CARL VON OSSIETZKY universität OLDENBURG

Trace- und Zeit-Zusicherungen beim Programmieren mit Vertrag

Mark Brörkens

Universität Oldenburg, Fachbereich Informatik

Email: Mark.Broerkens@informatik.uni-oldenburg.de

Motivation

Idee für diese Diplomarbeit

- Aufgabe aus dem Programmieralltag: Importierung von Benutzerdaten in ein neues System
- Optimierung durch viele parallele Zugriffe
- Die Reihenfolge einzelner Import-Aktionen ist kritisch
- Bei zu vielen parallelen Zugriffen dauert jede einzelne Aktion zu lange. Es kommt zu Timeouts.

CARL VON OSSIETZKY universität OLDENBURG

Einleitung

Voraussetzungen für das Prüfen von Trace- und Zeit-Zusicherungen zur Laufzeit:

- Ermittlung einer Trace

CARL VON OSSIETZKY universität OLDENBURG

Einleitung

Voraussetzungen für das Prüfen von Trace- und Zeit-Zusicherungen zur Laufzeit:

- Ermittlung einer Trace
- Prüfen der ermittelten Trace

CARL VON OSSIETZKY universität OLDENBURG

Programmieren mit Vertrag

- Eingeführt von Bertrand Meyer als Bestandteil der Programmiersprache Eiffel
- Vor- und Nachbedingungen sowie Invarianten
- Konzept der Vererbung
- Erweiterung: Trace Assertions

Erweiterungen von Java

- Direkte Erweiterung der Programmiersprache um weitere Schlüsselworte
  (JDK 1.4, AspectJ)

CARL VON OSSIETZKY universität OLDENBURG

Erweiterungen von Java

- Direkte Erweiterung der Programmiersprache um weitere Schlüsselworte
  (JDK 1.4, AspectJ)
- Quellcode-Instrumentierung
  (iContract, Jcontract, JML, Jass)

Erweiterungen von Java

- Direkte Erweiterung der Programmiersprache um weitere Schlüsselworte
  (JDK 1.4, AspectJ)
- Quellcode-Instrumentierung
  (iContract, Jcontract, JML, Jass)
- Bytecode-Instrumentierung
  (jContractor, Handshake, Java-MaC)

CARL VON OSSIETZKY universität OLDENBURG

Erweiterungen von Java

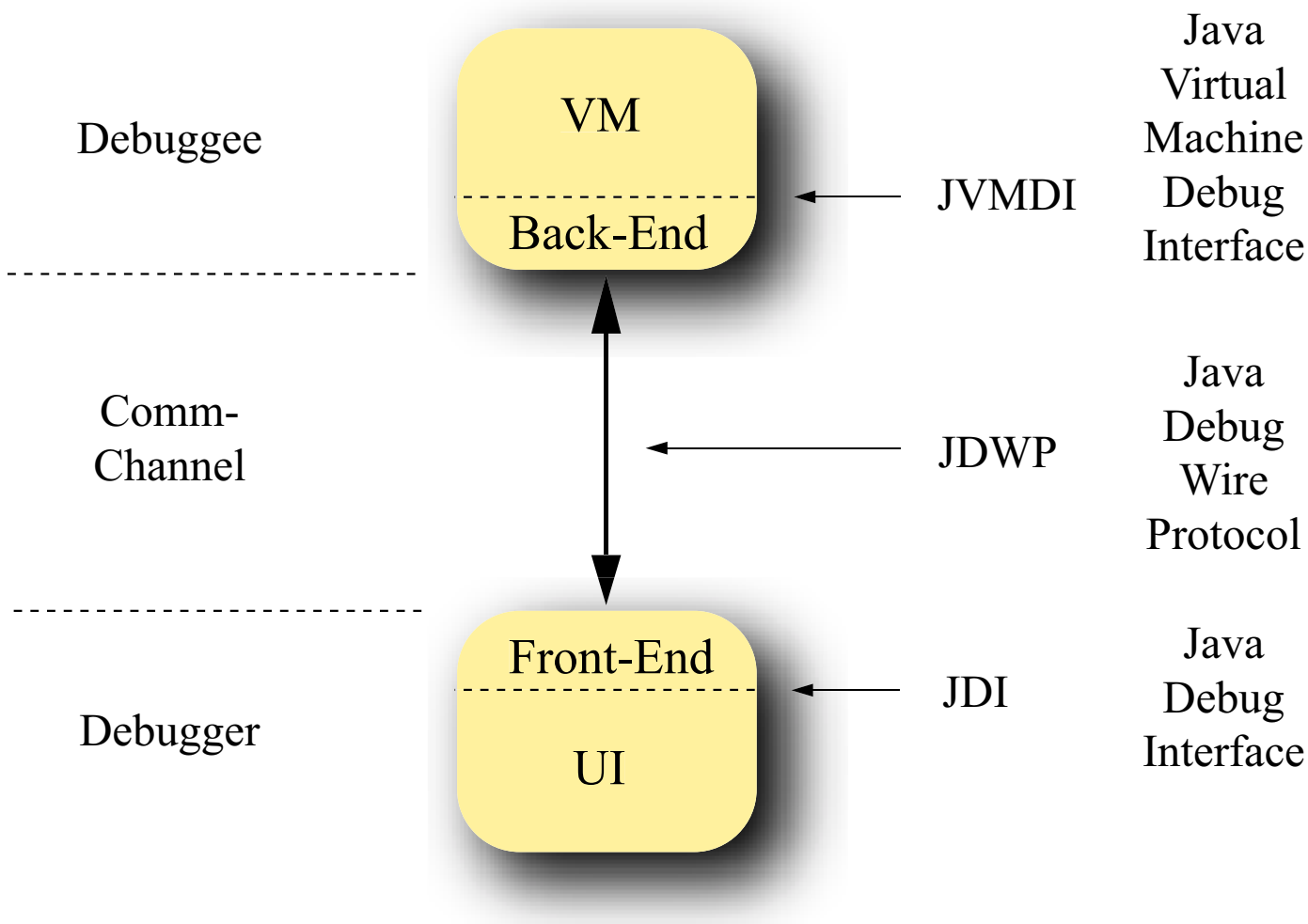
- Direkte Erweiterung der Programmiersprache um weitere Schlüsselworte
  (JDK 1.4, AspectJ)
- Quellcode-Instrumentierung
  (iContract, Jcontract, JML, Jass)
- Bytecode-Instrumentierung
  (jContractor, Handshake, Java-MaC)
- Einsatz von Debuggern
  (JMSAssert, Lucent Technologies, Query Debugger)

CARL VON OSSIETZKY universität OLDENBURG

Ermitteln einer Trace

- **J**ava **P**latform **D**ebugger **A**rchitecture
- Wird von vielen Java Virtual Machines unterstützt
- Bietet eine Java API für den Zugriff auf Java-Programme zur Laufzeit

JPDA


Debuggee
VM
Back-End
JVMDI
Java Virtual Machine Debug Interface
Comm-Channel
JDWP
Java Debug Wire Protocol
Front-End
UI
Debugger
JDI
Java Debug Interface

CARL VON OSSIETZKY universität OLDENBURG

Einschränkungen des JDI

- Rückgabe-Werte von Methoden können nicht ausgelesen werden.

CARL VON OSSIETZKY universität OLDENBURG

Einschränkungen des JDI

- Rückgabe-Werte von Methoden können nicht ausgelesen werden.
- Klassen müssen beim Übersetzen mit Debug-Optionen versehen werden.

CARL VON OSSIETZKY universität OLDENBURG

Prüfen einer Trace

- Beschreibung der Trace- und Zeit-Zusicherungen durch den CSP-Dialekt CSP_{jassda} .
- Die Trace-Semantik definiert die Menge erlaubter Traces.
- Die operationellen Semantik beschreibt, wie ein Process schrittweise verarbeitet werden kann.

CSP_{jassda}

BNF-Syntax von CSP_{jassda} :

```
P ::= STOP
    | TERM
    | ANY[A]
    | a -> P
    | P ; Q
    | P [] Q
    | P || Q
    | X
```

CARL VON OSSIETZKY universität OLDENBURG

Ereignis

$$event = \begin{pmatrix} type \\ jdi - interface \\ clocks - interface \end{pmatrix}$$

CARL VON OSSIETZKY universität OLDENBURG

Ereignis

$$event = \begin{pmatrix} type \\ jdi - interface \\ clocks - interface \\ virtualmachine \\ thread \\ instance \\ method \\ parameter \\ returnvalue \end{pmatrix}$$

CARL VON OSSIETZKY universität OLDENBURG

Unterschiede zu CSP

- Abstraktion von konkreten Ereignissen durch Ereignismengen

CARL VON OSSIETZKY universität OLDENBURG

Unterschiede zu CSP

- Abstraktion von konkreten Ereignissen durch Ereignismengen
- Nur sichtbare Ereignisse werden berücksichtigt

CARL VON OSSIETZKY universität OLDENBURG

Unterschiede zu CSP

- Abstraktion von konkreten Ereignissen durch Ereignismengen
- Nur sichtbare Ereignisse werden berücksichtigt
- Quantifizierende Operatoren lassen Aussagen über eine Menge von Threads und Instanzen zu.

CARL VON OSSIETZKY universität OLDENBURG

Beispiel

```
trace trace1 {
   eventset helloThread
   {class="jass.debugger.examples.HelloThread"}
   eventset run           {method="run"}
   eventset getHello      {method="getHello"}

   hellothread() {
      ||x:[thread]@
      x.helloThread.run.begin ->
      x.helloThread.getHello.begin ->
      x.helloThread.run.end ->
      TERM
   }
}
```

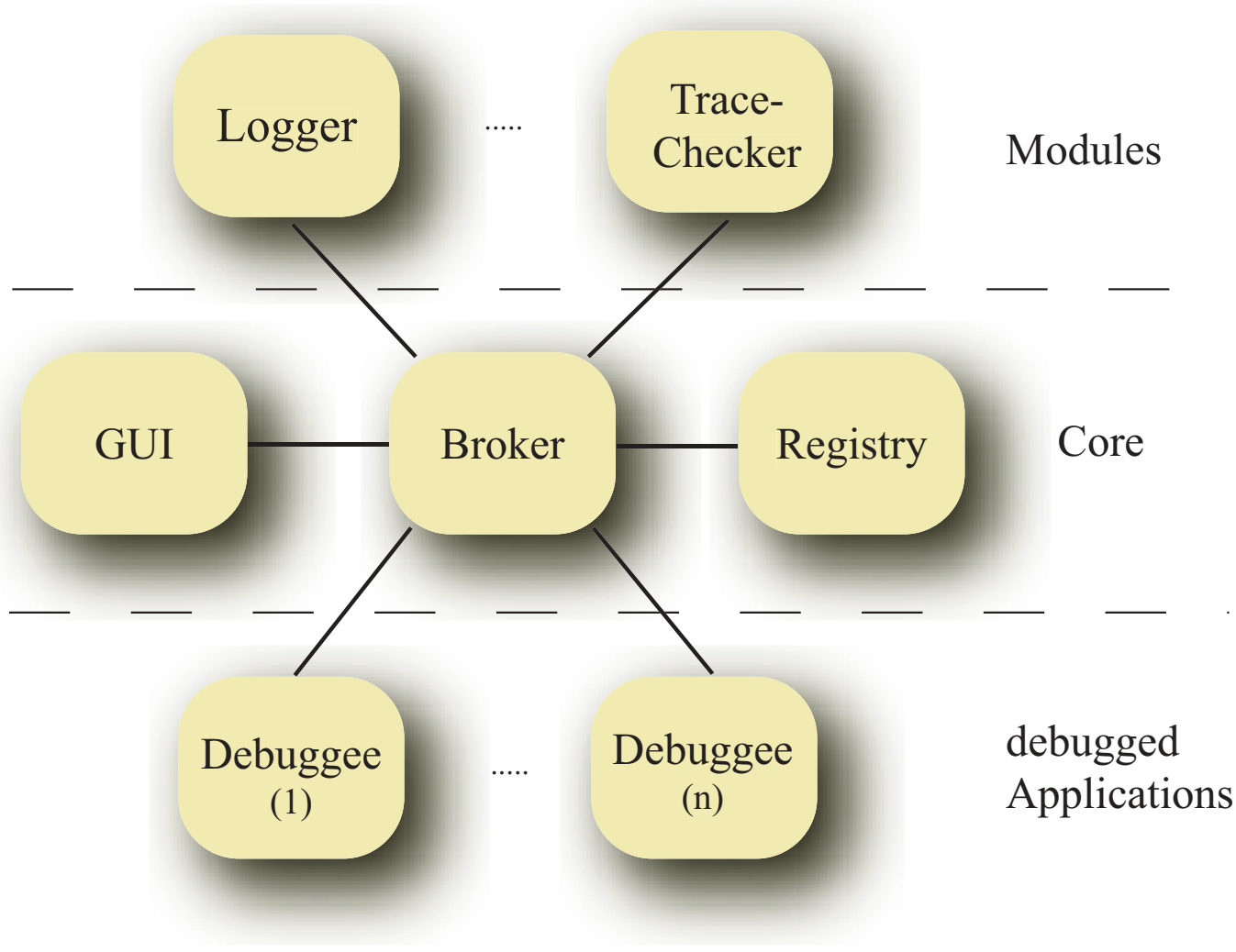
CARL VON OSSIETZKY universität OLDENBURG

Jassda

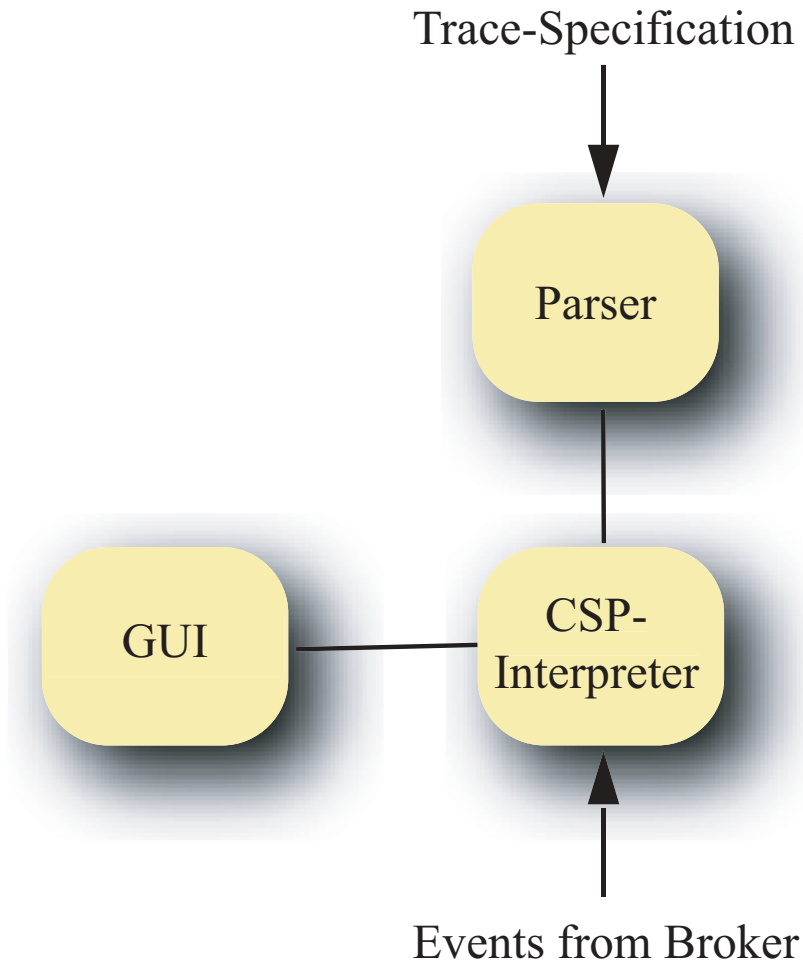
- Jassda is ein Acronym für **J**ava with **Ass**ertions **D**ebugger **A**rchitecture.
- Die Trace eines Java-Programmes wird zur Laufzeit unter Verwendung des JDI ermittelt.
- Sie kann in eine Datei protokolliert
- oder gegen eine CSP_{jassda} -Spezifikation geprüft werden.

CARL VON OSSIETZKY universität OLDENBURG

Architektur



Trace-Prüfer



Benchmark

- Bubblesort-Algorithmus
- zwei Implementierungen für unterschiedliche Anzahl von Methodenaufrufen.
  - Implementierung 1 benötigt für die Sortierung von 10000 Zahlen einen Methodenaufruf.
  - Die zweite Implementierung benötigt 10001 Methodenaufrufe.

CARL VON OSSIETZKY universität OLDENBURG

Bubblesort (1)

```
void sort1(int[] field) {
  for(int element = 0; element < field.length; element++) {
    for(int run = field.length; --run > element; ) {
      if(field[run-1] > field[run]) {
        int tmp = field[run-1];
        field[run-1] = field[run];
        field[run] = tmp;
      }
    }
  }
}
```

Bubblesort (2)

```
void sort2(int[] field) {
  for(int element = 0; element < field.length; element++) {
    exchange(field,element);
  }
}
void exchange(int[] field, int element) {
  for(int run = field.length; --run > element; ) {
    if(field[run-1] > field[run]) {
      int tmp = field[run-1];
      field[run-1] = field[run];
      field[run] = tmp;
    }
  }
}
```

Benchmark (Ergebnisse)

Beschreibung	hotspot		classic	
	1	2	1	2
optimierter Code ohne Debug-Informationen	1,302*s*	1,262*s*	11,897*s*	12,067*s*
optimierter Code mit Debug-Informationen	1,332*s*	1,252*s*	11,897*s*	12,067*s*
nicht optimierter Code ohne Debug-Informationen	1,302*s*	1,252*s*	11,907*s*	12,067*s*
nicht optimierter Code mit Debug-Informationen	1,342*s*	1,252*s*	11,918*s*	12,057*s*
VM im Debug-Modus	13,629*s*	13,630*s*	50,983*s*	51,013*s*
VM im Profiling-Modus	1,442*s*	1,372*s*	12,458*s*	12,458*s*
VM mit Jassda verbunden	15,031*s*	2310,393*s*	56,812*s*	629,095*s*

CARL VON OSSIETZKY universität OLDENBURG

Zusammenfassung

- Marktanalyse
- Technische und theoretische Grundlagen
- Entwurf und Implementierung einer Architektur zur Runtime-Validierung von Java-Programmen
- Analyse des zeitlichen Verhaltens

CARL VON OSSIETZKY universität OLDENBURG

Ausblick

- Erweiterung von Jassda um weitere Module
- Ergänzung von CSP_{jassda} um weitere Operatoren
- Einbettung in grafische Benutzungsoberfläche wie NetBeans
- Optimierung der Ausführungsgeschwindigkeit