AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

Praca dyplomowa magisterska

Debuggowanie aplikacji kominukujących się asynchronicznie oparte o historię komunikatów.

History-based approach for debugging applications using asynchronous communication.

Autor: *Krzysztof Romanowski*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr hab. Arkadiusz Janik, dr inż*

Kraków, 2015

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wprowadzenie 7

1.1. Motywacja 7

1.2. Cele pracy 7

2. Asychroniczny debugger 9

2.1. Techniki i narzędzia służących do debuggowania 9

2.1.1. Traceing 9

2.1.2. Wyjątki i ich analiza 9

2.2. Instrumentacja kodu 9

2.2.1. Debugger 9

2.3. Java Platform Debugger Architecture 9

2.3.1. Architektura 10

2.3.2. JVM TI: instrumentacja JVM 10

2.3.3. JDW: protokół transportowy 10

2.3.4. JDI: wysokopoziomowe API 10

2.3.5. JDI: implementacja w środowisku Eclipse 10

2.4. Zasada dzialania 10

2.4.1. Asychrnoniczna historia wywołań 10

2.4.2. Historia komunikatów 10

2.4.3. Budowanie historii komunikatów 10

2.4.4. Asychroniczne debuggowanie aplikacji aktorowych: Akka 10

3. Aplikacje testowe 11

3.1. Opis technologii 11

3.1.1. Model aktora 11

3.1.2. Framework Akka 11

3.1.3. Integracja z Asychronicznym debuggerem 11

3.2. Aplikacja 1 11

3.2.1. Motywacja 11

3.2.2. Opis diałania ... 11
3.2.3. Opis debuggowania ... 11
3.3. Aplikacja 2 ... 11
3.3.1. Motywacja ... 11
3.3.2. Opis diałania ... 12
3.3.3. Opis debuggowania ... 12
4. Sposoby tworzenie historii komunikatów ... 13
4.1. Dwa etapy: zbieranie i wysyłanie danych ... 13
4.2. Metody zbierania danych ... 13
4.2.1. Plain JDI ... 13
4.2.2. JVM TI: Java Agent ... 13
4.2.3. JDI: instrumentacja kodu ... 13
4.3. Metody przesyłania danych ... 13
4.3.1. Plain JDI ... 13
4.3.2. Plain JDI: lazy mode ... 13
4.3.3. Filesystem ... 13
4.3.4. Socets ... 13
4.4. Testowane złożenia ... 13
5. Wyniki oraz ich a analiza ... 15
5.1. Aplickacja 1 ... 15
5.1.1. Test 1 ... 15
5.1.2. Test 2 ... 15
5.1.3. Test 3 ... 15
5.1.4. Analiza ... 15
5.2. Aplickacja 2 ... 15
5.2.1. Test 1 ... 15
5.2.2. Test 2 ... 15
5.2.3. Test 3 ... 15
5.2.4. Analiza ... 15
5.3. Zestawienie zbiorcze ... 15
5.3.1. Test 1 ... 15
5.3.2. Test 2 ... 15
5.3.3. Test 3 ... 15
5.3.4. Analiza ... 15

6. Analiza oraz wnioski 17

6.1. Dalsze możlowości rozwoju 17

1. Wprowadzenie

1.1. Motywacja

Podczas tworzenia aplikacji asychronicznych twórcy wielokrotnie napotykają ograniczenia narzędzi które nie są przystosowane do pracy z tą klasą aplikacji. Podstawowe techniki takie jak analiza wyjątków czy klasyczne debuggery przeważnie nie daje nam wystarczających informacji o naturze problemów. Iulian Dragos w swojej prezentacji [**?**] przedstawił koncepje asychronicznego debuggera przeznaczonego do debugowania aplikacji stworzych przy wykorzystaniu technologi Akka oraz mechanizmu Feature'ów z języka Scala. Po jej wysłuchaniu uznałem że przedstawiona koncepcja jest dobra, jednakże wykorzystane sposoby persystencji komunikatów będą miały zbyt duży wpływ na działania debuggowanej aplikacji.

1.2. Cele pracy

Celem poniższej pracy jest zbadanie możliwości oraz efektywaności debuggowania aplikacji komunikujących się asychronicznie w oparciu o historię komunikatów. Głównym obszarem zainteresowań pracy będzie narzut sposobu persystowania i analizy komunikatów na czas wykonywania poszczególnych części aplikacji. Zamierzam zaimplementować, przetestować różne podejścia oraz zestawić wyniki wraz z ograniczeniami danej metody. Jako że przedmiotem tej pracy nie jest stworzenie asychronicznego debuggera zamierzam wykorzystać pracę Iuliana [**?**]. Zaimpelemtowany debugger jest częścią ScalaIDE - IDE dedykowanego Scali. Zamierzam testować wydajność wykorzystując aplikacje napisane w frameworku Akka - najpopularniejszej technologi do pisania aplikacji asychronicznych opartych o wymiane komunikatów w ekosystemie Scali.

2. Asychroniczny debugger

W tym rozdziale zamierzam przedstawić zasade działania wykorzystanego asychronicznego debuggera. Zamierzam nakreślić problemy oraz sposoby ich rozwiązywania oraz pokazać dlaczego sposób perysytencji komunikatów jest kluczowy dla minimalizacji wpływu debuggera na debuggowaną .

2.1. Techniki i narzędzia służących do debuggowania

W tym podroziale przedstawie po krótce techniki oraz narzędzia służące analizie oraz debuggowaniu aplikacji które udostępnia ekosystem JVM.

2.1.1. Traceing

2.1.2. Wyjątki i ich analiza

2.2. Instrumentacja kodu

2.2.1. Debugger

2.3. Java Platform Debugger Architecture

W tym podrozdziale zamierzam przedstawić Java Platform Debugger Architecture która jest podstawą dla każdego debuggera dla JVM.

2.3.1. Architektura

2.3.2. JVM TI: instrumentacja JVM

2.3.3. JDW: protokół transportowy

2.3.4. JDI: wysokopoziomowe API

2.3.5. JDI: implementacja w środowisku Eclipse

2.4. Zasada dzialania

W tym podrozdziale zamierzam przedstawić zasadedziałania asychronicznego debuggera oraz nakreślić miejsca które będą przedmiotem tej pracy.

2.4.1. Asychrnoniczna historia wywołań

TODO

2.4.2. Historia komunikatów

TODO

2.4.3. Budowanie historii komunikatów

TODO

2.4.4. Asychroniczne debuggowanie aplikacji aktorowych: Akka

3. Aplikacje testowe

W tym rozdiale zamierzam przedstawić aplikacje które posłużą do testowania debuggera. Opiszę technologię oraz algorytmy w nich zastosowane.

3.1. Opis technologii

3.1.1. Model aktora

3.1.2. Framework Akka

3.1.3. Integracja z Asychronicznym debuggerem

3.2. Aplikacja 1

3.2.1. Motywacja

Co chcemy zbadać, po co itp.

3.2.2. Opis diałania

Opis algorytmu, architektury itp.

3.2.3. Opis debuggowania

W jaki sposób aplikacja bedzie debuggowana, opis breakpointów, testów wydajności oraz walidacji wyników.

3.3. Aplikacja 2

3.3.1. Motywacja

Co chcemy zbadać, po co itp.

3.3.2. Opis diałania

Opis algorytmu, architektury itp.

3.3.3. Opis debuggowania

W jaki sposób aplikacja bedzie debuggowana, opis breakpointów, testów wydajności oraz walidacji wyników.

4. Sposoby tworzenie historii komunikatów

W tym rozdiale zamierzem przedstawić sposoby tworzenia historii komunikatów. Zamierzam podzielić ten proces na dwa etapy i przedstawić sposoby ich implementacji. W ostatnim podrozdiale zamierzam przedstawić testowane sposoby (złożenia).

4.1. Dwa etapy: zbieranie i wysyłanie danych

4.2. Metody zbierania danych

4.2.1. Plain JDI

4.2.2. JVM TI: Java Agent

4.2.3. JDI: instrumentacja kodu

4.3. Metody przesyłania danych

4.3.1. Plain JDI

4.3.2. Plain JDI: lazy mode

4.3.3. Filesystem

4.3.4. Socets

4.4. Testowane złożenia

TODO: wyjdzie podczas implementacji

5. Wyniki oraz ich a analiza

5.1. Aplickacja 1

5.1.1. Test 1

5.1.2. Test 2

5.1.3. Test 3

5.1.4. Analiza

5.2. Aplickacja 2

5.2.1. Test 1

5.2.2. Test 2

5.2.3. Test 3

5.2.4. Analiza

5.3. Zestawienie zbiorcze

5.3.1. Test 1

5.3.2. Test 2

5.3.3. Test 3

5.3.4. Analiza

6. Analiza oraz wnioski

TODO: w zależności co wyjdzie

6.1. Dalsze możlowości rozwoju