Debuggowanie aplikacji kominukujących się asynchronicznie oparte o historię komunikatów.

History-based approach for debugging applications using asynchronous communication.

Promotor: dr inż. A. Byrski

Krzysztof Romanowski

Agenda

- ➔ Cel pracy
- ➔ Stan wiedzy oraz literatura
- ➔ Problem
  - ➔ Aplikacje asynchroniczne
  - ➔ Debuggowanie
  - ➔ Modele zbierania danych
- ➔ Harmonogram pracy
- ➔ Spis treści

Cel pracy

Próbuje opisać i zaimplementować nowe podeście do debuggowania aplikacji asychronicznych działających na JVM.

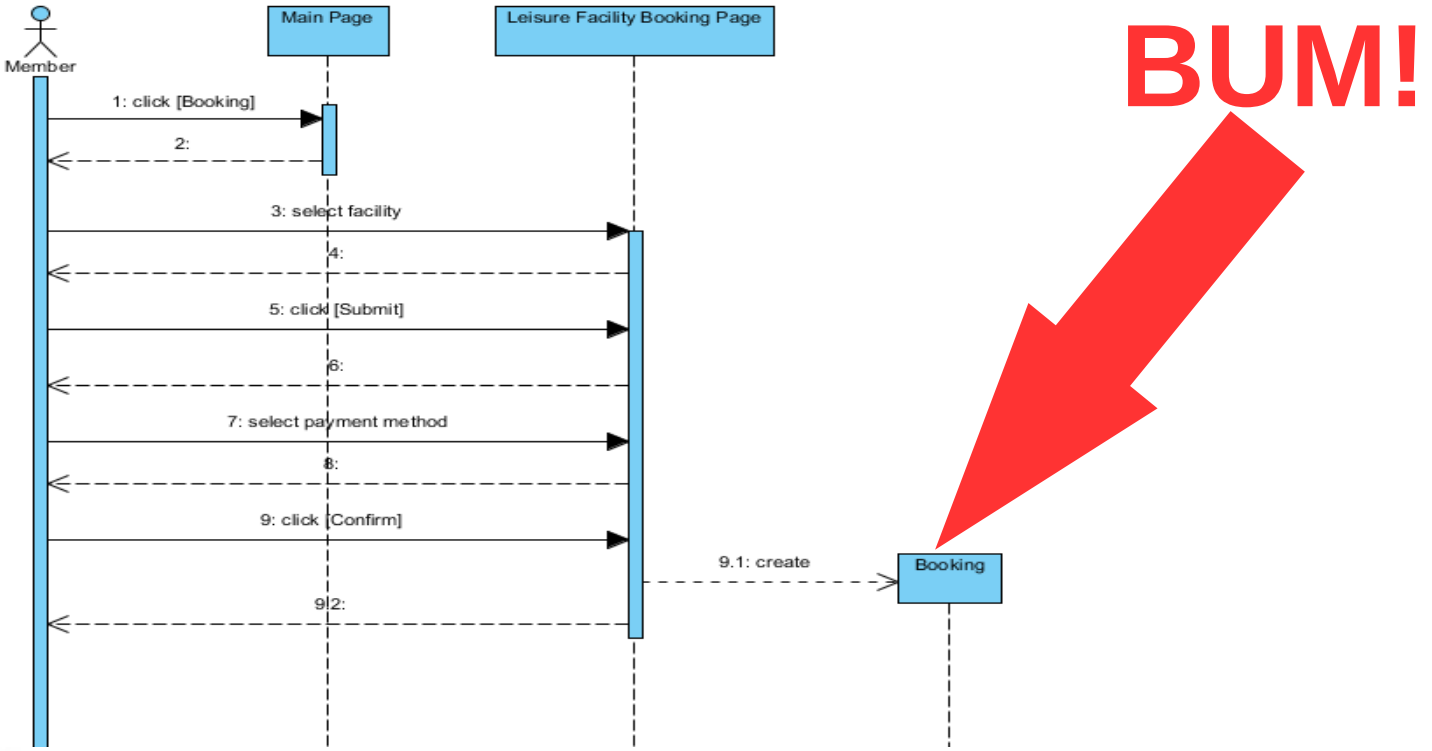
Wykorzystam podeście post-mortem ale celem jest zachowanie właściwości stadnadrowego debuggera.

Stan wiedzy

Prace nad takim debuggerem rozpoczął Iulian Dragos jako dodatek do ScalaIDE.

Jego podejście było jednak bardzo proste i nawine ale pokazało że praca taka jak moja ma sens.

Problem


Member
Main Page
Leisure Facility Booking Page
1: click [Booking]
2:
3: select facility
4:
5: click [Submit]
6:
7: select payment method
8:
9: click [Confirm]
9.1: create
Booking
9.2:
BUM!

Problem

```
java.lang.ArithmeticException: / by zero
        at com.micronautics.akka.dispatch.futureScala.FallbackTo$$anonfun$1.apply$mcI$sp(FallbackTo.scala:11)
        at com.micronautics.akka.dispatch.futureScala.FallbackTo$$anonfun$1.apply(FallbackTo.scala:11)
        at com.micronautics.akka.dispatch.futureScala.FallbackTo$$anonfun$1.apply(FallbackTo.scala:11)
        at akka.dispatch.Future$$anon$2.liftedTree1$1(Future.scala:168)
        at akka.dispatch.Future$$anon$2.run(Future.scala:167)
        at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
        at java.lang.Thread.run(Thread.java:662)
```

Kto nam wysłał to 0 ?

Rozwiązanie

- Dla danego frameworku określamy miejsca „zmiany kontekstu wykonania”
  - ! w dla Akki
  - tworzenie Futura itp..
- W momencie wysłania wiadomości – zapamiętujemy stack trace (ew. stan) w strukturze
  (czas, id_otrzymanej_wiadomości, wiadomoś)
- Ustawiamy breakpoint
- Gdy się na nim zatrzymamy – analizujemy historię wstecz

Problemy

1. Wydajność
2. Niejednoznaczność
3. Konieczność np. intrumentalizacji JVM czy redefinicji klass

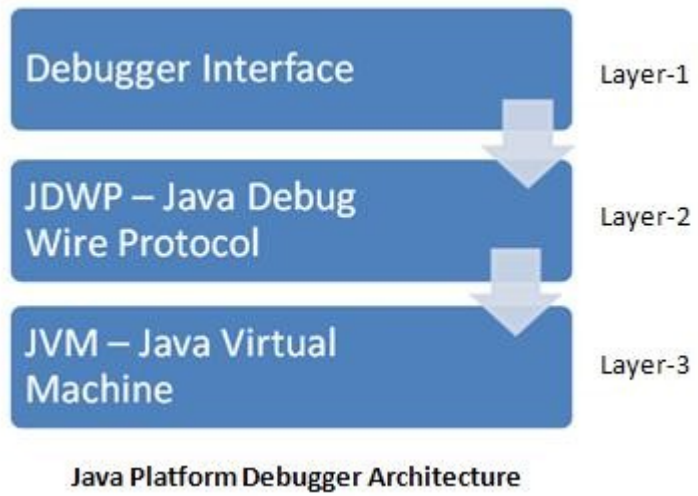
Zbieranie danych

Podejścia:

1. Naiwne oparte od JDI – podejście Iuliana
2. JDI oparte o redefinicje klas
3. JavaAgent i instrumentalizacja
4. AspectJ ?

Celem jest jak najmnieszy impakt w wykonywanie programu.

Technologie



Java Platform Debugger Architecture

Harmonogram Pracy

Do końca czerwca 2014

- Anializator
- Zbieranie przez JDI

Do końca lipca 2014

- Redefinicje
- wstęp i podstawy teoretycznie

Do końca sierpnia 2014

- Intrumentalizacja i AspectJ
- opis rozwiązania

Do końca września 2014

- Dokończenie pracy
- Obrona

Spis Treści

Bibliografia

1. Specyfikacja JDI

http://docs.oracle.com/javase/1.5.0/docs/guide/jpda/jdi/

2. Specyfikacja JVMTI

http://docs.oracle.com/javase/6/docs/technotes/guides/jvmti/

3. Rethinking the debugger

http://scalacamp.pl/data/async-debugger-slides/index.html#/29

4. DEBUGGING IN AN ASYNCHRONOUS WORLD

http://queue.acm.org/detail.cfm?id=945134

5.Recompilation for Debugging Support in a JIT-Compiler

http://www.cs.umd.edu/~hollings/papers/paste02.pdf