Use-Case-basierte Abhängigkeitsanalyse von Legacy-Systemen mit Kieker

We make IT better!

b+m Informatik AG

Engineering & Solutions

Holger Knoche

Kieker Days 2012

Kiel, 29.11.2012

b+m business IT management

ALLGEIER GROUP

Dynamische Abhängigkeitsanalyse im Rahmen der Software-Modernisierung

- In vielen Fällen ist nicht mehr bekannt, **welche Systemteile** an der **Abarbeitung einer Funktion** beteiligt sind
- Wichtiges Grundziel: **Wiedergewinnung** von Informationen zu üblichen **Anwendungsfällen**
- Übliches Vorgehen: **Profiling** der Anwendung während der **Ausführung definierter Anwendungsfälle**
- **Prüfung** der gewonnenen Erkenntnisse über Erhebungen aus dem **produktiven Betrieb**

Zusätzlicher Nutzen:

- **Ergänzung** bzw. **Schärfung statischer Analyseergebnisse** (z.B. bei dynamischem Dispatching)
- Anhaltspunkte für produktiv **nicht mehr genutzte Features**

Kurzportrait Fallstudie 1: Analyse einer COBOL-Anwendung

Ausgangssituation

- Kernsoftwarekomponente in COBOL (Unix und Großrechner)
- **Strukturerosion** über mehr als 20 Jahre und mehrere Entwicklergenerationen
- **Aufnahme** von **fachfremder Funktionalität**

Modernisierungsziele

- **Auftrennung** in fachlich motivierte Komponenten
- **Herauslösen** von fachfremder Funktionalität

Rolle der dynamischen Analyse

- Erhebung der an bestimmten **Anwendungsfällen** beteiligten **Komponenten**
- Untersuchung der **Interaktion mit Umsystemen**

Vorbereitung der dynamischen Analyse

Herausforderung: Es gibt **kein AOP / Bytecode Weaving** für COBOL.

Lösung: Automatisierte **Instrumentierung des Quellcodes**

```
INITIALIZE LOG-RECORD
MOVE 'MODULE-B' TO LOG-MODULE-NAME
MOVE SPACES TO LOG-SECTION-NAME
MOVE 'CALL' TO LOG-EVENT
MOVE "MODULE-C"
     TO LOG-EVENT-DATA
MOVE 1 TO LOG-INSTR-REVISION

CALL LOG-PROGRAM USING LOG-RECORD
CALL "MODULE-C".
```

Herausforderung: Verknüpfen der **Anwendungsfälle** mit den zugehörigen **Traces**

Lösung:

- Aufnahme zusätzlicher **Kontextinformationen** (insb. Benutzerkennung) in die Logdaten
- Werkzeuggestützte Aufnahme von **Metadaten der Testdurchführung** (Zeitintervall, Benutzerkennung)

Kurzcharakterisierung der durchgeführten Analyse

- **Vollinstrumentierung** von ca. 1.100 COBOL-Modulen **auf Section-Ebene**
- Anzahl Messpunkte im Quellcode: ca. 140.000
- Durchgeführte Anwendungsfälle: 60
- Events pro Anwendungsfall: i.d.R. zwischen 3.000 und 8.000
- Gesamtdatenvolumen (inkl. Fehlversuche): ca. 1,8 Mio Events

Umgang mit lückenhaften Traces

Herausforderung: Quellcodeinstrumentierung erlaubt nur eine **partielle Abdeckung** der Anwendung.

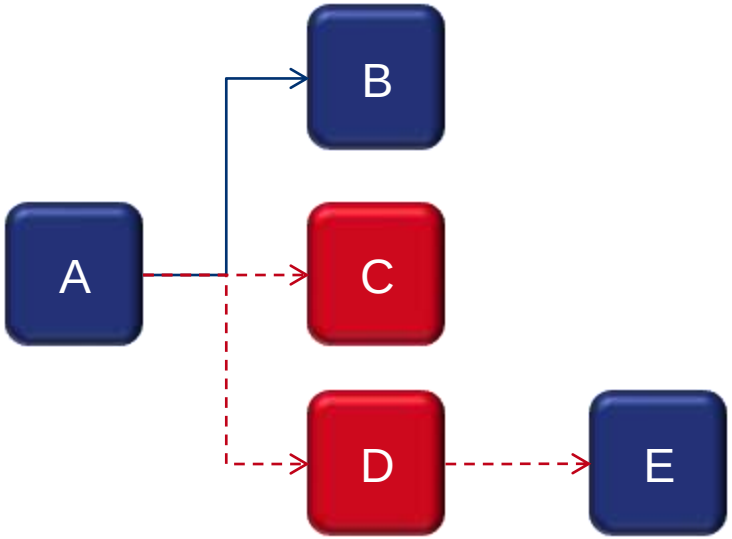
Lösung: Nutzung eines **ereignisbasierten** Aufruf-Modells

@A: CALL B
@B: ENTRY
@B: EXIT

@A: CALL C
@A: …

@A: CALL D
@E: ENTRY

Events

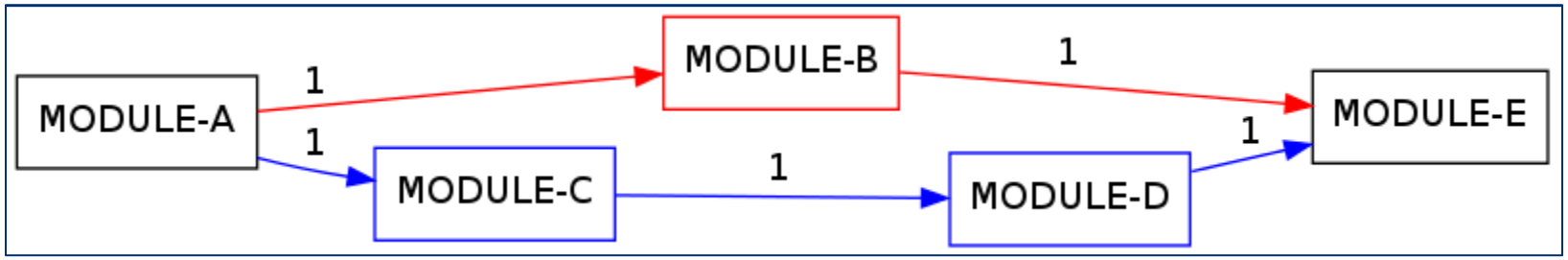


Aufrufmodell

Hervorhebung von Traces

Motivation: Bei ähnlichen Anwendungsfällen sind häufig insbesondere die **Differenzen** interessant.

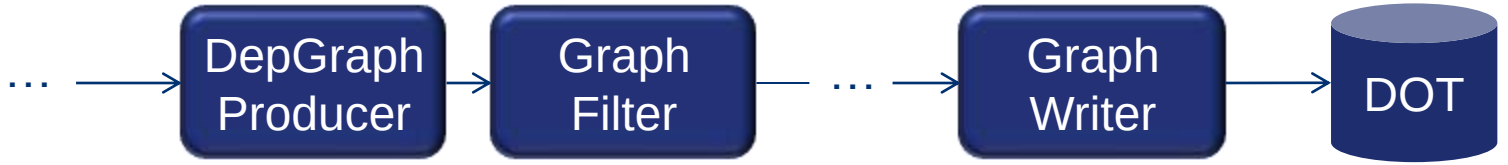
Lösung: Unterschiedliche **Einfärbung** der eindeutig zugeordneten und der gemeinsamen Elemente



Exkurs: Graph-Filter in Kieker

Bis Kieker 1.5

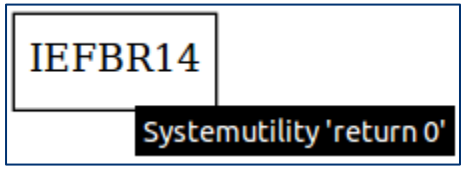
Seit Kieker 1.6
(nur für Dependency-Graphen)



Weitere neue Graph-Features

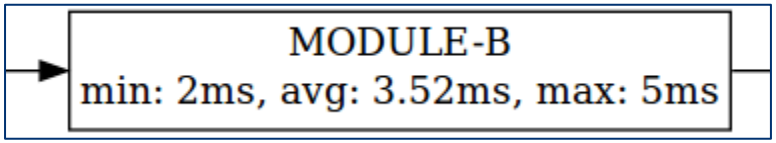
Tooltip-Beschreibungen an Dependency-Graphen

Motivation: Programme im Großrechner-Umfeld tragen oft **kryptische Namen** (z.B. IEFBR14)



Dekoratoren an Graph-Knoten

Motivation: Einige Informationen (z.B. Antwortzeiten) sollen **unmittelbar an den betreffenden Entitäten sichtbar** sein



Kurzportrait Fallstudie II: Analyse einer Visual Basic 6-Anwendung

Ausgangssituation

- Client-Server-Anwendung in Visual Basic 6
- Seit ca. 10 Jahren durchgehend vom selben Entwicklerteam betreut und gewartet

Modernisierungsziel

- **Reimplementierung** des bestehenden Systems auf einer neuen Plattform

Rolle der dynamischen Analyse

- Analyse potentiell **nicht mehr genutzter Features**
- Betrachtung der **tatsächlichen Datenbank-Nutzung**
- Untersuchung von **produktiv genutzten Anwendungsfällen**

Kurzcharakterisierung der durchgeführten Analysen

- **Vollinstrumentierung** von 33 Visual Basic 6-Modulen **auf Methodenebene inkl. Datenbankzugriffe**
- Anzahl Messpunkte im Quellcode: ca. 1.000

Analyse I: Gezielte Bedienung durch Entwickler

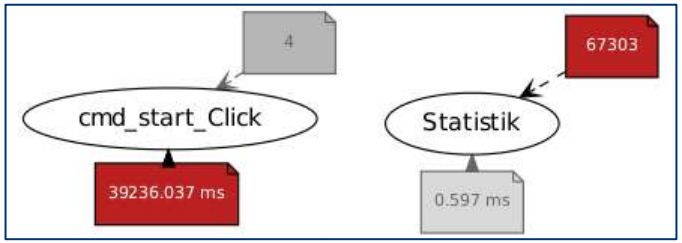
- Durchgeführte Anwendungsfälle: 19
- Gesamtdatenvolumen: 7.345 Execution Records

Analyse II: Monitoring in Produktion

- Produktiver Einsatz einer instrumentierten Version über 6 Monate auf 2 Arbeitsplätzen
- Gesamtdatenvolumen: ca. 1,3 Mio Execution Records

Dynamische Analyse mit gezielter Bedienung: Ausgewählte Ergebnisse

Befund: Auffällig **hohe Antwortzeit** (ca. 40 Sekunden) bei der Ausführung einer Statistikfunktion



Ursache: Ausimplementiertes GROUP BY innerhalb der Applikation

Befund: 9 Masken wurden während der Anwendungsfälle **nicht genutzt**

Ursache: 6 Masken wurden nicht abgedeckt, 3 waren tatsächlich **nicht mehr in Gebrauch**

Rekonstruktion von Maskenflussmodellen

Nutzen von Maskenflussmodellen

- Maskenflüsse bzw. UI-Verhalten bieten eine **benutzerorientierte Sicht** auf die analysierte Anwendung
- Bedienungsabläufe im UI lassen sich einfach **auf Anwendungsfälle abbilden**
- UI-Abläufe **abstrahieren** von irrelevanten technischen Details

Vorgehensweise

- **Gezielte Bedienung** einer instrumentierten Anwendung
- **Filterung** oberflächenrelevanter Events aus den Traces
- **Konstruktion eines Modells** aus den gefilterten Events

Session-Rekonstruktion mit Kieker

Herausforderung: Traces sind teilweise **zu kurz**, um Anwendungsfälle vollständig zu enthalten.

Lösung: Rekonstruktion von **Sessions**

Trennung anhand einer Session-ID

Trennung anhand einer maximalen „Denkzeit“

Kieker-Interface zur Einbettung in Analysewerkzeuge

Herausforderung: Kieker ist (bisher) als **eingebettete Analyseengine unhandlich**.

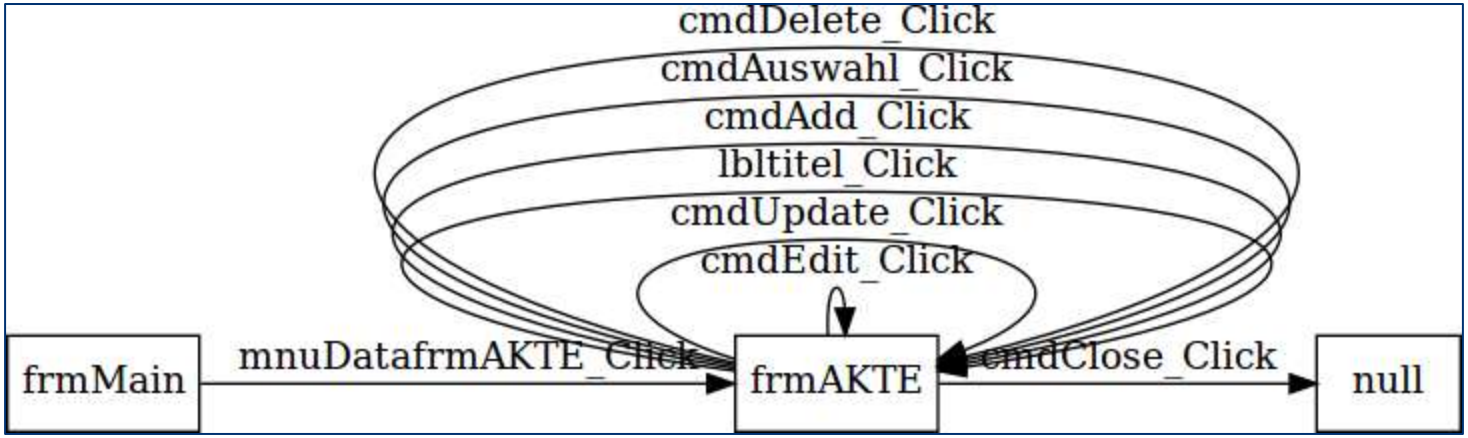
- AnalysisController instanziieren
- SystemModelRepository einrichten
- FSReader einrichten
- ExecutionTransFilter einrichten
- TraceReconstructionFilter einr.
- SessionReconstructionFilter einr.
- ListCollectionFilter einrichten

Bisher

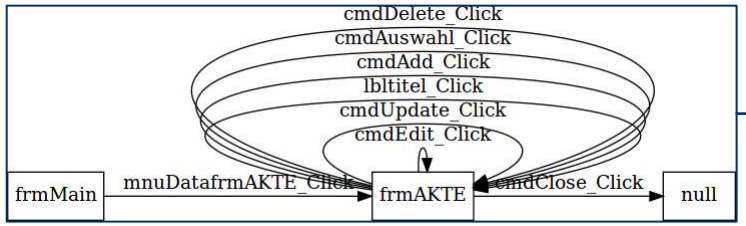
```
RunnableKiekerSetup<ExecutionTraceBasedSession> sessionData =
        KiekerEmbeddedInterface.readExecutionBasedSessionsFromFilesystem(
                MAX_TRACE_DURATION_MILLIS,
                MAX_THINK_TIME_MILLIS,
                inputPath);
Iterable<ExecutionTraceBasedSession> sessions = sessionData.getData();
```

Neuerdings

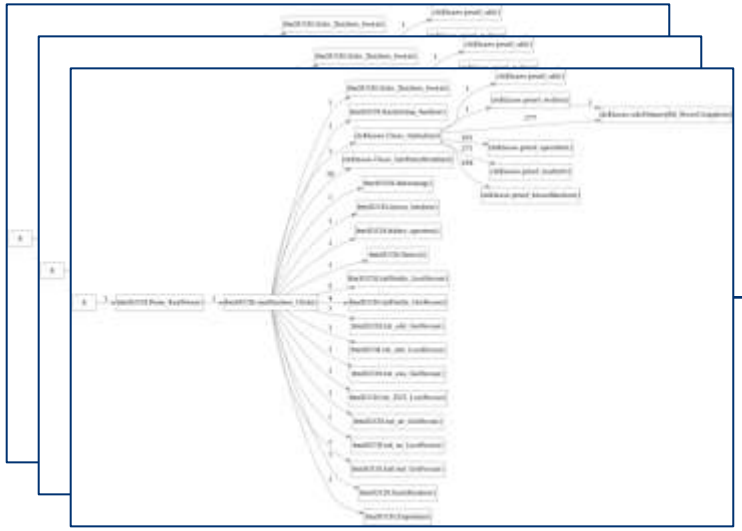
Beispiel eines extrahierten Maskenflussmodells



Lokation von Maskenflüssen in Traces

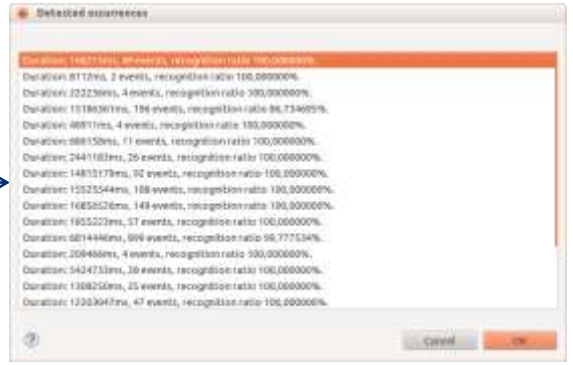


Zu suchender Flow



Erhobene Traces

Flow-Detektor



Gefundene Auftreten

Zusammenfassung

- Dynamische Abhängigkeitsanalyse von Legacy-Systemen ist **aufwendig, aber lohnenswert**
- Durch gezieltes Bedienen der Anwendung lassen sich wertvolle Informationen über den **Zusammenhang von Code und Anwendungsfällen** gewinnen
- Es gibt **neue Features** in Kieker:
  - Ereignisbasiertes Aufrufmodell
  - Graph-Filter (Trace-Färbung, Beschreibungen)
  - Knoten-Dekoratoren
  - Session-Rekonstruktion
  - Komfortables Interface zur eingebetteten Verwendung

Ansprechpartner

Holger Knoche
Senior-Softwarearchitekt
Engineering & Solutions

holger.knoche@bmiag.de
F +49 (0)4340 404 1620

b+m Informatik AG
Rotenhofer Weg 20
24109 Melsdorf
www.bmiag.de