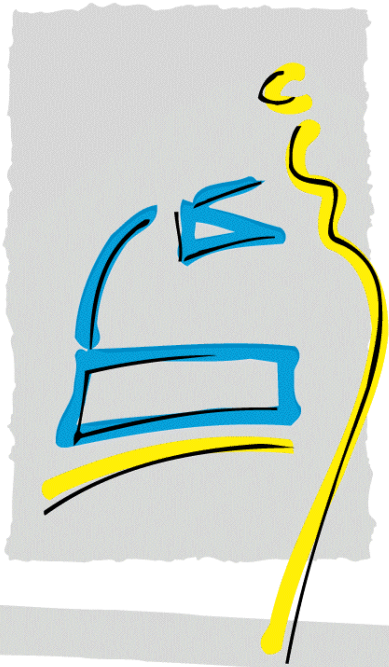
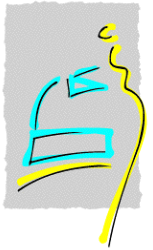
Programmstrukturen

Werkzeugunterstützte Softwareadaption mit Inject/J

Volker Kuttruff
Thomas Genßler





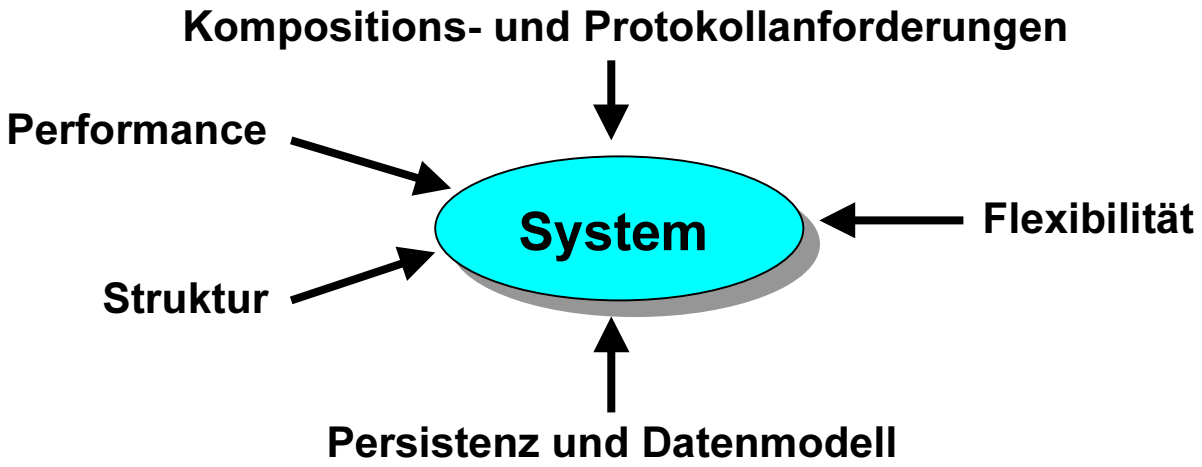
Übersicht

- **Einführung**
- **Softwareadaption mit Inject/J**
  - **Anforderungen**
  - **Überblick Inject/J**
  - **Konzepte**
  - **Korrektheit**
  - **Beispiele**
- **Zusammenfassung und Ausblick**

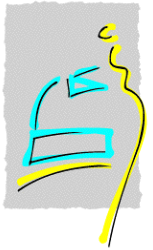
Einführung

Kontext

- **Anforderungen an ein System ändern sich in den verschiedenen Entwicklungsstufen**



- **Änderung der Anforderungen erfordert Adaption**

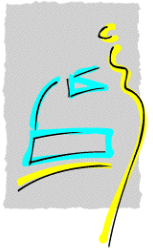


Einführung

Adaptionsprobleme

- **Adaptionen oftmals querschneidend**
- **Implementierung der Anforderungen im System verteilt**
  - **Konsistente Adaptionen aufwendig**
  - **Oftmals interne Implementierungsstrukturen betroffen**
- **Manuelle Adaption erfordert Verständnis des Quellcodes**

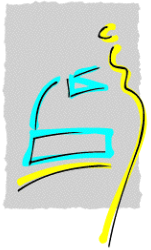
➢ **Werkzeugunterstützte Softwareadaption**



Softwareadaption mit Inject/J

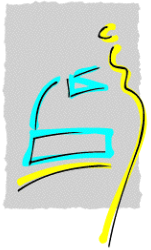
Anforderungen

- **Invasive Softwareadaption (Greybox-Adaption)**
- **Einfache Spezifizierung häufiger Trans-formationen auf geeignetem Abstraktionsniveau**
- **Verarbeitung bereits vorhandenen Quellcodes, insbesondere kein Paradigmenwechsel**
- **Wiederverwendbarkeit der Adaptionen**
- **Korrektheitsbegriff**



Überblick Inject/J

- **Werkzeug für automatisierte Codetransformation**
  - **Einfaches Metamodell**
  - **Einfache Skriptsprache inkl. Skriptprozessor**
  - **Arbeitet direkt auf Java-Quellcode**
- **Einsatzmöglichkeiten**
  - **Ad-hoc Transformationen**
  - **Querschneidende Adaptionen**
  - **Entwurfsmusteroperatoren**
  - **Gluecodegenerierung (Konnektoren)**
  - **Allgemeine Codegenerierung**



Zentrale Konzepte

Namensraum

Enthält alle für die Adaption benötigten Java-Klassen

Webepunkte

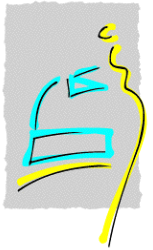
Bilden Metamodell von Inject/J

Navigation

Auffinden der gesuchten Webepunkte

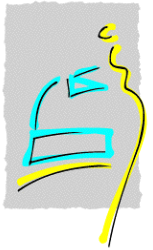
Transformation

sog. Webeoperationen ändern den Quelltext invasiv



Webepunkte

- Mögliche Transformationsstellen im System
- Identifizierung durch Name bzw. Signatur
- Implizite Webepunkte
  - Klasse
  - Methode
  - Attribut
  - Methoden-/Attributzugriff
  - Zuweisungen
- Explizite Webepunkte
  - Explizite Markierung potentieller Adaptionsstellen

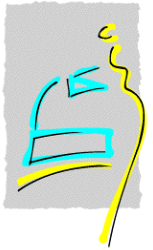


Navigation

- **Hierarchische Navigation**
- **Direkte Navigation**
- **Spezifikation der Navigationsziele mit Hilfe regulärer Ausdrücke und Quantoren**

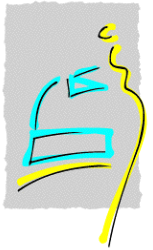
Beispiele:

```
in class ´fzi.injectj.Main´ do ...

foreach method ´put(java.lang.Object,*) do ...
```



Webeoperationen

- **before / after**
  - **Einfügen von Code vor bzw. nach dem Webepunkt**
- **add**
  - **Einfügen von Code in listenartige Strukturen**
- **change modifier**
  - **Ändert die Modifizierer eines Webepunktes**
- **replace**
  - **Ersetzt den momentanen Quelltext durch ein neues Codefragment**
- **delete**



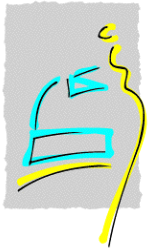
Korrektheit

Ist das System vor der Operation übersetzbar,
so ist es auch nach der Operation übersetzbar.

- **Sichergestellt durch umfangreiche Überprüfungen**
  - **Prüfung auf Existenz von statischen Referenzen auf zu löschende Elemente**
  - **Prüfung von Namenskonflikten**
  - **Inkrementeller Parser**
- **Zusätzlich: Skriptbasierte Überprüfungen durch Webepunkt-Attribute**

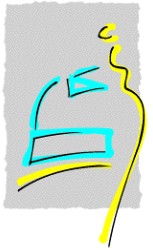
Beispiel (I)

```
foreach class ´*´ do {
 foreach method ´*´ do {
  foreach access ´pop()´ <=a>
                 to subclass ´java.util.Stack´ do {
   before ${

    if (<a.prefixText>.empty())
      System.err.println(„Empty stack...“);

   }$
  }
 }
}
```



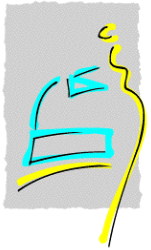
Beispiel (II)

```
if (staticStack.empty())
   System.out.println(„Empty stack...“);
privateStack.push(staticStack.pop());

if (privateStack.empty())
   System.out.println(„Empty stack...“);
Object top = privateStack.pop();

this.pop();
```



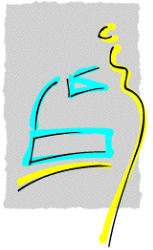
Weitere Beispiele

- **Observifier**
  - **Transformiert direkte Methodenaufrufe in nachrichtenbasierte Notifikation (Observer / Observable)**
  - **ca. 170 Zeilen Skriptcode inkl. graphischer Benutzerinteraktion**
- **Beanifier**
  - **Ersetzt Attributzugriffe durch Get/Set-Methoden**
- **Automatische Visitorgenerierung**



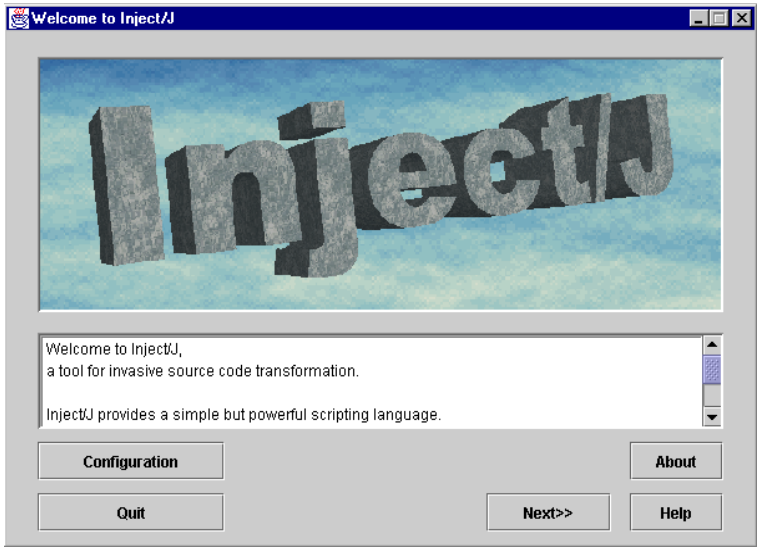
Zusammenfassung

- **Problem:**
  - **Aufwendige Adaptionen in großen Systemen aufgrund sich ändernder Anforderungen**
- **Neues Werkzeug: Inject/J**
  - **Invasiv, nicht auf Blackbox-Adaption beschränkt**
  - **Einfache Skriptsprache**
  - **Kein Paradigmenwechsel, arbeitet mit vorhandenem Quelltext**



Ausblick

- Unterstützung weiterer Webepunkte
- Verfeinerung des Transaktionsbegriffs
- Sprachmittel zur Spezifikation von Vorbedingungen
- Umgang mit unvollständigem Code
- Bibliotheken mit vorgefertigten Operationen
- Bessere Integration in Entwicklungsumgebungen



Inject/J Webseite

http://injectj.fzi.de

Kontakte

Thomas Genßler (genssler@fzi.de)

Volker Kuttruff (kuttruff@fzi.de)