

## **Wikiprint Book**

**Title: Conformance Checker**

**Subject: Java Path Finder - summer-projects/2012-conformance**

**Version: 4**

**Date: 03/01/2013 06:13:43 PM**

## Table of Contents

<b>Conformance Checker</b>	<b>3</b>
Abstract	3
Contact	3
Repository	3
Description	3

# Conformance Checker

## Abstract

The conformance checker provides functionalities for both user and developer. This tool can be used in two different modes: on-the-fly, and as a checker for JPF. User can employ this tool on-the-fly to handle missing methods within the model classes. Developer can use the conformance checker to detect inconsistencies between JPF model classes and standard ones, and also to create model classes from scratch based on static analysis of the system under test (SUT).

## Contact

student: Matteo Ceccarello <matteo.ceccarello AT gmail.com>

mentor: Nastaran Shafiei <nastaran.shafiei AT gmail.com>

co-mentor: Franck van Breugel <franck AT cse.yorku.ca >

## Repository

The sources for this project are available from a Mercurial repository at <https://bitbucket.org/nastaran/conformance-checker>

## Description

Project documentation/wiki/blog are available at TBD

The implementation of this project is divided into three phases which are as follows,

- Checking JPF: Any inconsistencies between the standard class, the corresponding model class, and the corresponding peer class should be checked.
- Automatically analyzing the compatibility of the current JPF verification system (including jpf-core and its extensions) compared to Java 1.5, 1.6, and 1.7 standards. That requires comparing each model class, its superclasses, and all implemented interfaces with the standard ones
- For each model class, checking if the state of the object can be captured by the same fields (including the ones in the superclasses) in both JPF and JVM environment
- For the classes that have native peers, detecting the native methods that their corresponding peer methods are missing, and also checking for orphans public native peers
- Providing a report for each JKD that outlines the detected inconsistencies
- Integrating the check with JUnit based unit tests to run it as a standalone tool or along with the other JPF unit tests
- Creating model classes based on static analysis: This phase of the project includes creating model classes from scratch, and adding method stubs and fields on demand. First the SUT is checked and information about fields and methods accessed by the SUT are retrieved. Using this information, the source for the missing model classes created, optionally with their superclasses and interfaces.
- Handling missing methods on-the-fly: If JPF runs into a case where the invoked method is not declared in the model class, but it is declared in the standard class, the conformance checker handles it on-the-fly by creating an empty stub for the method.