Event-Based Runtime Verification of Java Programs

Workshop On Dynamic Analysis 2005

Marcelo d'Amorim, University of Illinois
Klaus Havelund, Kestrel Technology

Runtime Verification (RV)

- Lightweight method of verification that introduces monitors in the program to observe its dynamic behavior specified in some formalism. Ex. LTL, ptLTL, MTL, ERE, etc.
- RV embodies many possibly orthogonal aspects: online/post-morten, sync./async., state-based/event-based, etc.
- Scalability ↑, Usefulness ↓, Overhead ↓

HAWK

- Language extension of the finite-trace meta logic Eagle [Barringer *et al.*, 2004] together with its compiler, where:
  - Events appear as atoms in formulae
  - Data values (actual parameters, return values, calling threads) can extend the environment where formulae are evaluated
  - Instrumentation is automated

Motivation & Goals

- Declarative property specification
  - Automate instrumentation of Eagle for Java
    - Event-Based x State-based RV

Related Work

- Java MAC [M. Kim *et al.,* 2001]
- Jass Trace Assertions [D. Bartetzko *et al.* 2001]
- Temporal Rover [D. Drusinsky, 2000]
- MOP [Chen *et al*., 2004]
- AOP [G. Kiczales *et al.*, 1997]

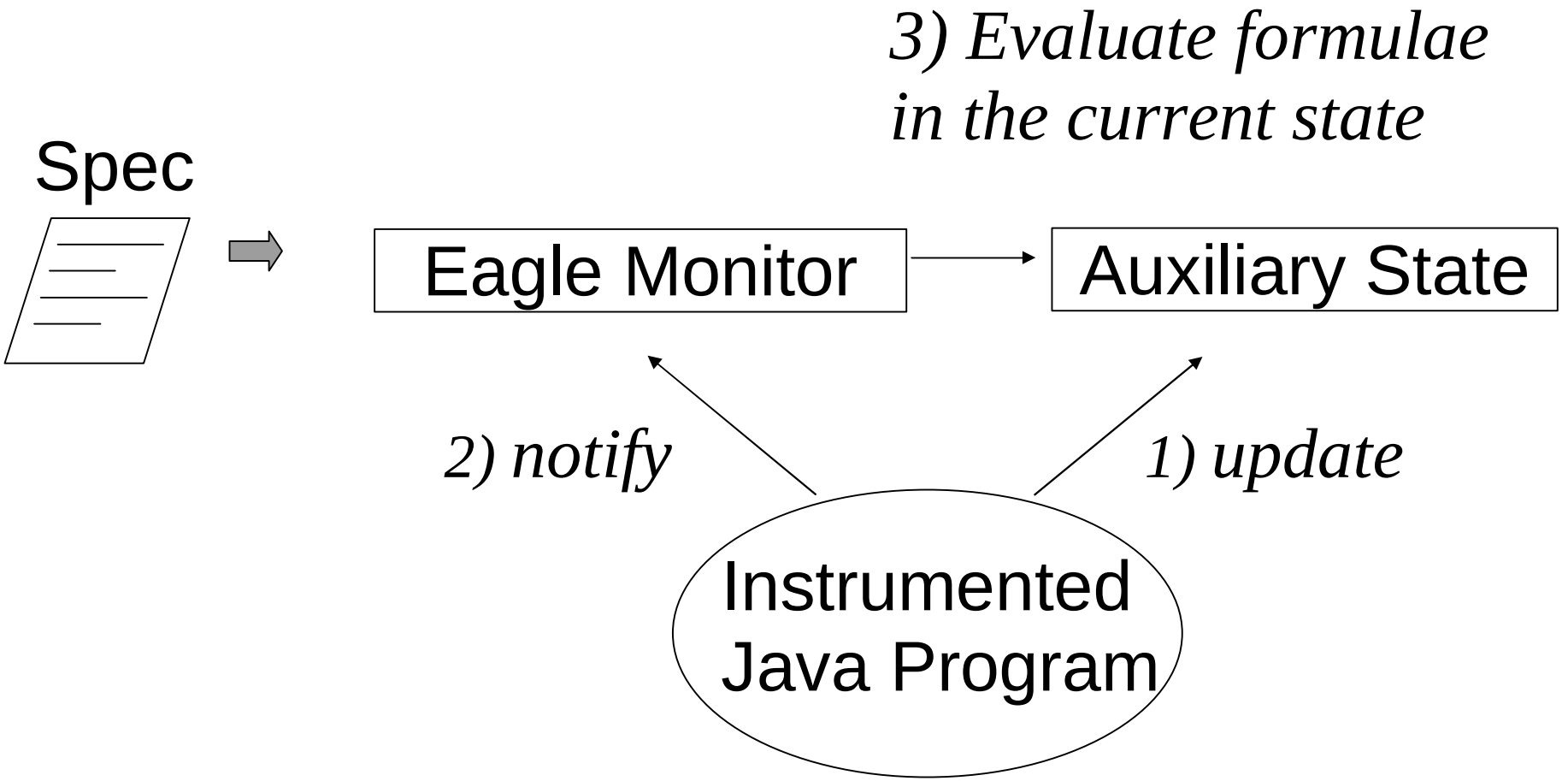
Modal Logics and HAWK

- Also inspired by Modal Logics of Transition Systems (CCS, π-calculus, etc.)

F ::= … | ~F | <Atom>F
| “Eagle Formula extended with F”

[Atom]F == ~<Atom>~F

HAWK: Eagle + Events + Java



HAWK Example 1

```
observer BufferObserver {
  classPath = C:/downloads/src
  targetPath = C:/downloads/src
  terminationMethod = bufferexample.Barrier.end()

  var Buffer b ;
  var Object o ;
  var Object k ;
  mon B = Always (
          [b?.put(o?)]
             Eventually (
                <b.get() returns k?> (o == k))) .
}
```

HAWK Example 2

```
observer FileSystemObserver {...
  var Thread t ;
  var FileSystem fs ;
  var int l ;
  mon F1 =
    Always ([t?:fs?.acquireLock(l?) returns]
    @ ( Until( [*:fs.acquireLock(l) returns]false,
                <t:fs.releaseLock(l)>true))
  ) .
  mon F2 =
    Always ( [t?:fs?.releaseLock(l?)]
    # ( Since( [*:fs.releaseLock(l)]false ,
                <t:fs.acquireLock(l) returns>true))
  ) .
}
```

Summary

- HAWK simplifies, via language integration and instrumentation, the creation of monitors for the Eagle logic, which includes: LTL with past, ERE, MTL, and many others.

Further Work & Question

- Further work
  - Capture other events
  - Add actions?
  - Program visualization
  - Vector clocks
- We used AspectJ as our instrumentation tool.
  - Could HAWK be used to introduce temporal cutpoints in the program?

Thanks!