JPF Visual

Cyrille Artho
KTH Royal Institute of Technology
Lindstedtsvägen 3, level 5
Stockholm 100 44
artho@kth.se

Qiyi Tang
York University
4700 Keele Street
Toronto M3J 1P3
qiyitang@cse.yorku.ca

ABSTRACT

Reading error traces produced by Java Pathfinder (JPF) can be difficult. JPF provides the traces in textual format, with a lot of condensed information. Error traces can be long, with many transitions containing code that is concerned with initializing the system under test, but does not contribute to a failure.

We provide a visual analytics tool for error traces produced by JPF. A tabular layout shows clearly which thread is active at which time, and dynamic filtering allows users to highlight information that may be related to the failure. Transitions can be collapsed or expanded to allow “zooming in” to a problem. We show in examples how filtering and zooming in can highlight a problem without having to read lengthy transition code.

CCS CONCEPTS
• Software and its engineering → Integrated and visual development environments; • Human-centered computing → Visual analytics;

KEYWORDS
Java Pathfinder, trace visualization, visual analytics

 ACM Reference format:
https://doi.org/10.475/123_4

1 INTRODUCTION

Java Pathfinder (JPF) [7] is a program analysis tool for Java programs. Its most common use is to model check a system given as Java bytecode, to explore all possible outcomes of non-determinism in the program. Non-determinism mainly arises from scheduling multiple concurrent threads.

JPF checks a given program against uncaught exceptions, deadlocks, and assertion violations. Additional properties can be checked by adding more assertions or specifying listeners. When JPF finds a property violation, it reports this as an error trace.

Error traces are shown in a textual format, which includes the full detail of the execution. For any non-trivial error, traces generated by JPF can be quite long, exceeding hundreds of transitions and thousands of lines of text.

Figure 1 shows a partial trace of the Dining Philosophers [5] example. As can be seen, the trace is cluttered with details such as parsing command line parameters. The first twelve transitions merely initialize all threads; the actual part that leads to a deadlock begins with transition 13. The only clue to a user that transition 13 is the first “interesting” transition is that the ID of the active thread changes from 0 to 1, something that is easily overlooked when analyzing long traces.

We present a pragmatic visualization that adopts a tabular layout to highlight the current thread spatially, while keeping the option to show the original (detailed) transition information in full text. We furthermore have a facility that selectively highlights and filters out operations that are expected to contribute to the failure, allowing the user to mine a trace interactively and understand its meaning using visual analytics [4].

2 JPF-VISUAL

Our visual analytics tool is implemented as an additional panel in the JPF shell [1]. It has a navigation panel on the left, which allows a user to expand/collapse all transitions at once, or to highlight individual aspects of all transitions. Operations that can be highlighted

Unpublished working draft. Not for distribution

for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright and rights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

JPF ’17, November 2017, USA.
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5677-2/17/11...
https://doi.org/10.475/123_4

116
include wait/notify, starting new threads and joining a thread, lock usage, field accesses, and method calls. The latter two options are accessed by a drop-down list (see Figure 2).

Highlighted parts of the source code are shown in up to 15 distinct colors, both in the expanded and collapsed modes.

On the right side, a "world map" (overview) is shown. This view can simplify the navigation in large traces and also supply a condensed representation of key thread attributes [6].

The users starts with a fully collapsed summary, which shows only the first and last line of code for each set of transitions that gets executed by the same thread. Typical selections for highlighting include "(un)lock" for debugging traces highlighting a data race or deadlock, and field accesses or method calls for more complex problems such as NullPointerExceptions. With each selection, the central panel is automatically updated; the summarized error trace is immediately visible. The collapsed summary hides the command line parsing that reduces the number of threads to five.

3 CONCLUSION AND FUTURE WORK

Our visual analytics tool "jpf-visual" allows a user to zoom in and highlight parts of JPF error traces. This makes it possible to look at parts of the trace in higher detail, and provides an interactive, dynamic visualization.

We plan to refine and extend the visualization in various ways. The current version of field/method call highlighting is limited to static information from the available data in the error trace of JPF. A more detailed highlighting could distinguish different object instances (rather than just types), but this would require using more detailed data. The trace server [2] could provide such information, once ported to JPF 8.

Furthermore, we want to add more "zoom levels" for transitions, such as a low-level view with individual bytecode instructions [8], and alternative views such as a diagrammatic visualization [3]. The "world map" will also be enriched with some key information on threads, such as their state.

REFERENCES