Environment Modeling for Modular Software Analysis with Java PathFinder Part 1

Oksana Tkachuk
SGT/NASA Ames
oksana.tkachuk@nasa.gov

Peter Mehlitz
SGT/NASA Ames
peter.c.mehlitz@nasa.gov

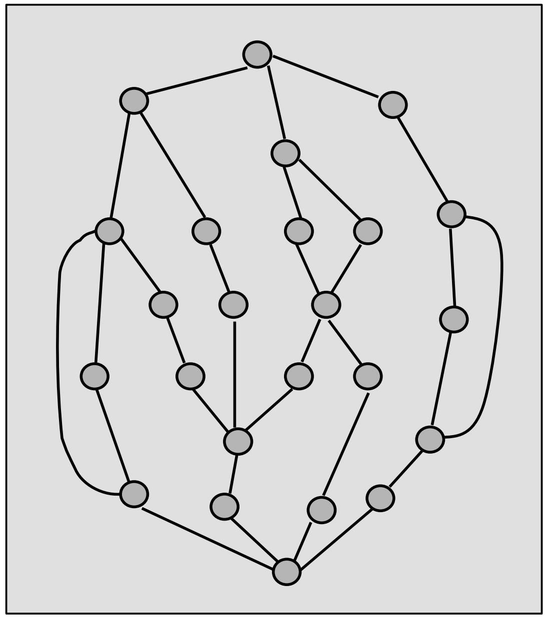
Software Model Checking

program / model

```
void add(Object o) {
 buffer[head] = o;
 head = (head+1)%size;
}

Object take() {
 …
 tail=(tail+1)%size;
 return buffer[tail];
}
```

model checker



YES (property holds)

property

always(ϕ **or** ψ)

NO + counterexample:

```
Line 5: …
Line 12: …
…
Line 41:…
Line 47:…
```

The Good

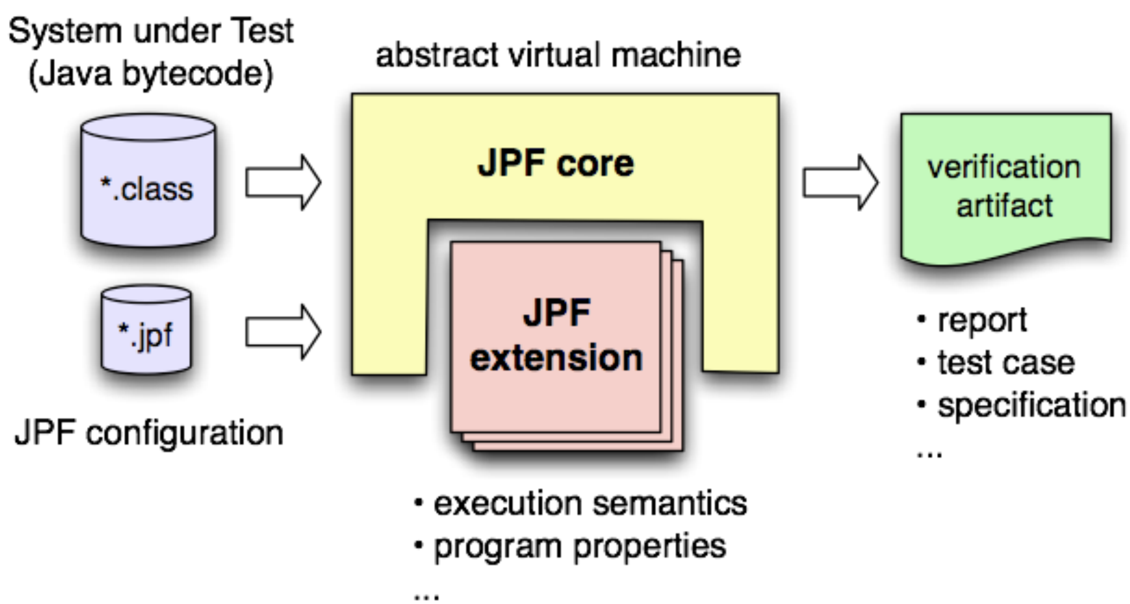
- Exhaustively explores all executions in a systematic way
- Reports error traces

The Bad and the Ugly

- Software is complex
- Not finite state
- State space explosion
- Complex libraries, native code
  - Many frameworks
  - GUI, Web, Android
- Open systems
  - User-driven
  - Event-driven
- Difficult to implement and use
- Extremely difficult to verify

Software Model Checkers

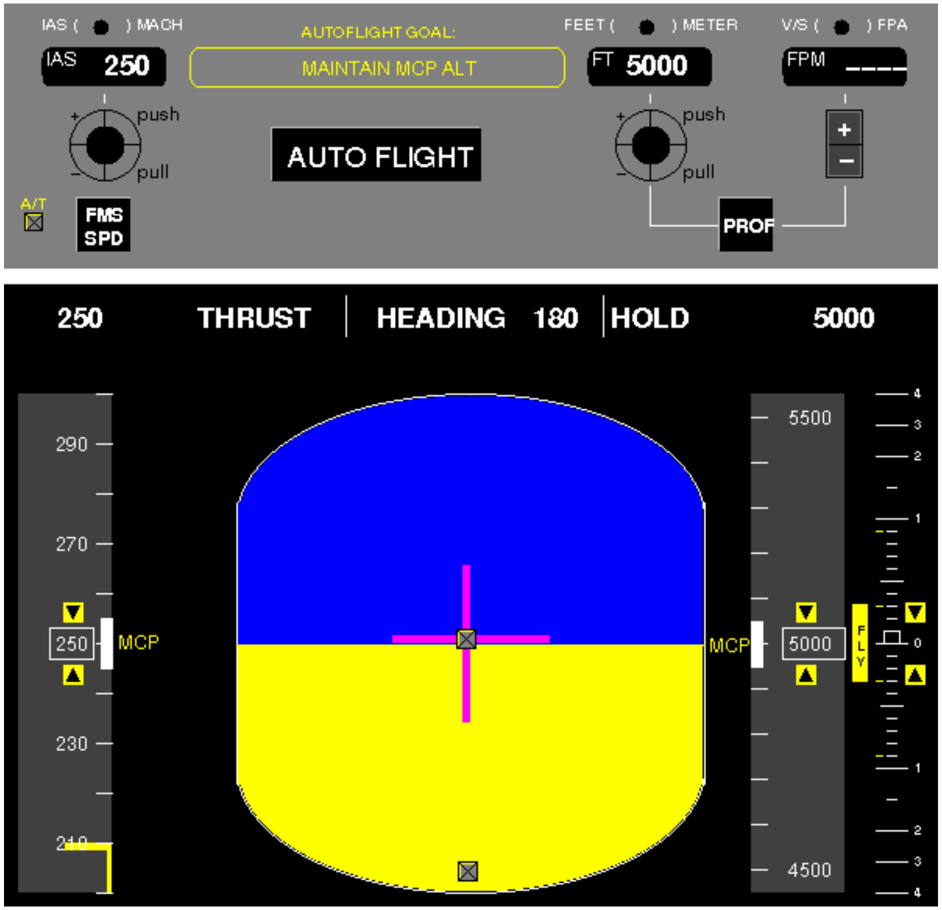
- Spin, SMV, SLAM, …
- In this talk: Java PathFinder (JPF)



- Extensible virtual machine framework for Java bytecode verification
- Workbench to efficiently implement many kinds of verification tools
  - software model checking (deadlocks, races, assert errors)
  - test case generation (symbolic execution) and more

Motivating Example: Autopilot Tutor

Ames Research Center



- ✦ Multiple components
  - User (pilot)
  - Machine (autopilot)
  - Interface (knobs, wheels)
- ✦ Pilot tasks
  - Climb and maintain altitude
  - Capture the altitude
- ✦ Mode Confusions
  - States where the pilot is mistaken about the state of the autopilot
- ✦ Kill the capture
  - Pilot expects to capture the goal altitude but autopilot misses the altitude

Autopilot Code

- Web-based applet
  - Complex Swing/AWT libs
  - GUI is used to display the state of the underlying machine
  - No buttons, just clickable areas
- One Java class
  - >3,500 LOC (dense)
- Open event-driven system
  - Takes user input
- Initial attempts to verify
  - Manual editing, final model erroneous

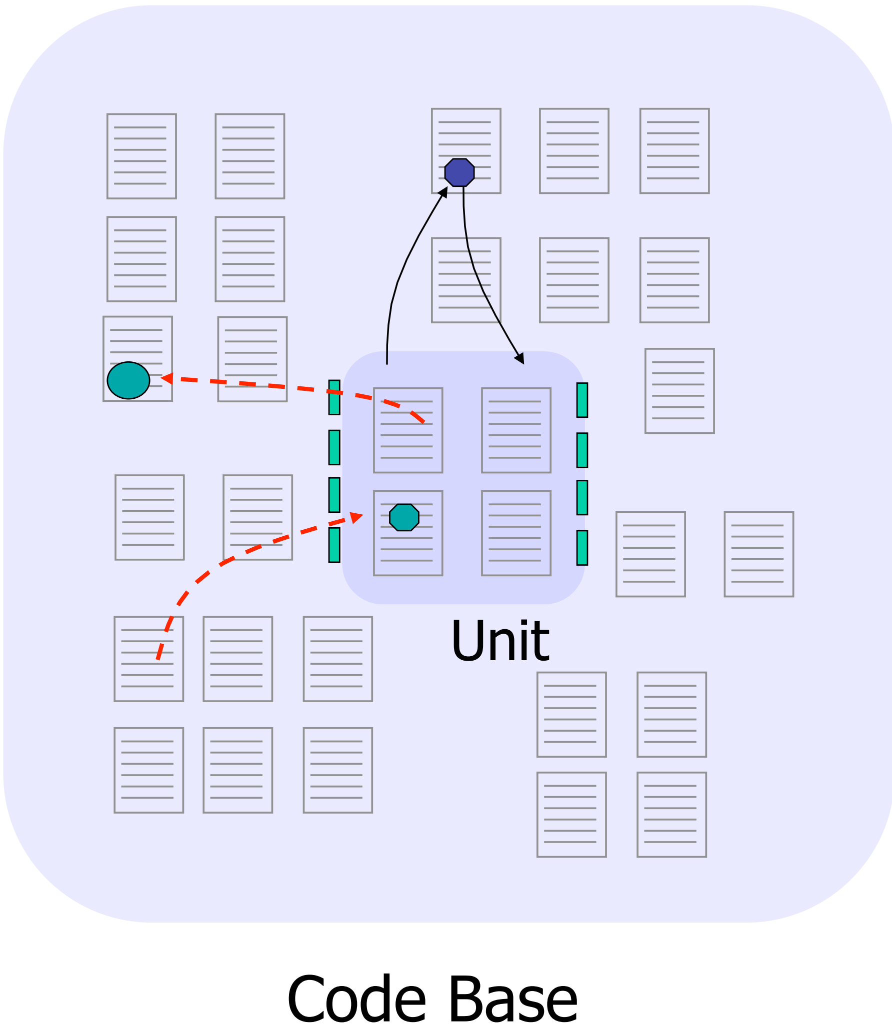
Need Solutions to Handle

- Large systems (scalability)
  - Modular analysis
  - Restrict analysis to selected parts (unit under analysis)
- Open systems/units (enabling)
  - Close with execution context (environment model)
  - Generate code for missing components
    - User model (drivers)
- Complex libraries/frameworks (reduction)
  - Generate simplified library models (stubs)

Environment Generation Problem

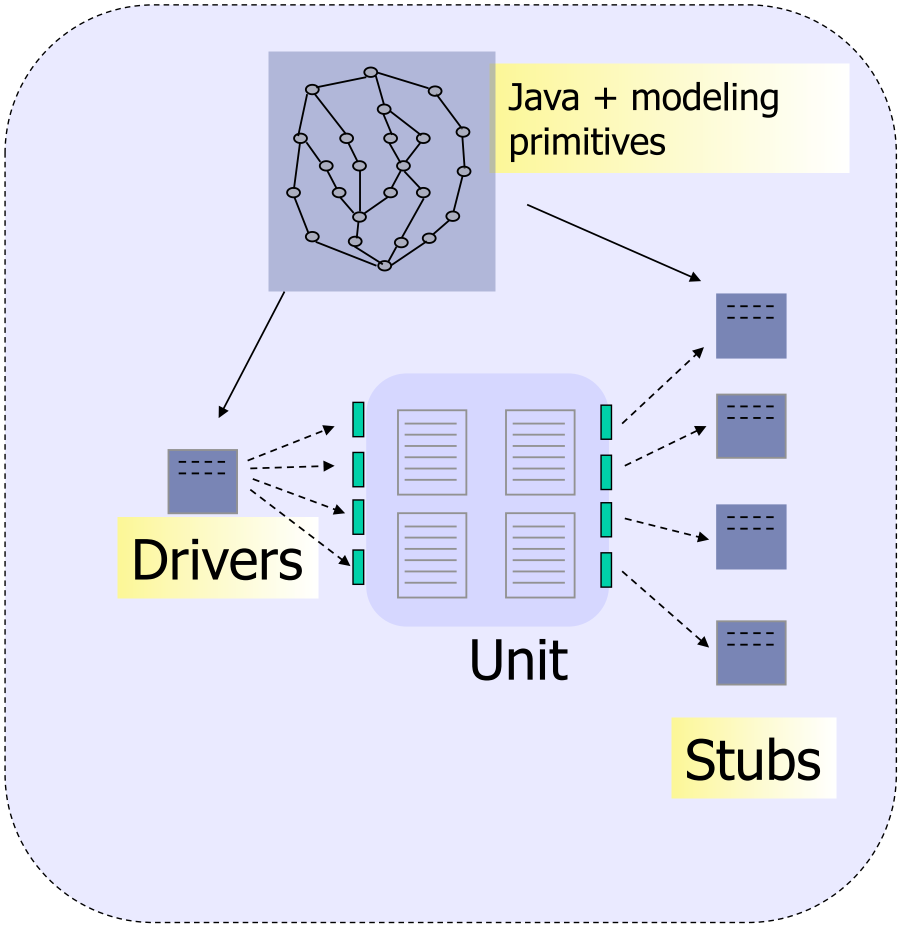
- Persistent across different types of analysis
  - Testing
    - test harness, mock objects
  - Static Analysis
    - stubs for native methods
  - Model Checking
    - main, library stubs
- Environment needs to be
  - Restrictive enough to allow for tractable analysis
  - General enough to uncover errors or produce good coverage for unit

Environment Generation Problem



- In OO (Java) systems, boundaries and interactions between unit and environment are complex
  - Control effects: invoking of methods
  - Data effects: passing data and modifying data
  - Locking, exceptions, global references
  - Hard to identify interaction points

Modular Verification



- Drivers
  - Active classes hold a thread of control
  - Usually make calls to unit
    - GUI, Web, Android user
- Stubs
  - Passive classes
  - Usually called by unit
- Modeling primitives
  - Non-determinism
  - Symbolic values

Closed Unit + Unit Properties → Java Model Checker

Environment Parts

- Structural Info
  - Classes, fields, methods
- Behavior
  - Universal environments
    - Perform all possible sequences of actions, with all possible input values
    - Safe but impractical
  - Environment assumptions
    - can be used to generate more precise environments
- Code
  - Java

Environment Generation Methodology

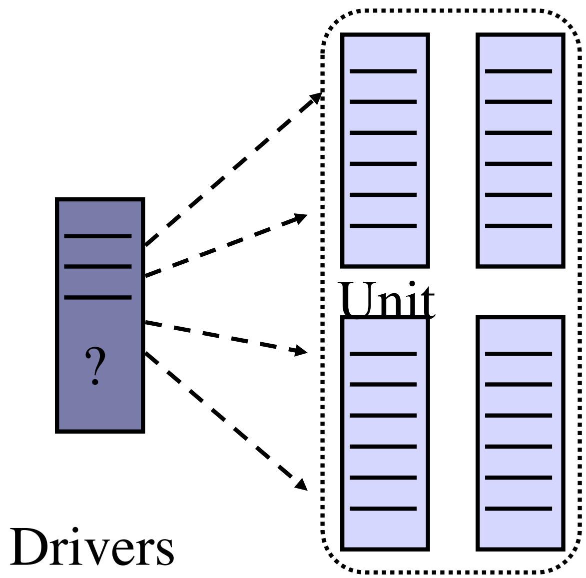
Ames Research Center

- Interface Discovery
  - Unit interface, environment interface
  - program actions
    - Method invocation, field assignment
- Acquiring Assumptions
  - No code to analyze
    - User specifications
  - Analyze environment implementation
    - Static analysis
- Code Generation
  - Modeling primitives
    - non-determinism, over-approximation

Balancing

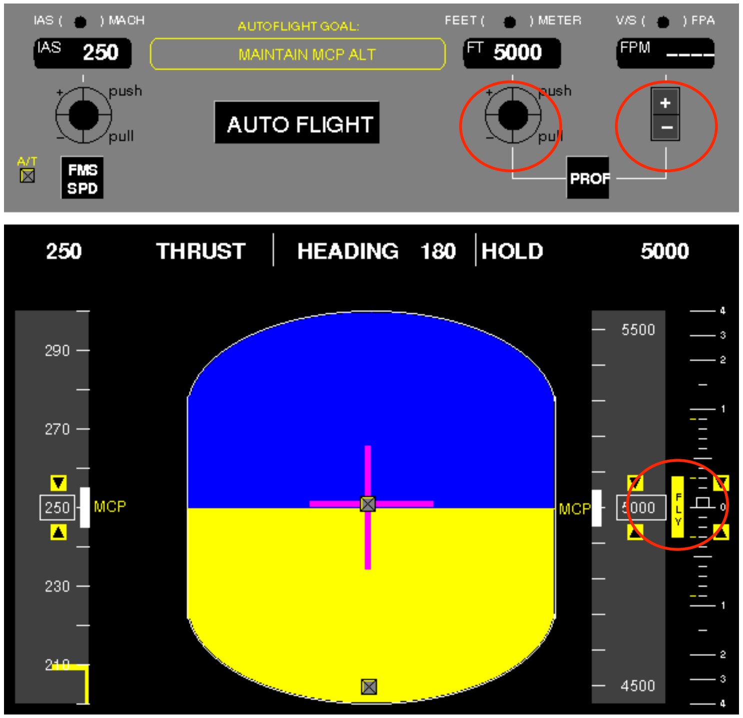
- Human cost
  - Effort to write specifications
- Tool cost
  - The expense of model checking
  - The more general the environment, the more expensive the model checking
- Degree of confidence
  - Coverage over unit code
  - The more restrictive the environment, the more poor the coverage

Unit Interface Discovery



- Scan the unit for possible env actions
- General Java units
  - Public methods and fields
- Event-driven systems
  - Domain-specific event-handling methods that process user inputs
  - NASA's Autopilot
    - mouseClicked(MouseEvent)

Pilot Actions



- incrMCPAlt
- decrMCPAlt
- pullAltKnob
- pushAltKnob
- incrMCPVS
- decrMCPVS
- fly
- init

```
MouseEvent incrMCPAltEvent = new MouseEvent(400, 110);
MouseEvent flyEvent = new MouseEvent (550, 440);
…
incrMCPAlt = mouseClicked (incrMCPAltEvent);
fly = mouseClicked (flyEvent);
```

Pilot Scenarios

- Climb and Maintain MCP Alt
  - incrMCPAlt * ; pullAltKnob; fly *
  - Until level off
- Capture MCP Alt
  - incrMCPAlt * ; pullAltKnob ; fly *
  - Until in capture region
- Climb and maintain MCP - fixed rate of climb
  - incrMCPAlt * ; pullAltKnob ; incMCPVS*; fly *
  - Until in capture region
- Climb away from MCP Alt – 2sec
  - incrMCPAlt * ; pullAltKnob ; fly * (until in capture) incrMCPVS * (small enough to stay in capture); fly *

init; incrMCPAlt *; pullAltKnob ; fly *; incrMCPVS*; fly *

init; incrMCPAlt^{1,10}; pullAltKnob ; fly^{1,10}; incrMCPVS^{1,10}; fly^{1,10}

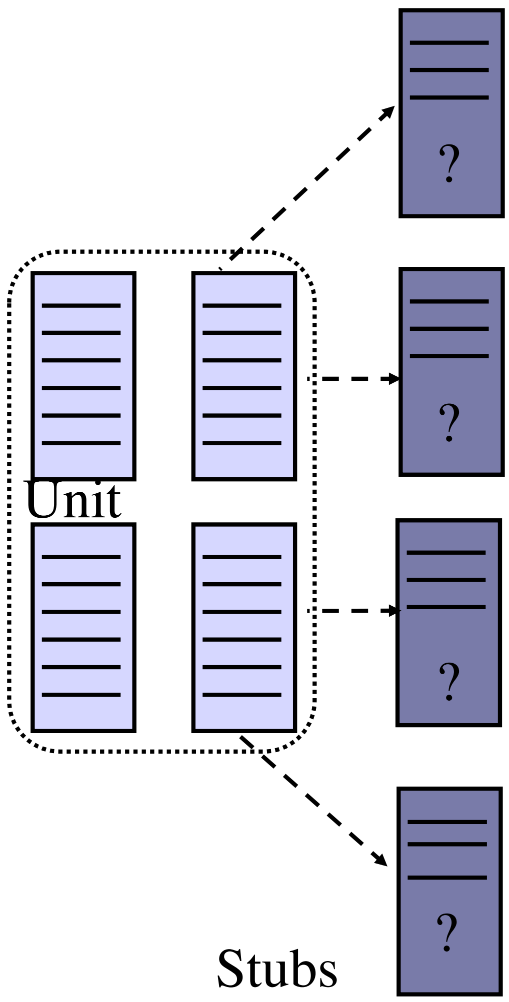
Generated Driver Code

```
…
System.out.println("@EnvDriver: init");
autopilot.mouseClicked(initEvent);

//executes from 1 to 10 times
for(int i=0;i<1+Verify.random(9);++i){
  System.out.println("@EnvDriver: incrMCPAlt");
  autopilot.mouseClicked(incrMCPAltEvent);
}

System.out.println("@EnvDriver: pullAltKnob");
autopilot.mouseClicked(pullAltKnobEvent);
…
```

Environment Interface Discovery



- Scan unit for all external references
  - Classes
  - Methods
  - Fields
- Side-effects analysis
  - Calculate the set of memory locations that may/must be modified by method execution
  - Domain-specific side-effects
  - Data specific to framework features

Stub Generation for Autopilot

- No side-effects to unit data
  - GUI displayed machine state, used to check properties
- Look-and-feel features
  - Size, shape, color
    - Irrelevant to logical state
  - All (but one) components for Autopilot in this category
    - No buttons or widgets
    - Clickable areas
  - Empty stubs
- Relevant to logical state
  - MouseEvent coordinates X, Y
    - Can make MouseEvent part of the unit

MouseEvent Side-Effects

```
public MouseEvent(… , int x, int y, …)
{       …
         this.x = x;
         this.y = y;
      …
}
```

// must side-effects
  this.x = param4;
  this.y = param5;

Property Specification

- Pilot mental model (simple, 3 states)
  - Climb
  - Descend
  - Hold
- Map autopilot states to
  - Pilot states
- Check pilot expectations with assertions
  - If pilot expectation == climb, then the autopilot state == climb

Property Specification

```
...
public void getExpectation(){
  if(ap.mcpAltitude - ap.altitude >= 100)
    expectation = climb;
  else if(ap.altitude - ap.mcpAltitude >= 100)
    expectation = descend;
  else
    expectation = hold;
  checkExpectation();
}
public void checkExpectation(){
  Verify.assert(expectation != climb || ap.getMode() == climb);
  Verify.assert(expectation != descend || ap.getMode() == descend);
  Verify.assert(expectation != hold || ap.getMode() == hold);
}
```

Autopilot Results

- Driver specification enhanced with property
  - init; incrMCPAlt ^{1,10}; pullAltKnob ; (check; fly)^{1,10}; incrMCPVS ^{1,10}; (check; fly)^{1,10}
- Verification
  - Using JPF, successfully identified mode confusion scenarios
  - init; incMCPALT; incMCPALT; pullAltKnob; fly; fly; incMCPVS; fly
- Results
  - First GUI case study for JPF (2001)
  - Formal Analysis of Human-Automation Interaction project

Other Frameworks

- GUI applications (2004)
  - Enabledness
  - Visibility
  - Modality
- Web applications (2008)
  - J2EE
  - Fujitsu internal framework
  - Struts
- Android applications (2012)
  - Google Summer of Code projects

Related Approaches

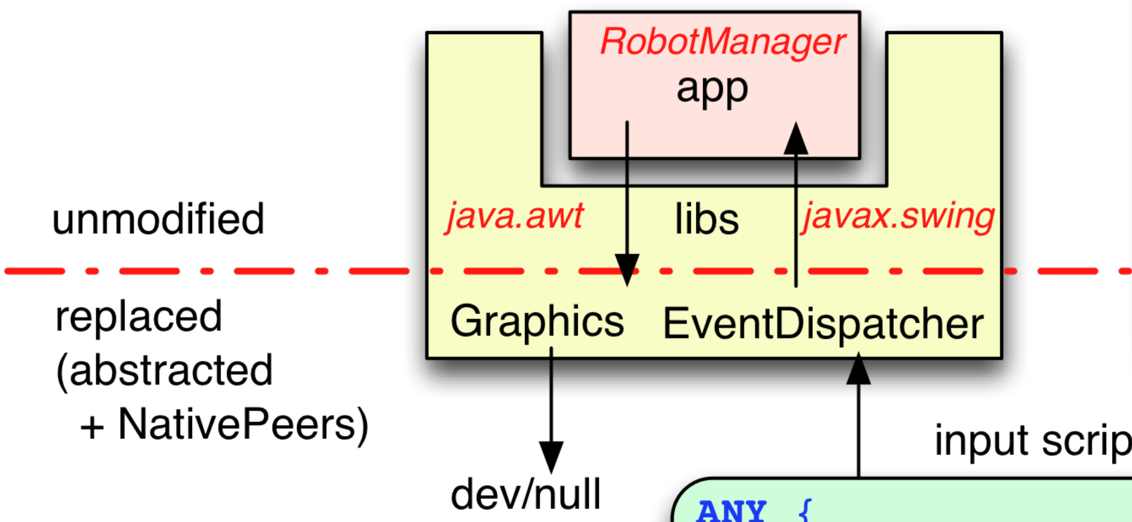
Ames Research Center

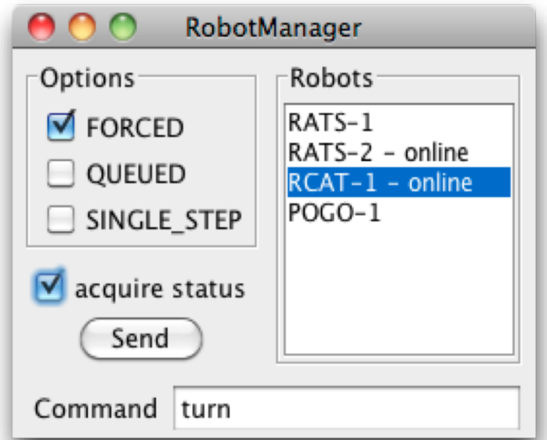
- Specifying assumptions
  - RE
  - LTL
  - Context Free Grammar
- Static analysis
  - Control effects
- Run-time analysis
  - Run the environment
  - Learn behavior from the traces
- Symbolic execution
  - Data generation
- Automated assumption generation
  - Given a unit, learn assumptions for environment
  - Learning and abstraction (Corina Pasareanu, next talk)

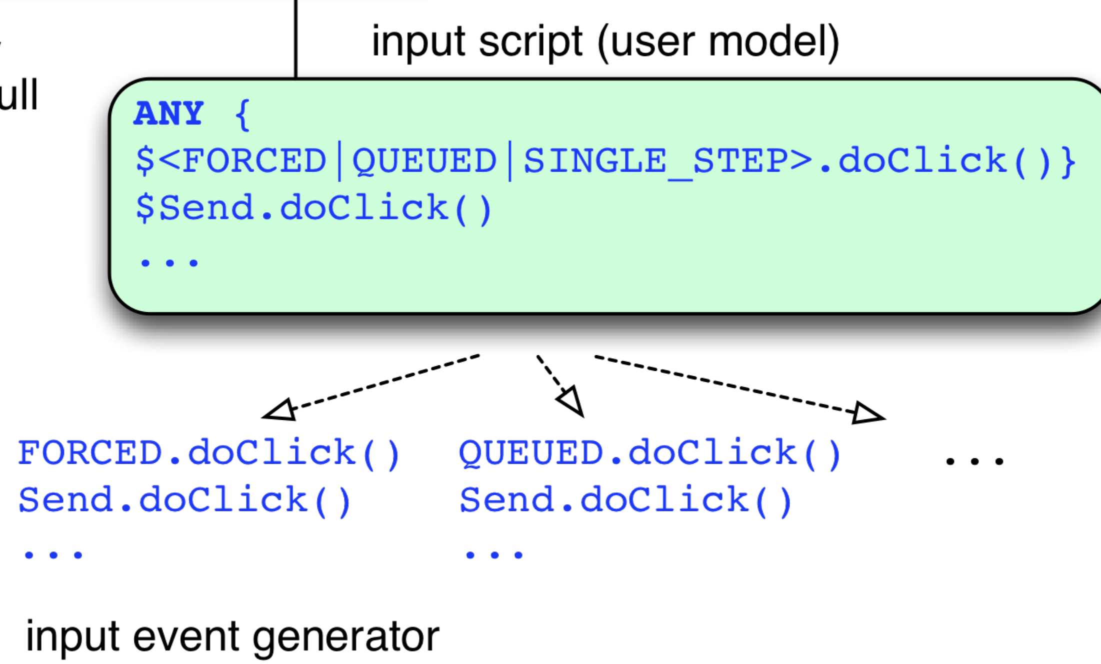
Related Approaches

- Automated
  - Universal drivers, stubs based on static analysis
    - May be over-approximate
  - Empty stubs, run-time analysis
    - May miss important behavior
- Semi-automated
  - May require manual refinement
  - Produce more precise, cost-effective models
  - Reusable
    - Library stubs
    - Cost can be amortized

JPF-AWT: Extension for GUIs







- Closes open GUI app with a user model
- Deals with libraries
  - Unmodified
  - Modeled
  - Abstracted away