TIE: An Interactive Visualization of Thread Interleavings

Gowritharan Maheswara, Jeremy S. Bradbury, Christopher Collins
Faculty of Science (Computer Science)
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
gowritharan.maheswara@mycampus.uoit.ca, {jeremy.bradbury, christopher.collins}@uoit.ca

ABSTRACT

Multi-core processors have become increasingly prevalent, driving a software shift toward concurrent programs which best utilize these processors. Testing and debugging concurrent programs is difficult due to the many different ways threads can interleave. One solution to testing concurrent software is to use tools, such as NASA's Java PathFinder (JPF), to explore the thread interleaving space. Although tools such as JPF provide comprehensive data about program errors, the data is generally in the form of bulk text logs, which provide little support for common analysis tasks, such as finding common and rare error states. In this paper, we present an interactive visualization tool, TIE, that integrates with JPF to enhance concurrency testing and debugging.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.4 [**Software Engineering**]: Software/Program Verification—*model checking*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*graphical user interfaces*

General Terms

Verification.

Keywords

concurrency, debugging, model checking, testing, visualization.

1. INTRODUCTION

Concurrent programming is becoming increasingly popular in software development since it allows developers to take full advantage of multi-core processors. Due to the many different interleavings (*i.e.*, nondeterministic behaviour) of concurrent programs, testing has become one of the major challenges with this style of programming.

Concurrency testing and debugging tools often involve techniques like path or state space exploration in order to ensure that a given concurrent program works under many different interleaving scenarios. Two examples of concurrency testing tools that explore thread interleavings are:

1. *Microsoft CHESS* [3]: a concurrency testing tool that can find and reproduce rare error states in concurrent .NET programs. CHESS works with two other tools, Chessboard and Concurrency Explorer, to create an interactive visualization of error data. One of the limitations of this visualization is that only one error path (*i.e.*, thread interleaving) can be visualized at any given time.

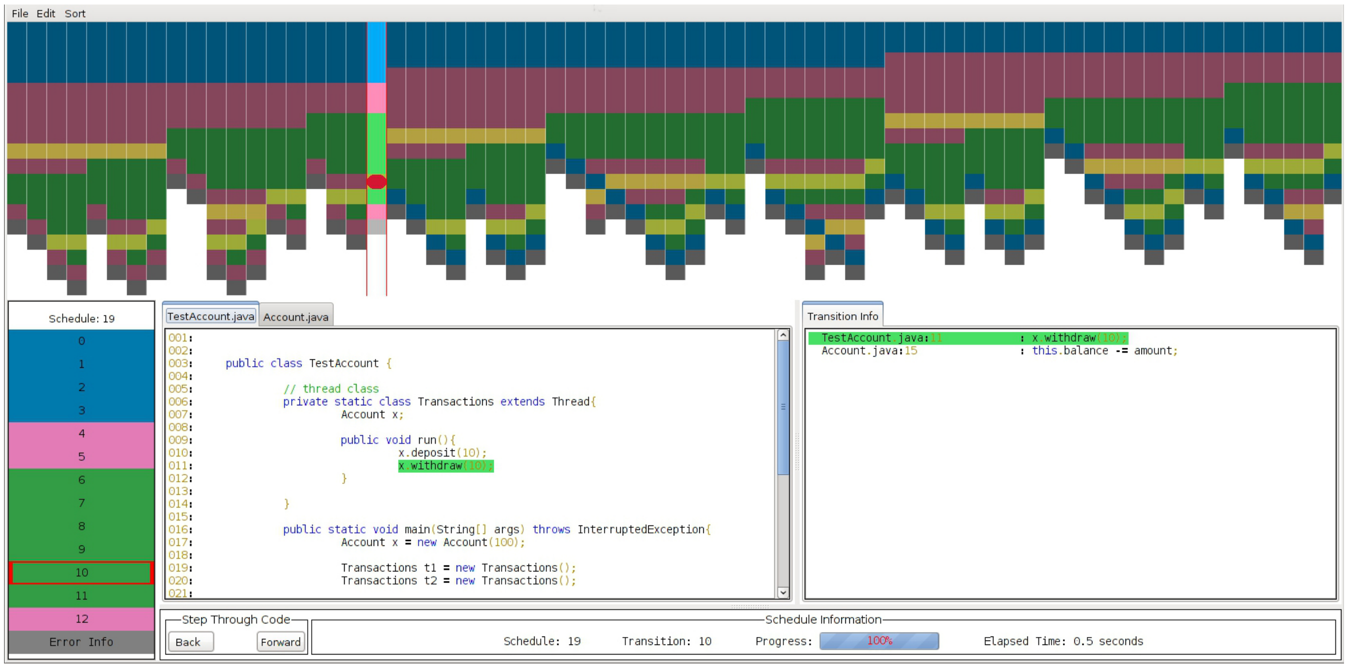
2. *NASA's Java PathFinder (JPF)* [1]: an explicit state model checker for Java byte code programs that focuses on finding concurrency related defects such as deadlocks and data race conditions. JPF determines if a program has a concurrency bug by searching all possible thread interleavings of a concurrent program. One of the limitations of JPF is the lack of a graphical interface for presenting the error output.

This paper introduces the Thread Interleaving Explorer (TIE), an interactive visualization tool to help software developers debug their concurrent programs that can be run with either the command-line or Eclipse version of JPF. We next describe the visual and interaction design of TIE, and conclude with a discussion of future research plans.

2. DESIGN OF TIE

We propose a solution to the analytic limitations of current thread interleaving outputs by introducing the TIE, a system to support interactive visual analysis of JPF's output. TIE is similar to Microsoft's Concurrency Explorer visualization, however it can scale to a larger number of interleavings and provides additional functionality. TIE is implemented as a JPF extension and thus directly integrates with JPF and retrieves information on threads, interleavings, and error paths dynamically while a concurrent program is tested. As this information is being retrieved, TIE populates the top panel of the user interface (see Figure 1). This panel provides a high-level overview of all interleaving sequences which caused a bug to occur. The bottom portion of the interface can be used to focus on one specific interleaving in order to more deeply understand a specific execution sequence.

In the top panel, each column represents a specific schedule that leads to a concurrency error. Each block within a column represents a transition in the specific schedule. Each thread in a concurrent program being tested is assigned a unique hue from a brightness-normalized, qualitative colour sequence. Unique hues allow an analyst to distinguish which



```
    public class TestAccount {

            // thread class
            private static class Transactions extends Thread{
                    Account x;

                    public void run(){
                            x.deposit(10);
                            x.withdraw(10);
                    }

            }

            public static void main(String[] args) throws InterruptedException{
                    Account x = new Account(100);

                    Transactions t1 = new Transactions();
                    Transactions t2 = new Transactions();
```

```
TestAccount.java:11              : x.withdraw(10);
Account.java:15                  : this.balance -= amount;
```

```
Schedule: 19      Transition: 10      Elapsed Time: 0.5 seconds
```

Figure 1: The Thread Interleaving Explorer (TIE)

threads execute at any given transition. The last block in a column represents the error resulting from the interleaving.

In the lower portion of the interface, an analyst can focus on one particular thread interleaving and can view both the source code and the raw JPF error data that corresponds with each transition. Both the source code and error data include syntax highlighting to improve readability.

3. INTERACTION DETAILS

Top Panel. If any specific transition on the top panel is selected, the whole schedule (column) that the transition belongs to is presented in greater detail in the lower-left panel. In order to focus the analyst's attention while maintaining a consistent hue for each thread, the selected schedule is rendered with brighter variants of the thread hues while all other schedules are rendered with darker shades. Within the selected schedule, the specific transition that is selected is distinguished by a red oval. If the "Highlight" function is selected from the Edit menu, all other transitions in any schedule that have the same executed code will be overlaid with a semi-transparent gold square, allowing patterns of code replication to be revealed while maintaining thread-specific colouring. The "Sort" function sorts the schedules in ascending order based on the total number of transitions.

Center Panels. Once the lower-left panel is populated with a set of transitions, the analyst can select any transition within that schedule to bring up source code in the center panel, and raw JPF transition information in the far right panel. The analyst can click the transitions to explore the selected sequence of interleavings. Each time a transition is selected, the corresponding source code and JPF output will be highlighted.

Bottom Panel. This panel supports sequential stepping through the selected interleaving sequence. Using the step through feature results in the same highlighting of source code and JPF output as direct selection of transition states. Details of the current JPF execution are also provided here.

4. CONCLUSIONS

We have described the design of TIE — an interactive visualization tool that extends JPF to support analysis of error states produced during testing of concurrent programs. Our work builds upon previous visualizations of concurrent programs [2, 4, 5]. However, while most previous work has focused on visualizing a single execution of a concurrent program, our design focuses on exploring multiple thread interleaving sequences. Furthermore, our design helps with revealing patterns of common errors and supporting drill down to discover specific problems in the source code. While we instantiate TIE with JPF, the general technique can be applied to other concurrency testing suites. In the future we plan to investigate additional sorting functions, such as sorting by sequence similarity. We will also study and refine the technique through deployment to the JPF community.

5. REFERENCES

[1] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *Int. J. on Software Tools for Technology Transfer (STTT)*, 2(4), 2000.

[2] E. Kraemer and J. T. Stasko. Creating an accurate portrayal of concurrent executions. *IEEE Concurrency*, 6(1):36–46, 1998.

[3] M. Musuvathi. Systematic concurrency testing using CHESS. In *Proc. of PADTAD*, 2008.

[4] S. P. Reiss and S. Karumuri. Visualizing threads, transactions and tasks. In *Proc. of PASTE*, pages 9–16, 2010.

[5] S. P. Reiss and M. Renieris. Demonstration of JIVE and JOVE: Java as it happens. In *Proc. of ICSE*, pages 662–663, 2005.