

Wikiprint Book

Title: Unit Checking for Java IDE

Subject: Java Path Finder - external-projects/start

Version: 8

Date: 02/19/13 07:59:28

Table of Contents

TracNav	3
Introduction...	3
Installing JPF...	3
User Guide...	3
Developer Guide...	3
Projects...	3
About...	3
Unit Checking for Java IDE	3
LTL Listener	3
Java RaceFinder	3
jpf-nhandler	4

TracNav

- [JPFWiki](#) - Welcome Page

[Introduction...](#)

[Installing JPF...](#)

[User Guide...](#)

[Developer Guide...](#)

[Projects...](#)

- [Summer Projects](#)
- [External Projects](#)
- [Change\(B\)log](#)

[About...](#)

- [Events](#)
- [Presentations](#)
- [Papers](#)
- [FAQ](#)
- [History?](#)
- [Support](#)
- [People?](#)
- [Playground](#)
- [Table of Context](#)

Not all projects are created equal. This section of the wiki is reserved for links to and descriptions of JPF related projects that are not (yet) hosted on <http://babelfish.arc.nasa.gov>

Unit Checking for Java IDE

This is a JPF extension that allows to run JUnit tests under JPF. The key idea is to employ model checking in a way similar to unit testing, i.e. to create simple scenarios for checking small portions of software - hence the term *unit checking*. While only a single execution path is explored when a JUnit test is run under standard JVM, running the tests under JPF leads to exhaustive examination of all possible executions (including all thread interleaving).

The tool can be run from command-line or from Eclipse IDE. A plugin for Eclipse was developed, which provides nice GUI to the tool - it allows to run the unit tests and display test results in the same way as standard JUnit plugin for Eclipse.

The extension was developed by Michal Kebrt (michal.kebrt "at" gmail.com). Additional information and the source code can be found at <http://aiya.ms.mff.cuni.cz/unitchecking/dist>.

LTL Listener

`jpf-ltl` is a Java Pathfinder extension which enables the verification of temporal properties for sequential and concurrency Java programs. `jpf-ltl` aims at supporting the verification of rich-configuration events, such as method invocations with object instance aware or local and global program variables. At this point, `jpf-ltl` can verify temporal properties of method call sequences, linear relations between program variables and the combination of both.

`jpf-ltl` follows the new JPF project structure, i.e. can be easily integrated into existing JPF installations. It makes use of Dimitra Giannakopoulou's LTL2BA translator to transform LTL expressions into Büchi automata.

The extension is actively developed by Nguyen Anh Cuong (anhcuong "at" nus.edu.sg) of the University of Singapore, sources are available from <http://bitbucket.org/nacuong/jpf-ltl>.

Java RaceFinder

[JRF](#) is an precise data race detectable extension to Java PathFinder ([JPF](#)). The motivation for [JRF](#) comes from the unsound nature of JPF in the Java memory model that is relaxed and does not require sequential consistency. The main contribution of [JRF](#) is that the data race freedom verified by [JRF](#) guarantees the soundness of the proof constructed using JPF. In other words, while JPF is exploring the reachable state space of the system, [JRF](#) is able to detect race conditions allowed by the relaxed memory model. If the program is free of races, then it only consists of sequentially consistent executions, which means that JPF did not miss any behaviors of the relaxed memory model in its search.

The JRF [download page](#) includes instructions for installation. To learn more:

- [Publications](#) at the JRF home page
- [JRF Overview](#) slides by [KyungHee Kim](#) from a May 2010 presentation at NASA Ames
- JRF [JPF Implementation Overview](#) slides by [KyungHee Kim](#) from a May 2010 presentation at NASA Ames

jpf-nhandler

`jpf-nhandler` is an extension of JPF that automatically delegates the execution of the system under test methods from JPF to the host JVM. The key application of `jpf-nhandler` is to automatically intercept and handle native calls within JPF. It can also be applied on non-native calls.

The implementation of `jpf-nhandler` mostly relies on [MJI](#). It creates bytecode (and source code, at user will) for native peers on-the-fly using the BCEL library. By using `jpf-nhandler`, rather than model checking a call, the call is executed outside of JPF, in its normal environment. Other applications of our tool include mitigating the state space explosion problem, saving internal memory for JPF, and speeding up JPF.

This extension is developed by Nastaran Shafiei ([nastaran.shafiei "at" gmail.com](mailto:nastaran.shafiei@atgmail.com)) and Franck van Breugel ([franck "at" cse.yorku.ca](mailto:franck@at.cse.yorku.ca)) at the York University. Sources are available from <https://bitbucket.org/nastaran/jpf-nhandler>.