JPF-Core-X:
Tool Qualification Plan (TQP)

Contents

Introduction

This document is one of a several exemplar documents prepared as part of a research Case Study whose objective is to simulate a Formal Methods Tool qualification exercise under DO-330. The specific tool considered in this case study is the core module of Java PathFinder (JPF-Core). As with any tool qualification exercise, the qualification is done with respect to a specific version of the tool. Therefore, throughout this document, we refer to our version of JPF-Core as "JPF-Core-X", as described in Section a of the Tool Qualification Plan. This particular document provides a representative example of the "Tool Qualification Plan" (TQP) for JPF-Core-X, and is written according to the guidelines in DO-330 Section 10.1.2.

Because this is a research study, in which there is no actual qualifying organization and accompanying context, and because the tool under consideration is a research implementation without specific versions, release control, user documentation, or standard configurations, some of the sections of an actual TQP will not be relevant. This document is thus a Framework for an actual TQP, organized according to the DO-330 enumeration of required contents. Accordingly, some sections will represent content that is concrete enough to be part of an actual TQP; some will discuss how a more concrete implementation of this tool might fulfill the required contents; and some will not be applicable for the purposes of this research simulation.

The sections that follow are organized according to sub-parts a) through j) of Section 10.1.2 in DO-330. In each section, we attempt to provide representative content of what should appear in an actual TQP for a real qualification exercise. In addition, we offer supplemental *meta-level* comments throughout the document.

Discussion

This is how a meta-level comment appears in the text. These comments are meant to provide insight into our process of writing the document, and to suggest interesting or important topics that relate to the qualification of formal methods tools.

a Identification of the Tool

DO-330

"Identification of the tool, and, if applicable, user configuration." [DO330-10.1.2-a]

Discussion

Java PathFinder presents multiple challenges to tool identification for qualification purposes. First, as a result of its long tenure as a research project, it has accumulated many related capabilities at various maturity levels. For the most part, these capabilities are segregated by the use of "modules" in the JPF source control repository. However, to be conservative, a tool

qualification package should specify both which modules will be used and which capabilities within the modules will be exercised. Second, as a non-commercial project there is no single commercial entity which controls the release of a consistent, integrated, and well-tested tool. Third, as an open-source tool, JPF is distributed as a set of source files, which depend on a build environment to create a correct executable.
For the purpose of this case study, we propose the following approach to overcoming these challenges. First, we identify a single code directory (*i.e.*, module), "jpf-core" to be qualified. Second, we specify a date, repository, tag, and changeset to unambiguously define a particular tool release. We give this particular release the name JPF-Core-X and treat it as the tool to be qualified. We do not attempt to solve the problem of specifying a build environment for JPF-Core-X, but suggest that this is an interesting issue for qualification of tools which are distributed as source.

Java PathFinder (JPF) is a model-checking tool that can perform explicit state model checking to check for errors over all possible state values in all possible paths of the program. As such, JPF can be used to verify the requirements of a system when those requirements are expressed in Java source code.

The tool to be qualified for certification is "JPF-Core-X". We obtained the core module of JPF (JPF-Core) from the JPF mercurial repository maintained by NASA on March 1, 2016. We refer to this version of the software as JPF-Core-X. On that date, JPF-Core-X was the `tip` tag on the `default` branch of the repository at `http://babelfish.arc.nasa.gov/hg/jpf/jpf-core`. The last changeset included in JPF-Core-X is `29:820b89dd6c97` committed on October 16, 2015.

b Qualification Considerations

DO-330

"Qualification considerations, including the proposed TQL and means of compliance with the objectives of this document." [DO330-10.1.2-b]

b.1. Proposed TQL

As described in the tool specific sections of the PSAC, section d, we propose to qualify JPF-Core-X at TQL-4.

b.2. Means of compliance

The means of compliance with the objectives of DO-330 are set out in the Case Study Preparation document Tables T-0 through T-10.

c Functional Overview

DO-330

"A functional overview of the tool, its interfaces, and its architecture. Additionally, any external components should be identified." [DO330-10.1.2-c]

Discussion

This section overlaps with Section "a.1" in the TOR.

Discussion

JPF is a configurable environment with its own virtual machine designed to enable customized verification of Java bytecode programs. The full JPF architecture includes a core virtual machine with a basic set of verification tools (JPF Core), plus several *optional* extension modules that may be added to perform more customized analysis. As discussed in Section a, however, this qualification exercise is for JPF-Core only.

Figure 1 shows JPF-Core-X's inputs and outputs in the context of its proposed use in certification activities. As shown, JPF-Core-X consumes files containing properties to be verified represented in its own input language and one or more files containing Java code representing system LLRs. It's output is presented to the user in the form of a report file.

A diagram of the architecture is shown in Figure 2. The JPF-Core-X installation resides and runs on top of the native Java installation on the host OS. It is therefore a VM on top of a VM. The JPF virtual machine then executes the Java application being tested. This Java source code being checked by JPF-Core-X is also referred to as the System Under Test (SUT).

In general, a potential challenge with running an application on JPF-Core-X (especially large scale applications) is that JPF-Core-X cannot execute Java libraries that use native code. Doing so would prevent the tool from matching and/or backtracking the program states. The workaround is to use *native peers* and/or *model classes*.

Native peers are Java classes that effectively replace native methods. The native peers are executed by the real Java VM (not JPF-Core-X). *Model classes* are simple replacements of standard classes, such as java.lang.Thread. The model classes provide alternatives for native methods which are fully observable and backtrackable.

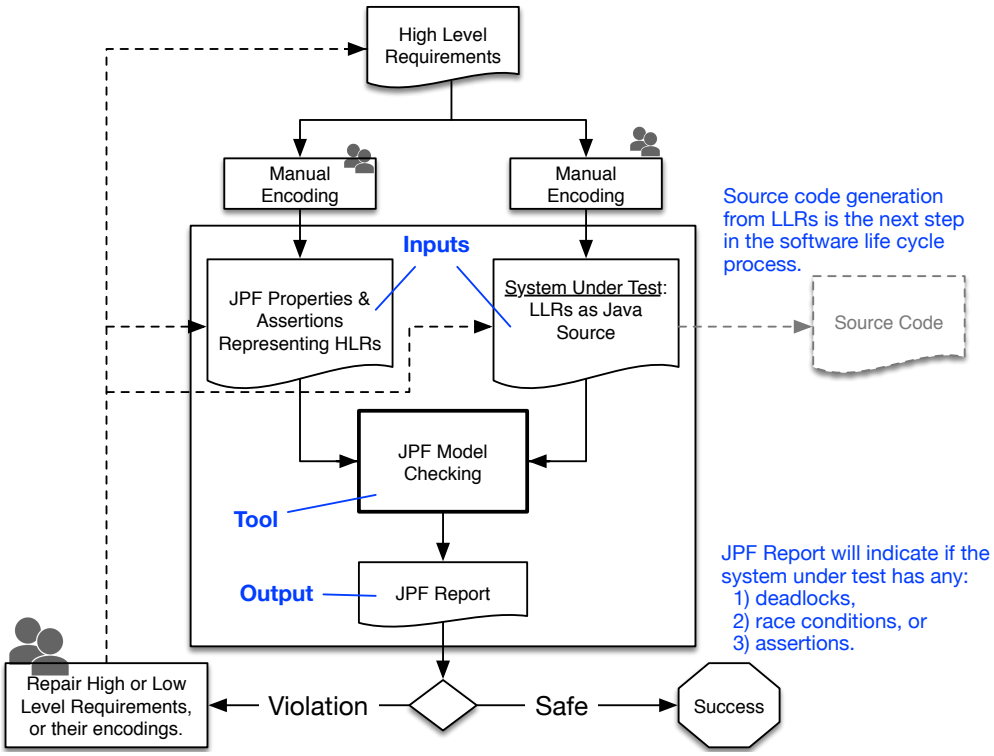

Figure 1: Flow of HLRs and LLRs for JPF-Core-X model checking. (draft graphic)

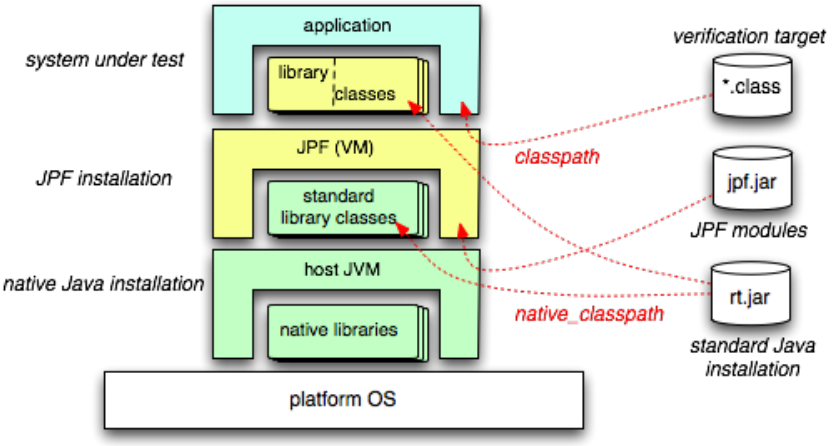

Figure 2: JPF Layered Architecture. Image taken from: http://babelfish.arc.nasa.gov/trac/jpf/wiki/user/components

d Tool Operational Environment

DO-330

"Description of the tool operational environment(s), and if different, the tool verification environment(s)." [DO330-10.1.2-d]

JPF-Core-X is a pure Java application. As such, it runs on the Java virtual machine, which itself can be run on Windows, OS X, or Unix operating systems. The minimal required Java version is Java SE 7, which corresponds to JDK 1.7.

Discussion

In a real qualification, the exact set of OS versions for which the tool is qualified should be listed. Note that there is nothing different or unique about defining the host OS requirements for formal methods tools as compared to any other tool.

JPF-Core-X is provided with project configurations for both the NetBeans and Eclipse IDEs. Either IDE may be used to define the project settings, edit the JPF properties, and run JPF.

Given that JPF-Core-X is generally a resource hungry application, it is recommended to run with a minimum of 16 GB of RAM and a minimum 2.3 GHz dual-core processor. These recommendations are based upon the observed performance of running JPF on applications that are representative in size and complexity of the software to be certified. The actual memory usage will depend on the actual application being tested.

Discussion

JPF, like all model checkers, tends to be a resource hungry application. In order to ensure the tool can successfully complete evaluations of arbitrarily large applications, it must be given sufficient resources in terms of CPU and memory. In a real qualification, the requirements on these hardware resources should be evaluated for the specific application, and those requirements should be reported in this section of the TQP.

Discussion

This section overlaps with Section "b" in the TOR.

e Visibility of Tool Life Cycle Process Activities

DO-330

"Description of the means the applicant will use to provide the certification authority with visibility of the activities of the tool life cycle processes so tool reviews can be planned." [DO330-10.1.2-e]

THIS SPACE INTENTIONALLY LEFT BLANK.

Discussion

For the purposes of the case study, the tool life cycle process activities have been identified and described in the Tables T-0 through T-10 of the "Case Study Preparation Document." There is nothing about this section of the TQP of special interest in the case where the tool to be qualified is a formal methods tool.

f Tool Life Cycle

DO-330

"Tool life cycle description and the qualification activities to be performed." [DO330-10.1.2-f]

THIS SPACE INTENTIONALLY LEFT BLANK.

Discussion

For the purposes of the case study, the tool life cycle process activities have been identified and described in the Tables T-0 through T-10 of the "Case Study Preparation Document." There is nothing about this section of the TQP of special interest in the case where the tool to be qualified is a formal methods tool.

g Tool Qualification Data

DO-330

"Tool qualification data to be produced." [DO330-10.1.2-g]

The main qualification data to be produced will be presented in two documents:

1. Test Cases and Procedures

2. Test Results

h Additional Considerations

DO-330

"Any additional considerations that may affect the qualification process, for example, deactivated code, COTS tools, reuse, tool qualification (of other tools used to develop or verify the tool), alternate means of qualification, tool service history, and means to ensure the determinism of the tool per the last paragraph of section 2.0 of this document." [DO330-10.1.2-h]

Discussion

JPF search functions could be configured to intentionally introduce non-determinism. However, this would only change the path taken internally to explore the state space, in an effort to explore it faster. JPF should still cover all paths regardless of the search method. Therefore, the output, which reports errors or confirms the absence of errors, should not change even if the search process is non-deterministic. This is consistent with the requirement of determinism specified in the last paragraph of section 2.0 of DO-330.

i Organizational Responsibilities

DO-330

"Organizational responsibilities within the tool life cycle processes. " [DO330-10.1.2-i]

THIS SPACE INTENTIONALLY LEFT BLANK.

Discussion

There is nothing about this section of the TQP of special interest in the case where the tool to be qualified is a formal methods tool.

j Suppliers

DO-330

"If suppliers are used, a means of ensuring that supplier processes and outputs will comply with approved tool plans and standards. " [DO330-10.1.2-j]

THIS SPACE INTENTIONALLY LEFT BLANK.

Discussion

There is nothing about this section of the TQP of special interest in the case where the tool to be qualified is a formal methods tool.

References