

Formal Methods for Java

Lecture 8: Java Pathfinder

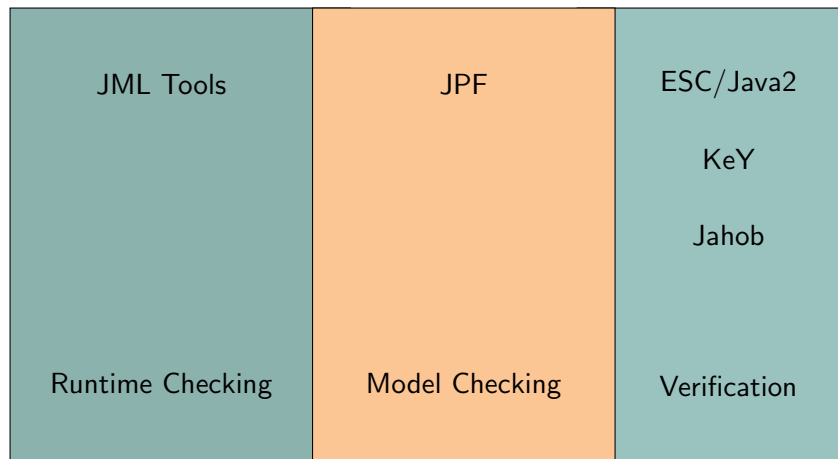
Jochen Hoenicke



Software Engineering
Albert-Ludwigs-University Freiburg

Nov 16, 2012

Runtime checking vs. Model checking vs. Verification



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

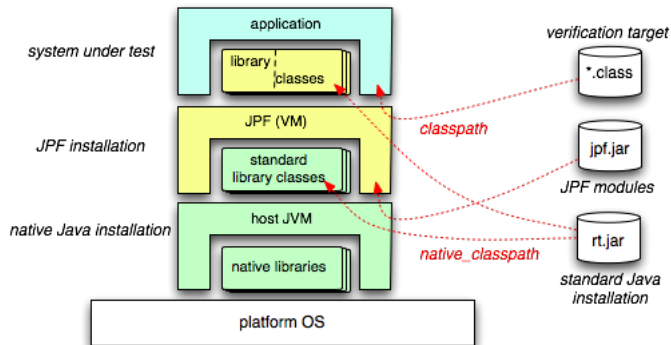
since 2008 Participation in Google Summer of Code

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



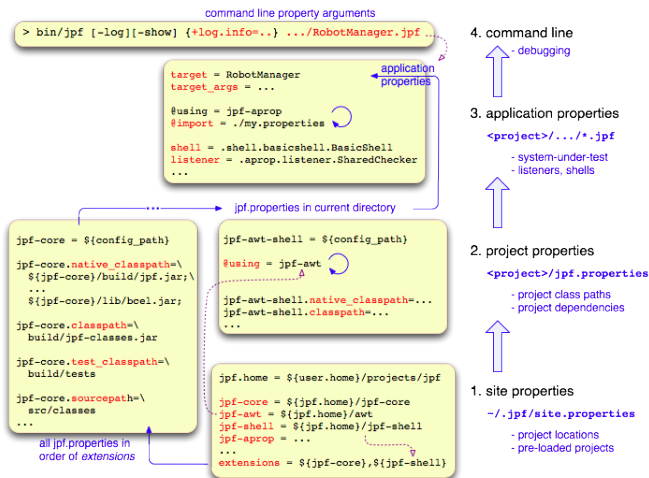
<http://babelfish.arc.nasa.gov/trac/jpf/wiki>

VM Inside a VM?

- JPF is written in Java \implies runs on a JVM
- JPF interprets Java Bytecode \implies 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



<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`

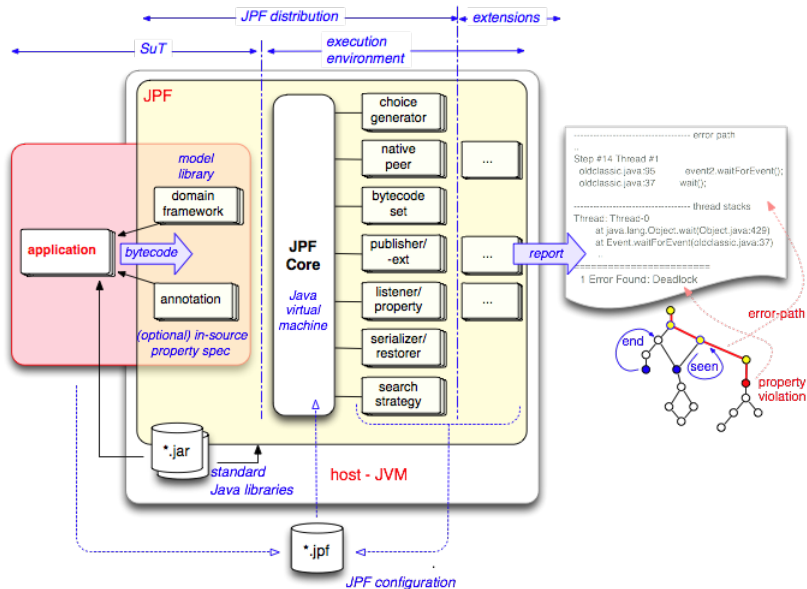
`extensions = ${jpf-core}`
- This creates the basic configuration
- Add line `jpf-proj = path to site.properties` for every additional project you download

- 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

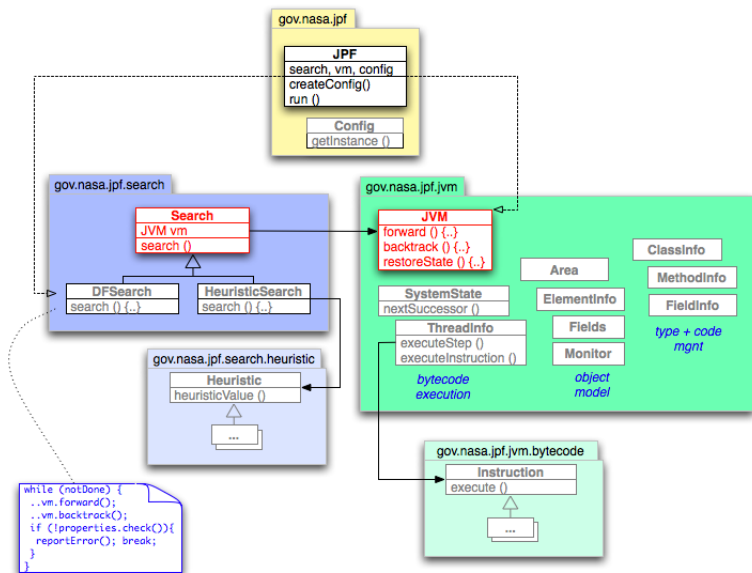
Insights into JPF

JPF Components



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

JPF Core Architecture



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

Explicit State Model Checking and JPF (1/3)

JVM

Unifies states, produces successor states, backtracks

Configurations:

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

Search

Selects next state to explore.

Configurations:

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

Explicit State Model Checking and JPF (3/3)

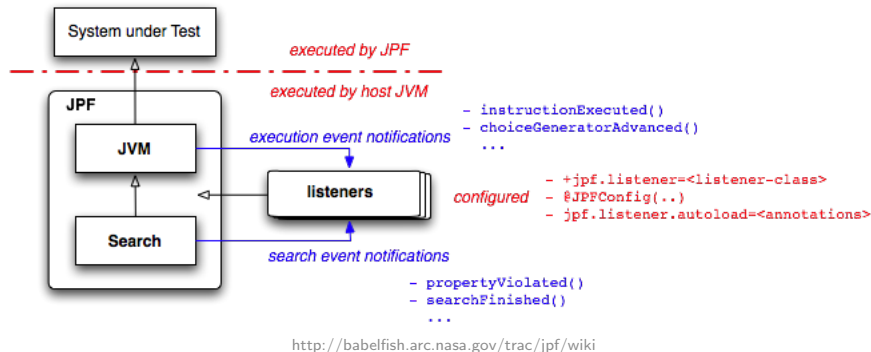
Listener

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.

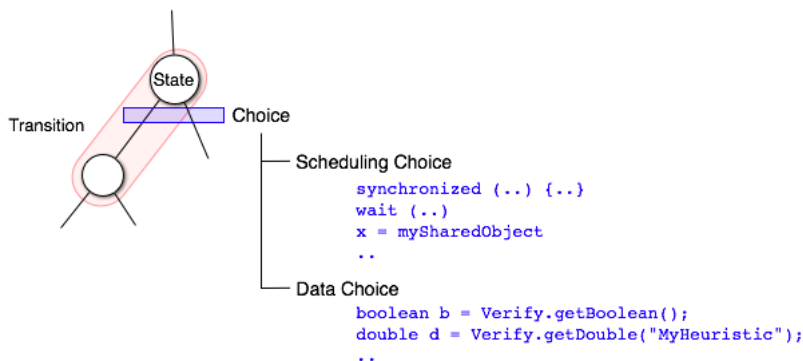


Collection of

- 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



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