Formal Methods for Java

Lecture 8: Java Pathfinder

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Nov 16, 2012

Runtime checking vs. Model checking vs. Verification

JML Tools	JPF	ESC/Java2
		KeY
		Jahob
Runtime Checking	Model Checking	Verification

Java Pathfinder (JPF)

JPF *.. the swiss army knife of Java™ verification*

http://babelfish.arc.nasa.gov/trac/jpf/wiki

- Developed at NASA Ames Research Center
- One tool – many different usage patterns
- Highly extensible core
- Core implements explicit state model checking on top of a Java VM
- Key concepts:
  - Execution choices as transition breakers
  - State matching
  - Backtracking (restoring previous state)
  - Listeners, Properties, and Publishers

History of JPF

1999	Start as front end for the Spin model checker.

. . . .

2000	Reimplementation as virtual machine
2003	Extension interfaces
2005	Open sourced on Sourceforge
since 2008	Participation in Google Summer of Code
since 2009	Project, extensions, and wiki hosted on NASA servers (still open source)

Obtaining and Building JPF

- Download from `http://babelfish.arc.nasa.gov/trac/jpf`
- Binary builds not recommended since tool still evolves
- Recommendation: use Mercurial repositories

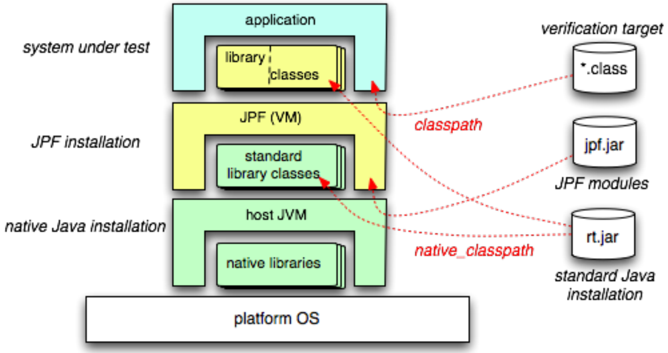
```
> hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-core
```

- Repository contains everything needed to build jpf-core

```
> bin/ant
```

- Instructions for Eclipse or NetBeans can be found in the JPF wiki

What We Got



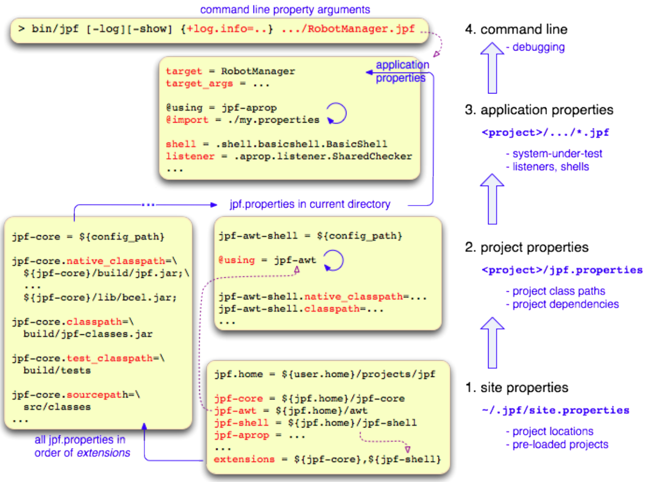
http://babelfish.arc.nasa.gov/trac/jpf/wiki

VM Inside a VM?

- JPF is written in Java $\Longrightarrow$ runs on a JVM
- JPF interprets Java Bytecode $\Longrightarrow$ acts as a JVM
- JPF operates differently:
  - Bytecode of System under Test (SUT) and
  - SUT-specific Configuration produce
  - a report and (possibly) some other artefacts (e.g., test cases)
- JPF might terminate the application if a property is violated

How to Configure JPF

JPF Configuration



```
> bin/jpf [-log][-show] {+log.info=..} .../RobotManager.jpf
```

```
target = RobotManager
target_args = ...

@using = jpf-aprop
@import = ./my.properties

shell = .shell.basicshell.BasicShell
listener = .aprop.listener.SharedChecker
...
```

```
jpf-core = ${config_path}

jpf-core.native_classpath=\
  ${jpf-core}/build/jpf.jar;\
  ...
  ${jpf-core}/lib/bcel.jar;

jpf-core.classpath=\
  build/jpf-classes.jar

jpf-core.test_classpath=\
  build/tests

jpf-core.sourcepath=\
  src/classes
...
```

```
jpf-awt-shell = ${config_path}

@using = jpf-awt

jpf-awt-shell.native_classpath=...
jpf-awt-shell.classpath=...
...
```

```
jpf.home = ${user.home}/projects/jpf

jpf-core = ${jpf.home}/jpf-core
jpf-awt = ${jpf.home}/awt
jpf-shell = ${jpf.home}/jpf-shell
jpf-aprop = ...
...
extensions = ${jpf-core},${jpf-shell}
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki

JPF Configuration Files

- Basically Java properties files:
  - `key=value` assigns `value` to `key`
  - `# This is a comment`
- Extensions:
  - `${x}` expands to current value of variable `x`
  - `key+=value` appends `value` to the value of `key`
    (No space between `key` and `+=`)
  - `+key=value` prepend `value` to the value of `key`
  - `${config_path}` expands to the directory of the currently parsed file
  - `${config}` expands to the filename of the currently parsed file
  - `@using=<project-name>` loads project `project-name` from location defined in `site.properties` with line
    `<project-name>=<project-path>`
  - ...
- Shortcut for class names: package prefix *gov.nasa.jpf* can be omitted
- Configuration of JPF can be difficult

Configuring Our Compiled Version

- Switch to your home directory
- Create folder `.jpf`
- Create file `.jpf/site.properties`

```
jpf.home = <Path where you downloaded jpf>

jpf-core = ${jpf.home}/jpf-core

extensions = ${jpf-core}
```

- This creates the basic configuration
- Add line `jpf-proj = path to site.properties` for every additional project you download

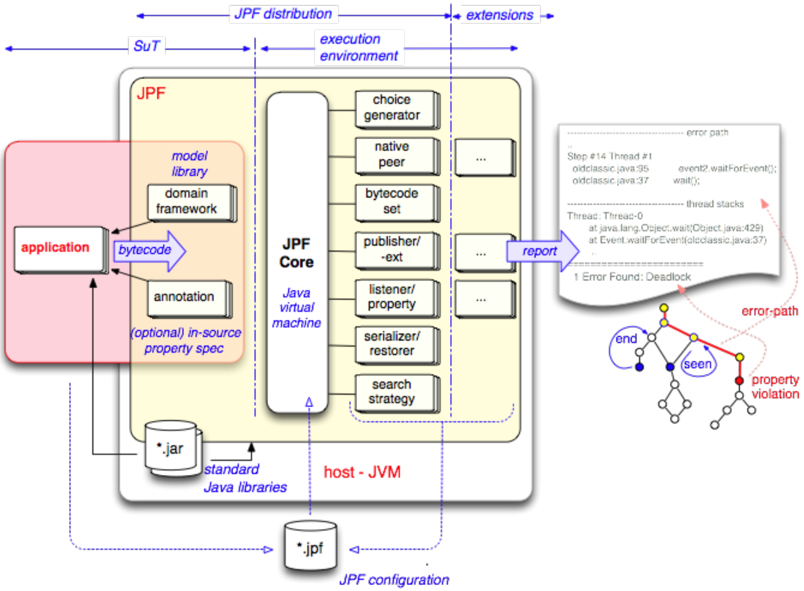
Configuring SuTs

- Create configuration file (typically ends with `.jpf`)
- Content:
  - Some `@using` directives (optionally)
  - One line `target = <SuT>`
  - Optional arguments in a line `target_args = <args>`
  - Additional JPF and related project configuration (optional)
  - Optional `classpath` entry to locate the `.class` file
  - Optional `sourcepath` entry to locate the `.java` file

Demo

Insights into JPF

JPF Components

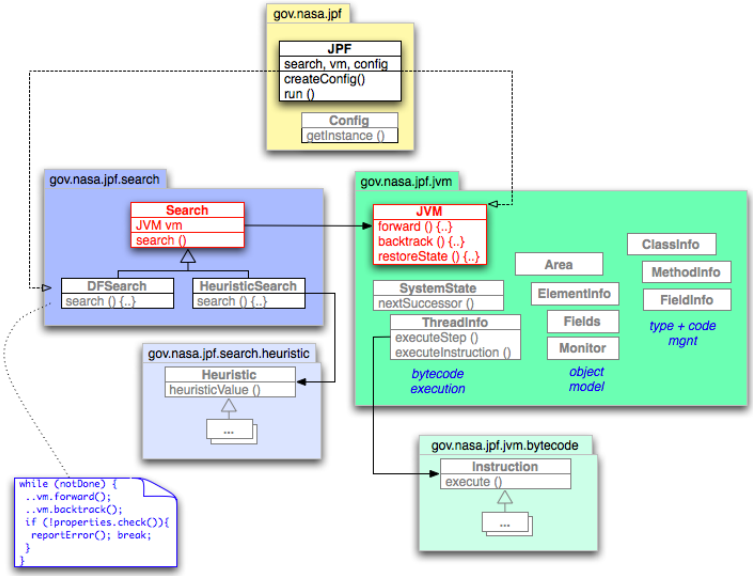


```
---------------------------------- error path
..
Step #14 Thread #1
  oldclassic.java:35         event2.waitForEvent();
  oldclassic.java:37        wait();

---------------------------------- thread stacks
Thread: Thread-0
        at java.lang.Object.wait(Object.java:429)
        at Event.waitForEvent(oldclassic.java:37)
         ..
==========================
  1 Error Found: Deadlock
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki

JPF Core Architecture



```
while (notDone) {
  ..vm.forward();
  ..vm.backtrack();
  if (!properties.check()){
   reportError(); break;
  }
}
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki

Explicit State Model Checking and JPF (1/3)

JVM

Unifies states, produces successor states, backtracks
Configurations:

vm.class	VM implementation
vm.insn_factory	instruction factory
vm.por	apply partial order reduction
vm.por.sync_detection	detect fields protected by locks
vm.gc	run garbage collection
vm.max_alloc_gc	maximal number of allocations before garbage collection
vm.tree_output	generate output for all explored paths
vm.path_output	generate program trace output
...	and many, many more

Explicit State Model Checking and JPF (2/3)

Search

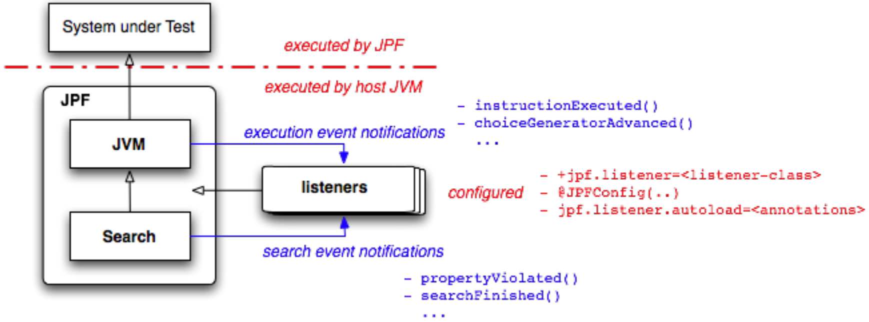
Selects next state to explore.
Configurations:

search.class	search implementation
search.depth_limit	maximal path length
search.match_depth	only unify if depth for revisit is lower than known depth
search.multiple_errors	do not stop searching at first property violation
search.properties	which properties to check during search
...	further options for each search

Explicit State Model Checking and JPF (3/3)

Listener

Evaluate states against properties.
Listeners can influence current transition while properties cannot.
Listener can monitor search and instruction execution.
Own listener can be set with the `listener` configuration option.



http://babelfish.arc.nasa.gov/trac/jpf/wiki

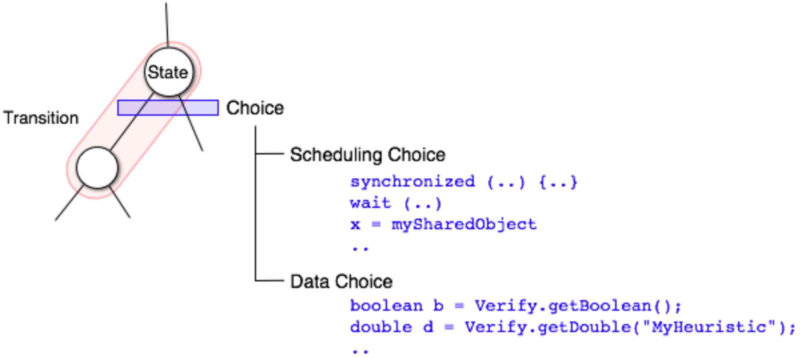
States

Collection of

- thread state (current instruction, stack),
- global variables,
- heap references, and
- trail (path to the state)

Transitions

- Sequence of instructions
- End of transition determined by
  - Multiple successor states (choices)
  - Enforced by listeners (*vm.breakTransition*();)
  - Reached maximal length (configuration `vm.max_transition_length`)
  - End or blocking of current thread



```
synchronized (..) {..}
wait (..)
x = mySharedObject
..
```

```
boolean b = Verify.getBoolean();
double d = Verify.getDouble("MyHeuristic");
..
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki