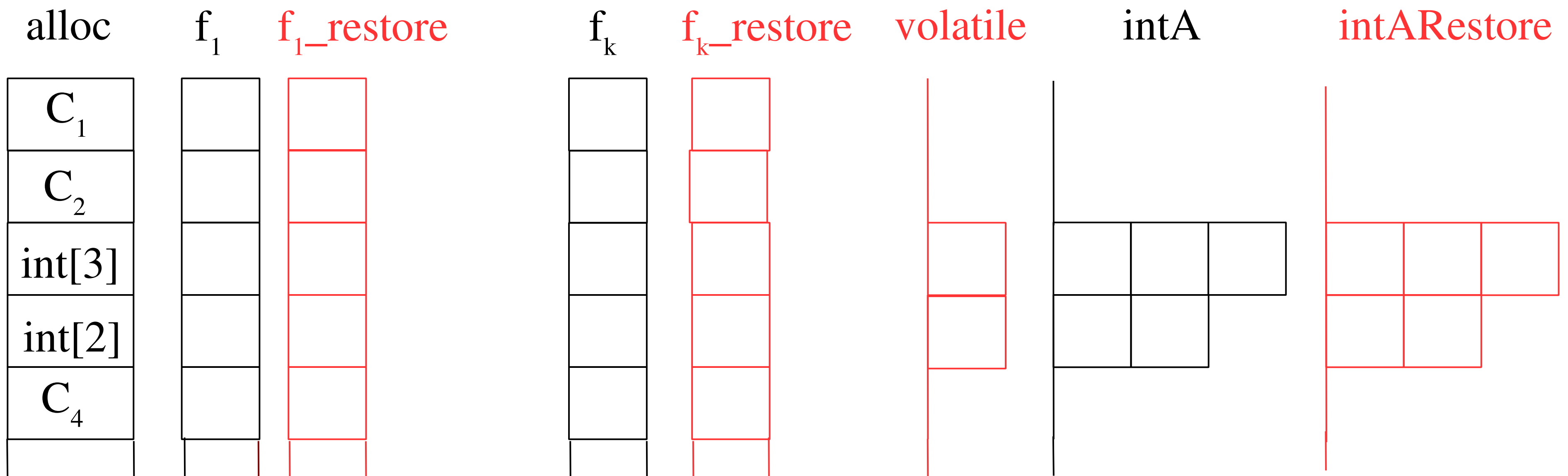
Traduction en Why



```
let TestTransaction_test_body = fun (this : value) ->
 { transactionDepth = 0 }
 let local = ref 0 in
   (JCSystem_beginTransaction_parameter void);
   bi := update !bi this 1;
   local := 1;
   intA := array_update !intA (acc !bt this) 0 1
   intA := array_update !intA (acc !bp this) 0 1
   (JCSystem_abortTransaction_parameter void);
   bl := update !bl this !local
 { acc(bi, this) = acc(bi@, this)
   and acc(bl, this) = 1
   and array_acc (intA, acc (bt, this), 0) = 1
   and array_acc(intA, acc(bp, this), 0) =
          array_acc(intA@, acc(bp@, this), 0) }
```

Modélisation avec Krakatoa (2)

- Modification du modèle mémoire
  - Séparation persistante/volatile
  - Variables pour stocker les valeurs à restaurer



Modélisation avec Krakatoa (3)

- Interprétation des méthodes en Why
  - spécification générique
    - makeTransient...Array()

makeTransientByteArray()

```
external parameter makeTransientByteArray : n : int ->
  { 0 <= n }
  value
  writes alloc, volatile
  {
    result <> Null
    and fresh(alloc@, result)
    and typeof(alloc, result, ArrIntType)
    and arraylength(alloc, result) = n
    and store_extends(alloc@, alloc)
    and volatile = update(volatile@, result, true)
  }
```

Modélisation avec Krakatoa (4)

- Interprétation des méthodes en Why
  - spécification générique
    - makeTransient...Array()
  - spécification relative à chaque programme
    - beginTransaction()
    - abortTransaction()

beginTransaction()

```
parameter JCSystem_beginTransaction_parameter : tt:unit ->
  { transactionDepth = 0 }
  unit
  reads bi,bl,bp,bt,intA
  writes transactionDepth,bi_restore,bl_restore,bp_restore,bt_restore,intARestore
  {
    transactionDepth = 1
    and  bi_restore = bi
    and  bl_restore = bl
    and  bt_restore = bt
    and  bp_restore = bp
    and  forall v:value. forall i:int. array_acc(intARestore, v, i) = array_acc(intA, v, i)
  }
```

abortTransaction()

```
parameter JCSystem_abortTransaction_parameter : tt:unit ->
  { transactionDepth = 1 }
  unit
  reads alloc,bi_restore,bl_restore,bp_restore,bt_restore,intARestore,volatile
  writes transactionDepth,bi,bl,bp,bt,intA
  { transactionDepth = 0
    and bi = bi_restore
    and bl = bl_restore
    and bt = bt_restore
    and bp = bp_restore
    and forall v:value. forall i:int.
          if acc(volatile, v)
          then array_acc(intA, v, i) = array_acc(intA@, v, i)
          else array_acc(intA, v, i) = array_acc(intARestore, v, i) }
```

Transactions dans Krakatoa

- Cette traduction spécifique pour supporter les transactions est utilisée si l'option **-javacard** est donnée à Krakatoa en ligne de commande.
- Preuves : sur l'exemple précédent, tout est prouvé par Simplify

Arrachage de la carte

- Exemple : vérification d'un code PIN
  - Code PIN à 4 chiffres
  - Laisser 3 essais au maximum
  - Trou de sécurité potentiel : retirer la carte entre chaque essai

Modélisation de l'arrachage de la carte (1)

- Spécification : ajout d'un mot clé cardtear à JML

Exemple : vérifier un code PIN

```
byte[] pin, triesLeft;
//@ invariant  ... && JCSystem.isTransient(pin) == JCSystem.NOT_A_TRANSIENT_OBJECT;
//@ invariant  ... && JCSystem.isTransient(triesLeft) == JCSystem.NOT_A_TRANSIENT_OBJECT;

//@ ghost boolean _checkFailed;

/*@ requires guess != null && guess.length >= 4 && !_checkFailed;
  @ cardtear _checkFailed ==> triesLeft[0] == \old(triesLeft[0]) - 1;
  @*/
boolean check(byte[] guess) {
}
```

Implémentation naïve

```
public boolean badCheck(byte[] guess) {
        boolean res = false;
        if (triesLeft[0] == 0) {
            res = false;
        } else {
            triesLeft[0] = (byte) (triesLeft[0] – 1);
            if (Util.arrayCompare(guess, 0, pin, 0, 4) == 0) {
                triesLeft[0] = 3;
                res = true;
            } else { //@ set _checkFailed = true;
                res = false;}
        }
        return res;
 }
```

Implémentation naïve : vulnérable à l'arrachage

```
public boolean badCheck(byte[] guess) {
    boolean res = false;
    if (triesLeft[0] == 0) {
        res = false;
    } else {
        triesLeft[0] = (byte) (triesLeft[0] – 1);
        if (Util.arrayCompare(guess, 0, pin, 0, 4) == 0) {
            triesLeft[0] = 3;
            res = true;
        } else { //@ set _checkFailed = true;
            res = false;}
    }
    return res;
}
```

```
JCSystem.beginTransaction();
...
boolean b = badCheck();
...
JCSystem.commitTransaction();
```

Si arrachage ici, durant une transaction, alors triesLeft[0] n'est pas décrémenté.
=> possibilité d'un nombre illimité d'essais

Modélisation de l'arrachage de la carte (2)

- Spécification : ajout d'un mot clé cardtear à JML
- Simulation : Insertion d'un appel de méthode possibleCardTear() après chaque affectation.

Traduction de badCheck() en Why

```
let TestPIN_badCheck_body = fun (this : value) (guess : value) ->
  { ... }
   let res = ref false in
       if (...)
       then res := false; (possibleCardTear void);
       else intA = array_update (acc !triesLeft this) 0 (sub_int (array_acc !intA (acc !triesLeft this) 0) 1));
                (possibleCardTear void);
               if Util_arrayCompare_parameter guess 0 (acc !pin this) 0 4 = 0
               then intA = array_update (acc !triesLeft this) 0 3; (possibleCardTear void);
                    res := true; (possibleCardTear void);
               else _checkFailed = update _checkFailed this true; res := false; (possibleCardTear void);
     raise (Return_bool !res)
 { ... }
```

Modélisation de l'arrachage de la carte (3)

- Spécification : ajout d'un mot clé cardtear à JML
- Simulation : Insertion d'un appel de méthode possibleCardTear() après chaque affectation.
- Comportement : Utilisation d'une exception Why pour décrire l'arrachage.
  - possibleCardTear() engendrée pour chaque programme

possibleCardTear()

```
parameter possibleCardTear : tt:unit ->
 { }
 unit
 reads alloc,intARestore,volatile, transactionDepth
 writes pin,triesLeft raises CardTear
 { intA=intA@ and pin=pin@ and triesLeft=triesLeft@
   | CardTear => if transactionDepth <> 0
                 then pin = pin_restore and triesLeft = triesLeft_restore
                      and forall v:value. forall i:int. if acc(volatile, v)
                                                        then array_acc(intA, v, i) = 0
                                                        else array_acc(intA, v, i) = array_acc(intARestore, v, i)
                 else pin = pin@ and triesLeft = triesLeft@
                     and forall v:value. forall i:int. if acc(volatile, v)
                                                        then array_acc(intA, v, i) = 0
                                                        else array_acc(intA, v, i) = array_acc(intA@, v, i) }
```

Implémentation résistant à l'arrachage

```
byte[] temps;
public boolean goodCheck(byte[] guess) {
        boolean res = false;
        if (triesLeft[0] == 0) {
             res = false;
        } else {
            temps[0] = (byte) (triesLeft[0] – 1);
            Util.arrayCopyNonAtomic(temps, 0, triesLeft, 0, 1);
             if (Util.arrayCompare(guess, 0, pin, 0, 4) == 0) {
                triesLeft[0] = 3;
                res = true;
             } else { //@ set _checkFailed = true;
                res = false;}
        }
        return res;
}
```

Implémentation résistant à l'arrachage

```
byte[] temps;
public boolean goodCheck(byte[] guess) {
        boolean res = false;
        if (triesLeft[0] == 0) {
              res = false;
        } else {
             temps[0] = (byte) (triesLeft[0] – 1);
             Util.arrayCopyNonAtomic(temps, 0, triesLeft, 0, 1);
              if (Util.arrayCompare(guess, 0, pin, 0, 4) == 0) {
                  triesLeft[0] = 3;
                  res = true;
              } else { //@ set _checkFailed = true;
                  res = false;}
        }
        return res;
}
```

```
JCSystem.beginTransaction();
...
boolean b = goodCheck();
...
JCSystem.commitTransaction();
```

Effectue la copie sans tenir compte d'une éventuelle transaction
=> triesLeft[0] est bien décrémenté

arrayCopyNonAtomic()

```
external parameter arrayCopyNonAtomic : src : value -> srcOff : int -> dest : value -> destOff : int -> length : int ->
 {  src <> Null and srcOff >= 0 and srcOff + length <= arraylength(alloc, src) and
   dest <> Null and destOff >= 0 and destOff + length <= arraylength(alloc, dest) and length >= 0 }
 int
 reads alloc, transactionDepth
 writes intA, intARestore
 { (forall i : int. 0 <= i and i <= length -> array_acc(intA, dest, destOff + i) = array_acc(intA@, src, srcOff + i)) and
    array_modifiable(alloc@, intA@, intA, array_sub_loc(dest, destOff, destOff + length - 1)) and
    if transactionDepth <> 0
    then (forall i : int. 0 <= i and i <= length -> array_acc(intARestore, dest, destOff + i) = array_acc(intA, dest, destOff + i)) and
            array_modifiable(alloc@, intARestore@, intARestore, array_sub_loc(dest, destOff, destOff + length - 1))
   else intARestore = intARestore@
 }
```

Arrachage dans Krakatoa

- Cette modélisation pour supporter l'arrachage est utilisée si l'option **-cardtear** est donnée à Krakatoa en ligne de commande.
- Exemple : vérification du code PIN
  - badCheck() : Simplify trouve un contre-exemple dans le cas du trou de sécurité
  - goodCheck() : prouvé par Simplify

Conclusions

- Modélisation indépendante du prouveur utilisé en sortie de Krakatoa.
- Tous les exemples ont été prouvés automatiquement.
- Expérimentation en cours sur une applet industrielle chez Axalto
  - Exprimer des propriétés sensibles à vérifier en cas d'arrachage