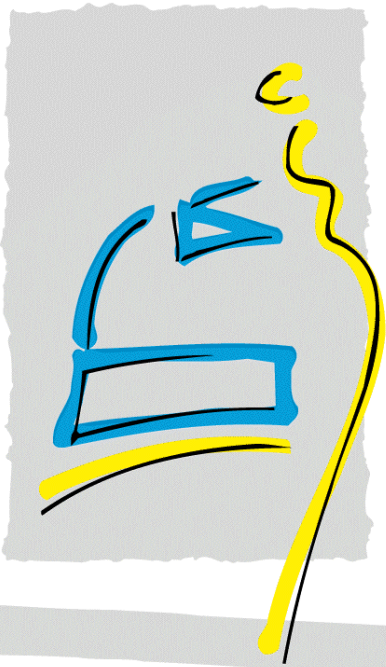
Programmstrukturen

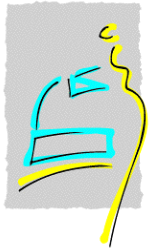
Adaptives Programmieren mit Inject/J

Olaf Seng

Thomas Genßler

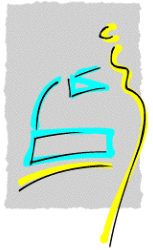
Benedikt Schulz





Übersicht

- **Einführung**
- **Grundlagen**
- **Vorgehensweise**
- **Beispiel**
- **Zusammenfassung**



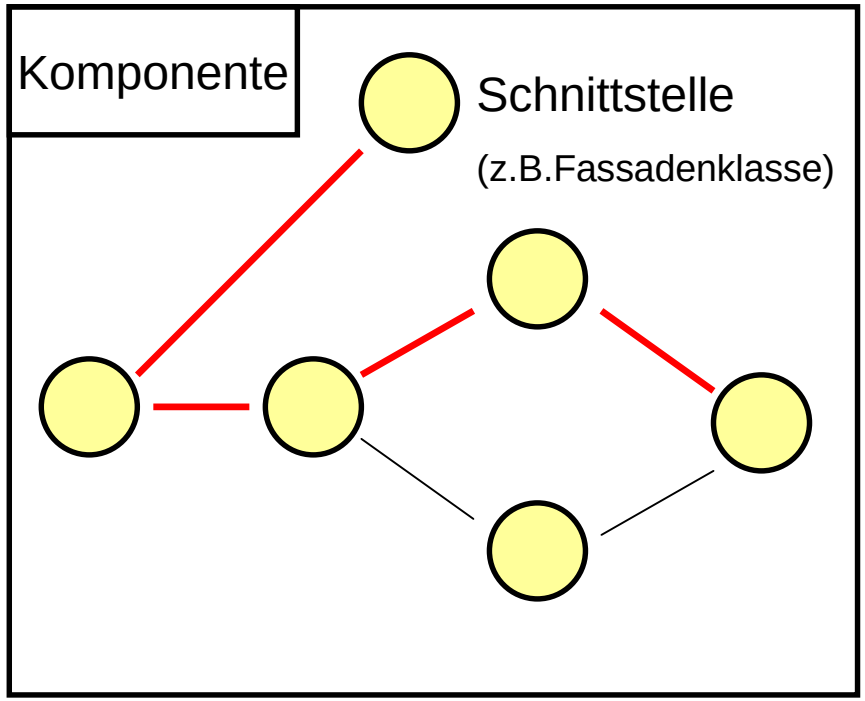
Einführung
Szenario & Probleme

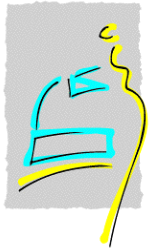
- **Hinzufügen einer neuen Funktion zu einer Komponente**
- **Erfordert Änderungen an allen beteiligten Klassen**
- **Oft nur einfache Durchreichmethoden**

➢ **Detailliertes Verständnis erforderlich (Whitebox)**

➢ **Aufwendig, fehleranfällig**

➢ **Evolution erschwert**

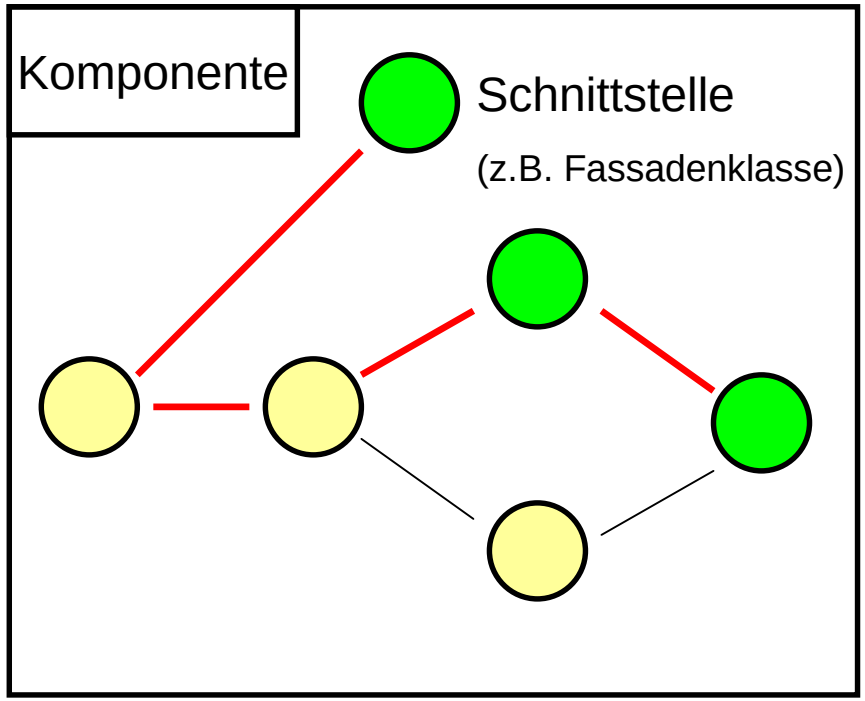




Einführung
Lösungsansatz

- **Nur an wenigen Stellen „passiert“ etwas**
- **Code für wichtige Klassen angeben**
- **„Navigation“ zu diesen Stellen kann berechnet werden**

➢ **Adaptives Programmieren**



Grundlagen

Adaptives Programmieren

- **Spezialfall von Aspektorientiertem Programmieren**
- **Getrennte Spezifikation von Verhalten und Struktur**
  (Verhalten = Schnittstellen & Methodenrümpfe)
  (Struktur = Klassen, Vererbung, Delegation)
- **Adaptive Programme als Pfade zwischen „wichtigen Progammelementen“**
- **Ausführbares Programm aus „class dictionary“ und Navigationsanweisung**

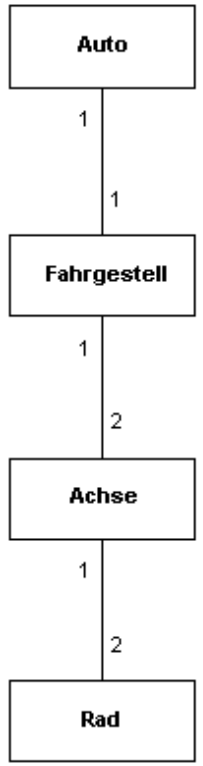
Grundlagen

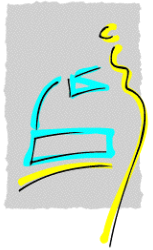
AP-Beispielprogramm

```
operation ‘void druckeReifendruck()’ {

   traverse from ‘Auto’ to ‘Rad’


   at `Rad`  do {
     before ${
        System.out.println(reifenDruck);
     }$
   }
}
```





Grundlagen

Adaptives Programmieren

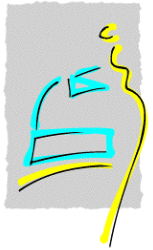
- **Vorteile:**
  - **Getrennte Änderungen von Struktur und Verhalten möglich**
    - **Einfaches Hinzufügen von Funktionalität**
    - **Änderungen an der Klassenstruktur**
  - **Bessere Wartbarkeit, Wiederverwendung**

- **Nachteile bestehender Implementierungen:**
  - **Auf Schreiben neuer Programme ausgerichtet**
  - **Kein Umgang mit bestehenden Softwaresystemen**

Vorgehensweise

Unser Ansatz

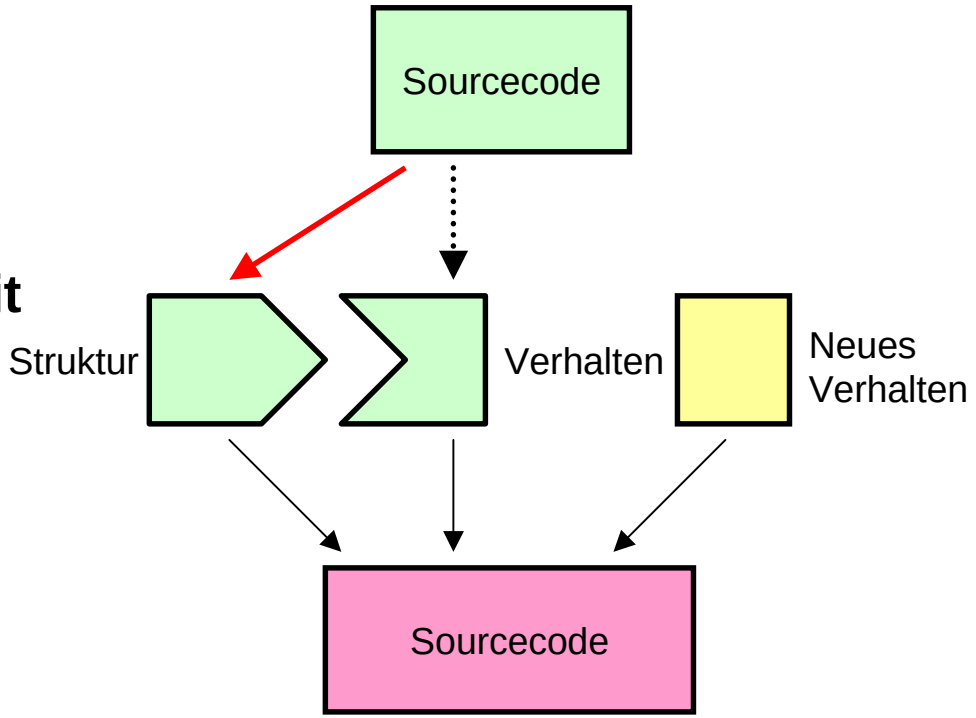
- **AP als Metaprogramm**
- **Kombination von Adaptivem Programmieren mit Skriptsprache für Transformationen (Inject/J)**
- **Neue Sprachmittel für Inject/J**
- **Auswahl aus bestehenden AP Konstrukten**
  - **Möglichst einfach**
  - **volle AP Funktionalität**
- **Ziel: Werkzeug zur einfachen Erweiterung von Programmen**



Vorgehensweise

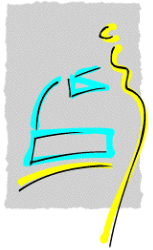
Ablauf

- **Gegeben:**
  - **Sourcecode**
  - **AP – Skript**
  - **Strukturmodell implizit (durch Sourcecode)**
- **Ergebnis:**
  - **Korrekter, erweiterbarer, wartbarer Code**



Vorgehensweise
Transformation

- **Extraktion der Struktur durch Sourcecodeanalyse**
  - **Problem: Vektoren,...**
  - **Lösung: Typinferenz, Benutzerinteraktion**
- **Berechnung der exakten Navigationspfade**
- **Einmischen der neuen Funktionen gemäß dem AP-Skript**
  - **Generierung neuer Methode in der Klasse**
  - **Bei „Wiederholungsklassen“ durchreichen an alle Mitglieder**
  - **An spezifizierten Stellen Code einmischen**



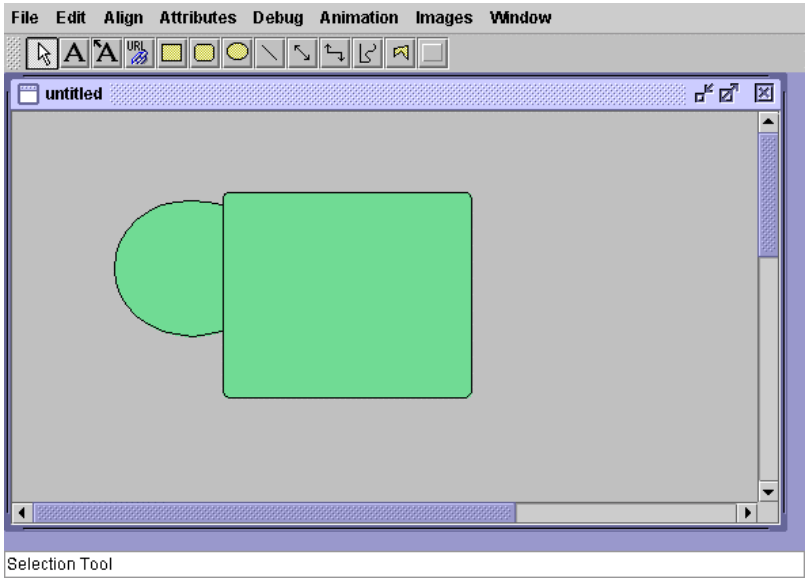
Vorgehensweise
Diskussion

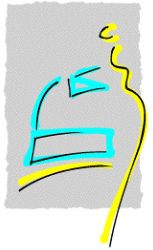
- **Vorteile:**
  - **Weniger Code**
  - **Code für neue Funktionalität an einer Stelle**
  - **Keine exakte Kenntnis der gesamten Klassenstruktur nötig**
  - **Umgang mit bestehendem Code**
    - **Erweiterungen**
    - **Änderungen**
  - **Ergebnis wieder lesbarer, erweiterbarer Code**

- **Nachteile:**
  - **Lernaufwand AP-Paradigma**

Beispiel
JHotDraw

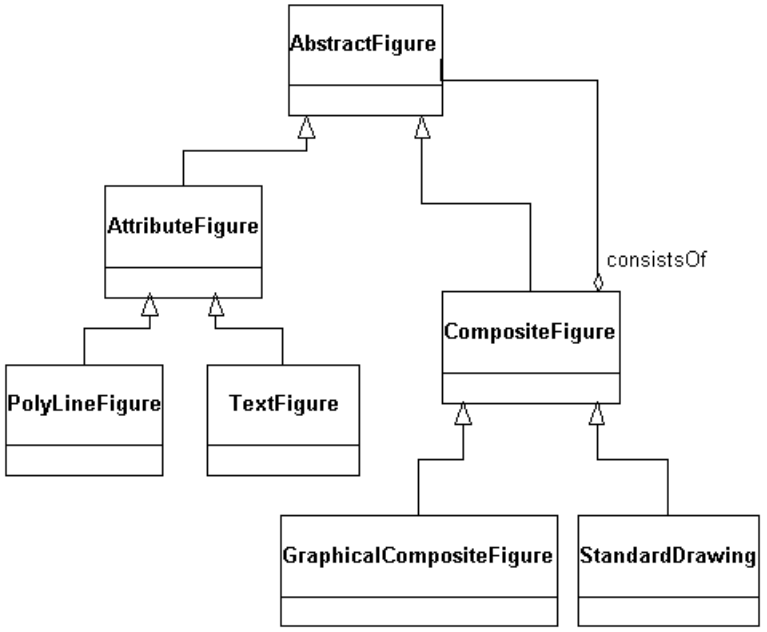
- **Java Framework für Zeichenprogramme**
- **Verwendet Entwurfsmuster**
- **Um neue Funktionen erweiterbar**
- **ca.170 Klassen**
- **ca. 15000 Lines of Code**

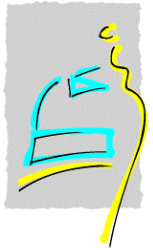




Beispiel
JHotDraw

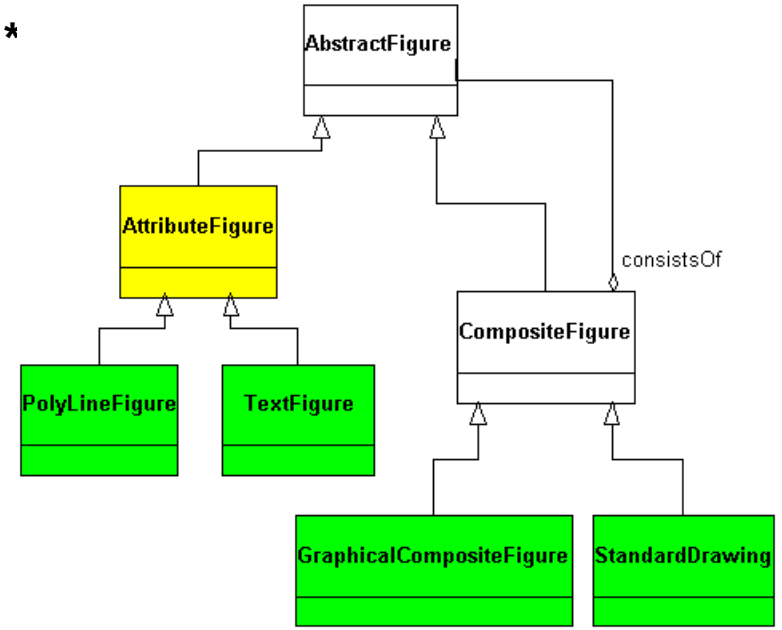
- **Aufgabe:**
  - **Spiegeln von Figuren, bzw. Gruppen an der x-Achse**
- **Vorgehen:**
  - **Klassenstruktur bestimmen**
  - **Bestimmung des Pfades**
    - **Start/Zielpunkte**
    - **Wegpunkte**
  - **Code für konkrete Klassen (TextFigure, PolyLineFigure) schreiben**

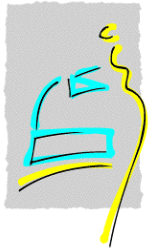




Beispiel
AP-Skript

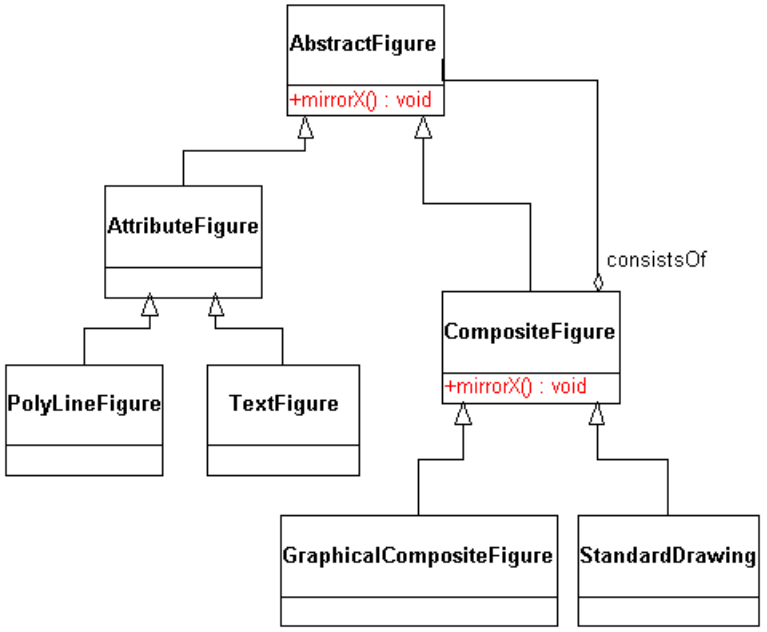
```
from {'GraphicalCompositeFigure',
      'StandardDrawing'} to *
through 'AttributeFigure', =>, *
public void mirrorX() {
   in PolyLineFigure ${
        // for each Point
        // y = -y
   }$
   in TextFigure ${
        ...
   }$...
}
```

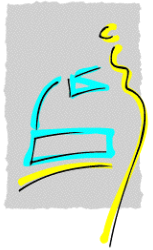




Beispiel
Ergebnis

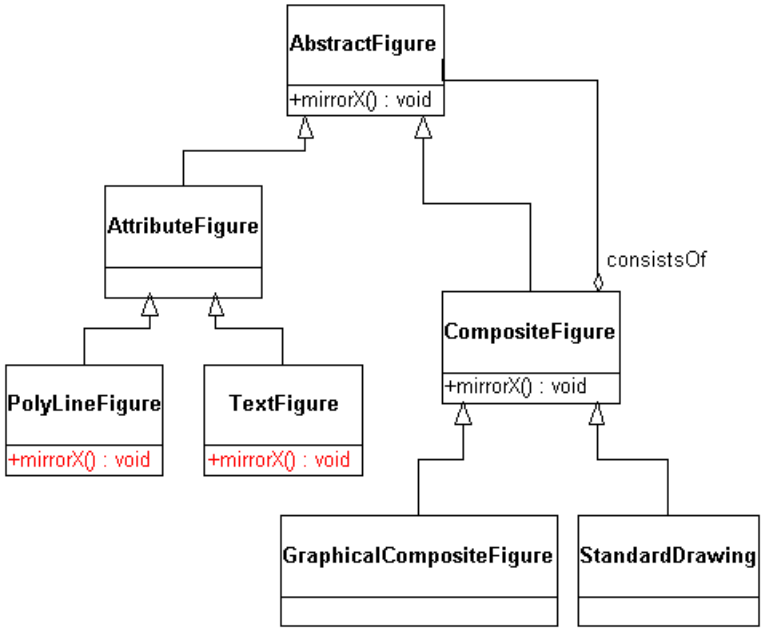
```
class CompositeFigure {
public void mirrorX {
    for (int i = 0;
          i < figures.size(); i ++){
          ((AbstractFigure)figures.
          elementAt(i)).
          mirrorX();
          }
    }
}
class AbstractFigure {
    public void mirrorX {
    }
}
```





Beispiel
Ergebnis

```
class PolyLineFigure {
...     public void mirrorX {
        // for each Point
        // y = -y
   }...
}
class TextFigure {
... public void mirrorX {
        // Textparameter
        // anpassen
   }...
}
```



Zusammenfassung und Ausblick

- **Problem:**
  - **Neue Funktionalität zu bestehender Software hinzufügen ist**
    - **Aufwendig**
    - **Fehleranfällig**
    - **Erfordert genaue Kenntnis des Systems**
- **Ziel:**
  - **Werkzeug um Erweiterungen**
    - **Schnell**
    - **Ohne tieferes Verständnis**

    durchführen zu können
  - ➢ **Änderungen bzw. Erweiterungen an bestehendem Code handhabbarer**
- **Werkzeug (in Arbeit):**
  - **Einbau des Adaptiven Sprachkonzepts in Inject/J**
  - **AP Konzepte auf existierende Software anwendbar machen**
  - ➢ **AP als Metaprogramm**