An overview of JML tools and applications

Lilian Burdy Gemplus

Yoonsik Cheon, Gary Leavens Iowa Univ.

David Cok Kodak

Michael Ernst MIT

Rustan Leino Microsoft

Joe Kiniry, Erik Poll Nijmegen Univ.

Overview

1. The JML language

3. Tools for JML

5. Applications

7. Conclusions

1. The JML language

Java Modeling Language

- Initiative of Gary Leavens [Iowa State Univ.]
- Behavioural Interface Specification Language for Java: annotations added to Java programs, expressing pre-, postconditions, invariants...
- Inspired by Eiffel (Design-by-Contract) & Larch
- Main design goal: easy to learn
  - simple extension of Java's syntax

JML example

```
private int balance;
final static int MAX_BALANCE;

/*@ invariant 0 <= balance &&
                  balance < MAX_BALANCE;
  @*/
```

JML example

```
/*@ requires    amount >= 0;
     assignable balance;
     ensures     balance == \old(balance) - amount;
     signals     (PurseException)
                   balance == \old(balance);
  @*/
public void debit(int amount) {
....
}
```

JML example

```
private byte[] pin;
private byte appletState;

/*@ invariant
        appletState == PERSONALIZED
      ==>
        pin != null &&
        pin.length == 4 &&
        (\forall int i; 0 <= i && i < 4
                       ; 0 <= pin[i] && pin[i] <= 9);
   @*/
```

2. Tools for JML

Tools for JML

- tools for reading & writing specs
- tools for generating specs
- tools for checking implementation against specs

Tools for reading & writing specs

- parsing & typechecking (as part of other tools)
- jmldoc: javadoc for JML

Tools for generating specs

- Invariant detection using Daikon
  [Michael Ernst, MIT]

  Daikon observes execution of code to detect likely invariants

Tools for checking specs (I)

- Runtime assertion checker
  [Gary Leavens et al., Iowa State Univ.]
  tests if specs are violated at runtime
  - not so exciting for academia, but appealing to industry
  - well-specified code is easy to test !
    - runtime checker handles \forall and \old
  - jmlunit: tool combining runtime checking with unit testing

Tools for checking specs (II)

- Extended static checker ESC/Java
  [Rustan Leino et al., ex-Compaq]

  automatic verification of simple properties

  - not sound, not complete, but finds lots of bugs quickly
  - eg. can "prove" absence of NullPointer- and ArrayIndexOutOfBoundsExceptions

- Chase tool [Nestor Cataño, INRIA] remedies one important source of unsoundness

Tools for checking specs (III)

"Real" program verification

- JACK tool [Gemplus]
  <u>automatic</u> verification of JML-annotated code
  Inspired by ESC/Java, integrated with Eclipse

- LOOP tool [Nijmegen]
  <u>interactive</u> verification of JML-annotated code

- Krakatoa tool [INRIA/Orsay] for interactive verification now also supports JML

Tools for checking specs

There is a range of tools
offering different levels of assurance
at different costs (ie. time & effort):

- runtime assertion checking
- extended static checking using ESC/Java
- automatic verification using JACK
- interactive verification using LOOP, Krakatoa

3. Applications

JavaCard

- Subset of a superset of Java for programming smart cards
  - – no floats, no threads, limited API, optional gc, ...
  - + support for allocation in EEPROM or RAM
- Ideal target for formal methods
  - small programs, written in simple language, using small API, whose correctness is critical
  - highest levels of security evaluation standards require use of formal methods (Common Criteria)

Applications of JML to JavaCard

as part of ist ect

- Writing JML specs of JavaCard API [Cardis'00]
- Checking applets using ESC/Java [FME'02]
  - 1000's of lines of code
- Verifying applets using LOOP [AMAST'02]
  - 100's of lines of code
- Runtime checking part of smartcard OS [Cardis'02]

4. Conclusions

Assertion-based languages promising way to use formal methods in industry

- Familiar syntax and semantics
- No need for formal model (code is formal model)
- Easy to introduce use incrementally

NB: JML does not provide or impose any design methodolody

What to specify ?

- Detailed functional specs often too difficult

- Just establishing weak specs, eg.

  ```
  requires   ....
  ensures    true;
  signals    (NullPointerException)  false;
  ```

  often suffices to expose most invariants

- Invariants make explicit many design decisions that are typically undocumented

Using JML for JavaCard applets

- For smartcard applets, verifying simple "safety" properties (eg. absence of certain exceptions)
  with JACK or ESC/Java has good return-on-investment
- Verification has found errors not found during testing
- Using JML tools to help manual code reviews when certifying code ?

JML

- Lots of ongoing work and open issues about JML, eg.
  - tricky questions about semantics
  - concurrency ?
  - alias control & ownership models ?
- Agreeing on common syntax & semantics is hard work! (witnessed by upcoming patch of ESC/Java)
- Most tools just support subsets of JML
- JML as standard or as vehicle for research ?

JML

- Having a common specification language supported by different tools important benefit
  - for individual tool builders, and
  - for users

- JML is an open collaborative effort, and we welcome cooperation with others

More info:

www.jmlspecs.org