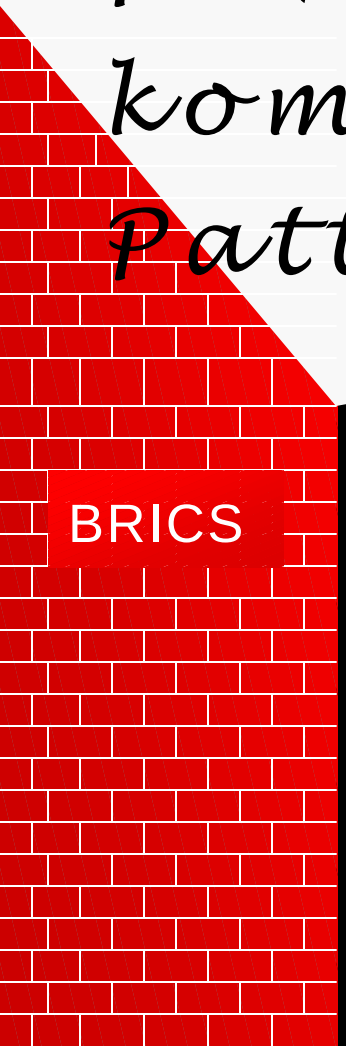
Flexibilität durch kombinierte Design Pattern



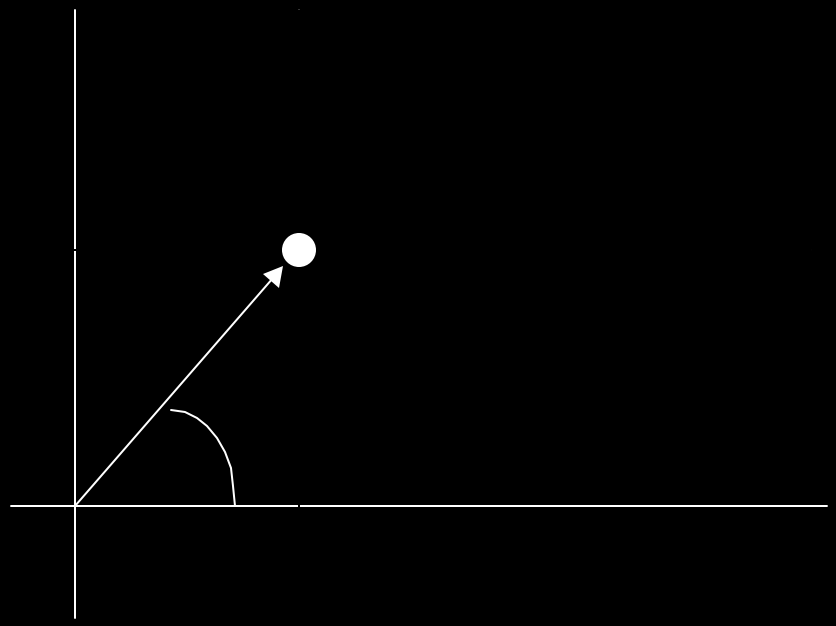
Carsten Weise

BRICS*, Universität Aalborg, DK

(*) BRICS = Basic Research in Computer Science,
Centre of the Danish National Research Foundation
http://www.brics.dk

Problemstellung

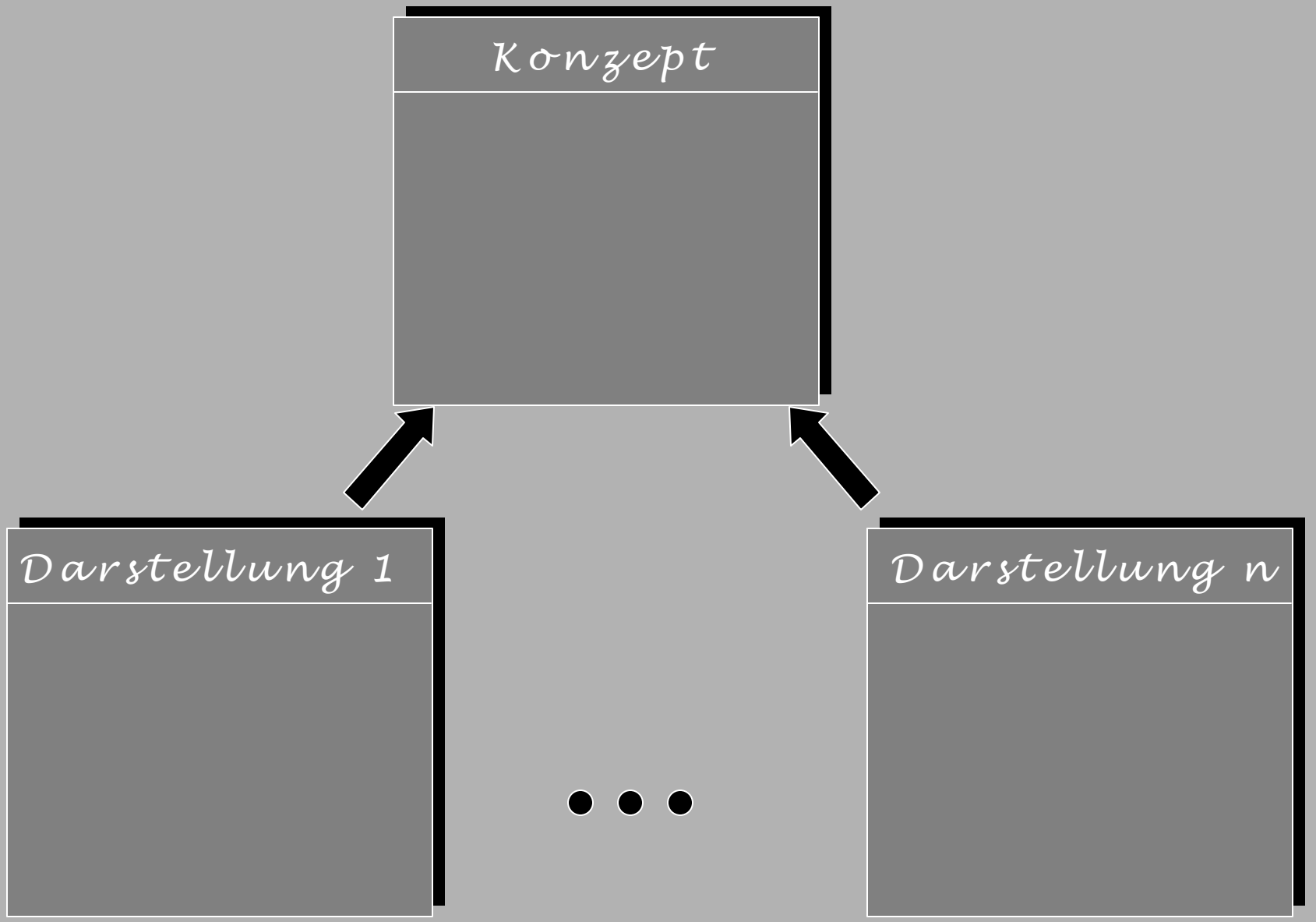
- Darstellung des gleichen Konzepts auf *verschiedene* Weisen



Anforderungen an die Lösung

- Saubere Trennung der Darstellungen
- Ausnutzen der Gemeinsamkeiten

Einfache Lösung

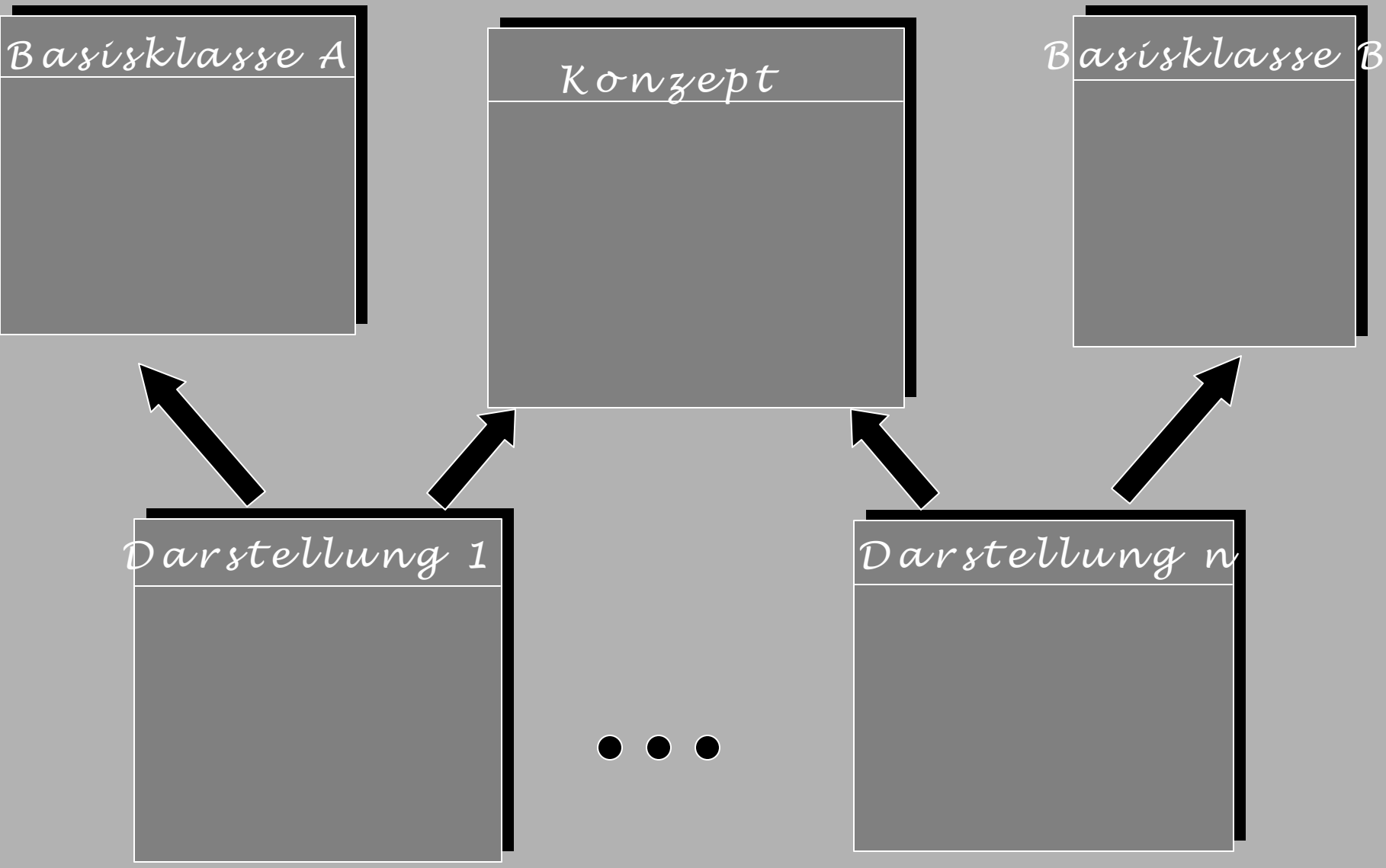


Brennpunkt

Hinzufügen neuer Darstellungsarten

Erweiterung einer bestehenden Klassenhierarchie

Nebenbedingungen



Aber keine Mehrfachvererbung!

Zusätzliche Komplikationen

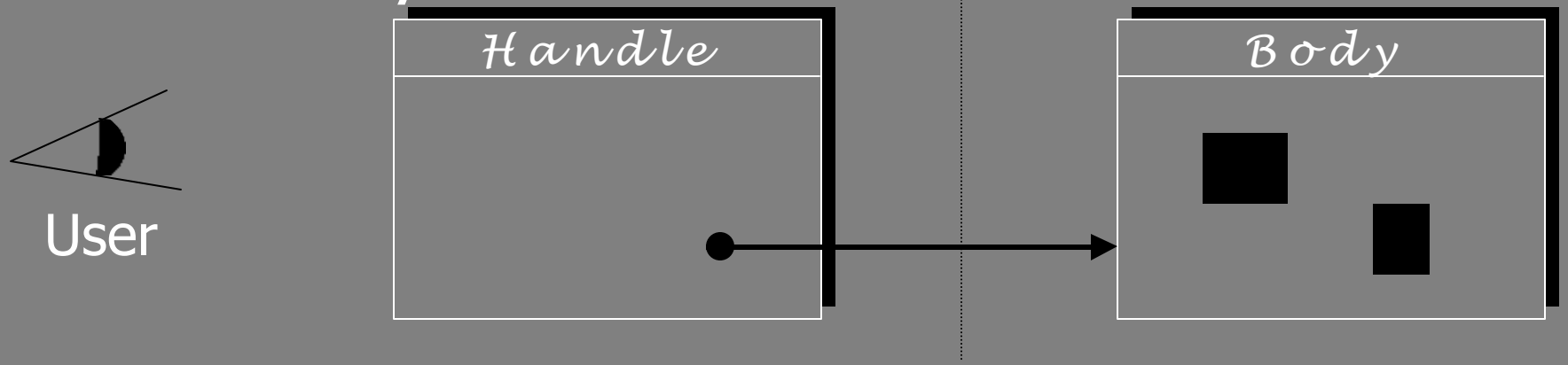
- Darstellungsart läßt sich nicht einfach in die bestehende Architektur einfügen

Lösung: Kombination von Design Pat

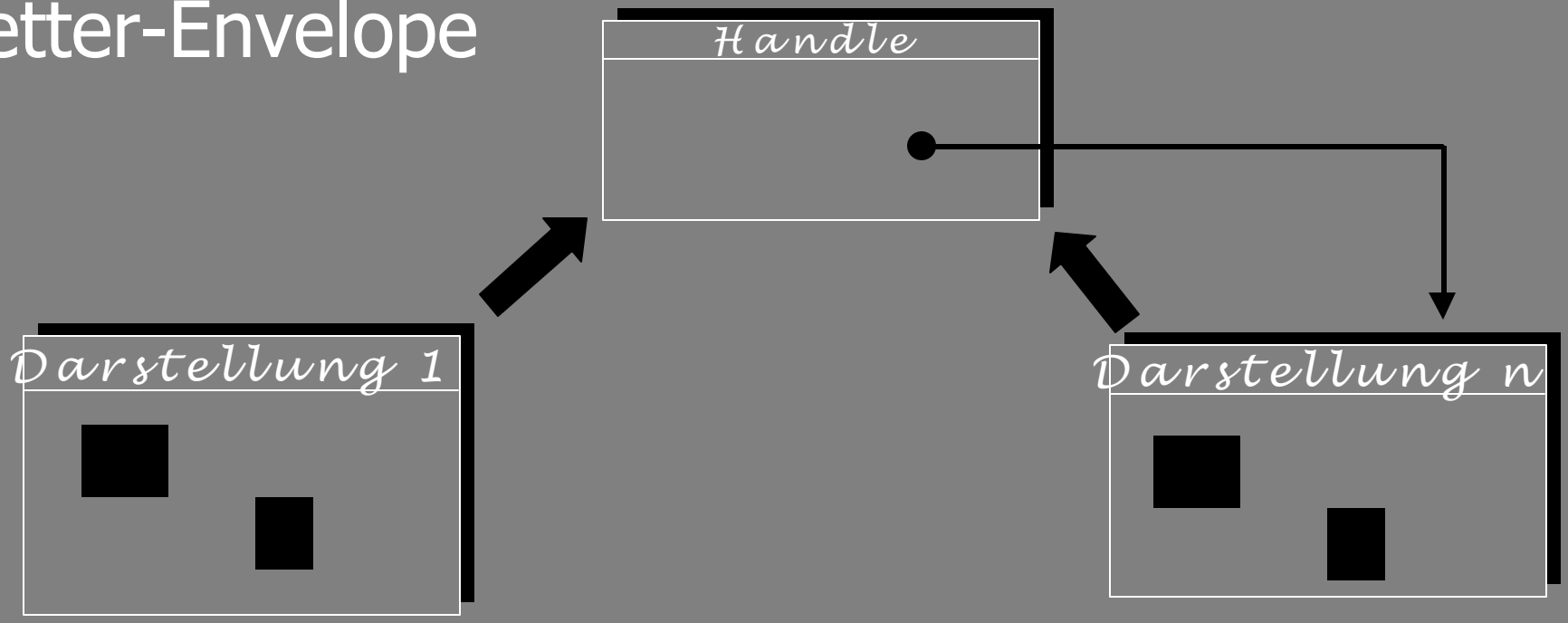
- einfach
- erweiterbar
- robust

Design Pattern

Handle-Body

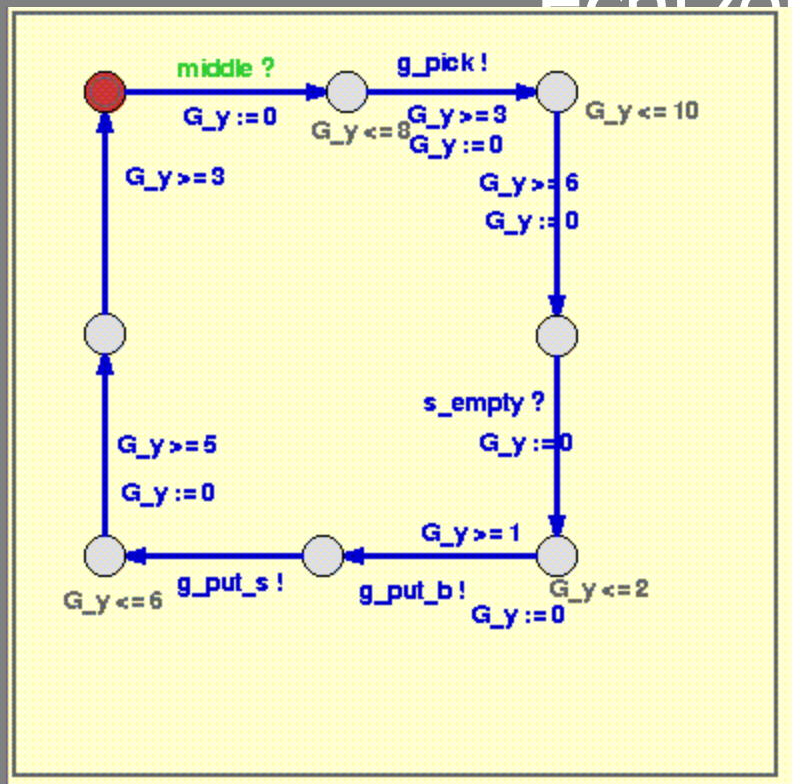


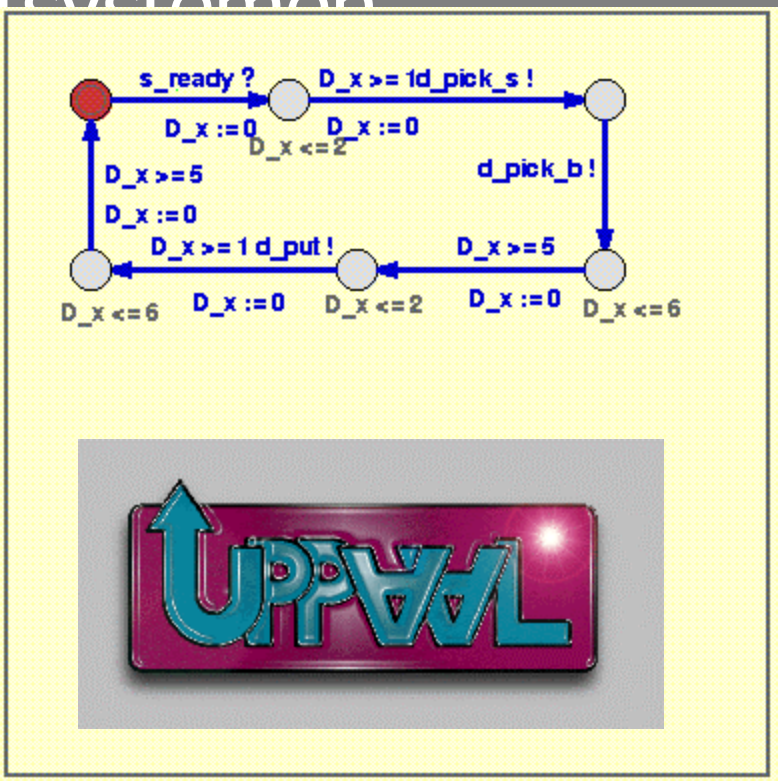
Letter-Envelope



Fallstudie

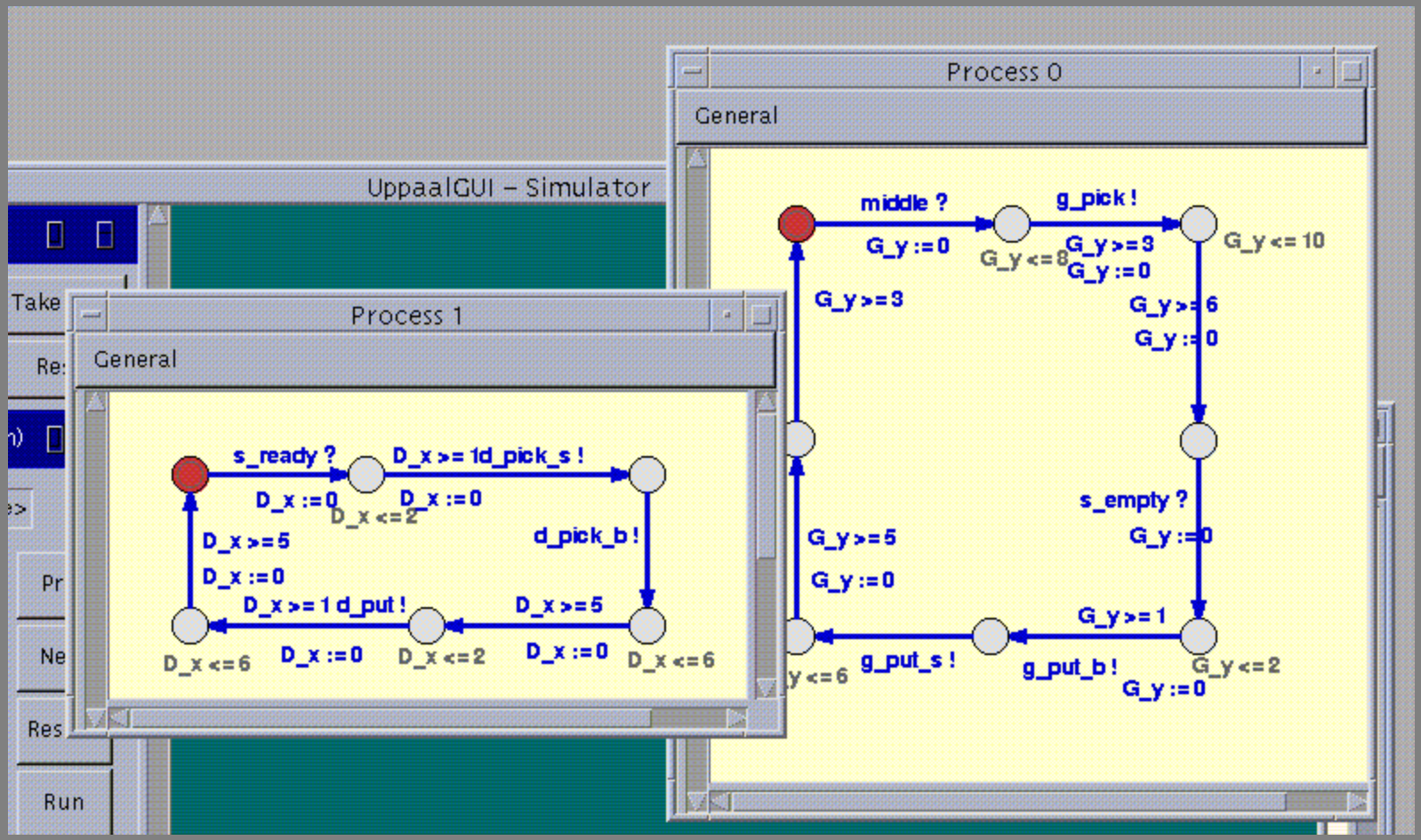
$\mathcal{U}ppaal$: Werkzeug für Modellierung, Simulation und Validierung von Echtzeitsystemen





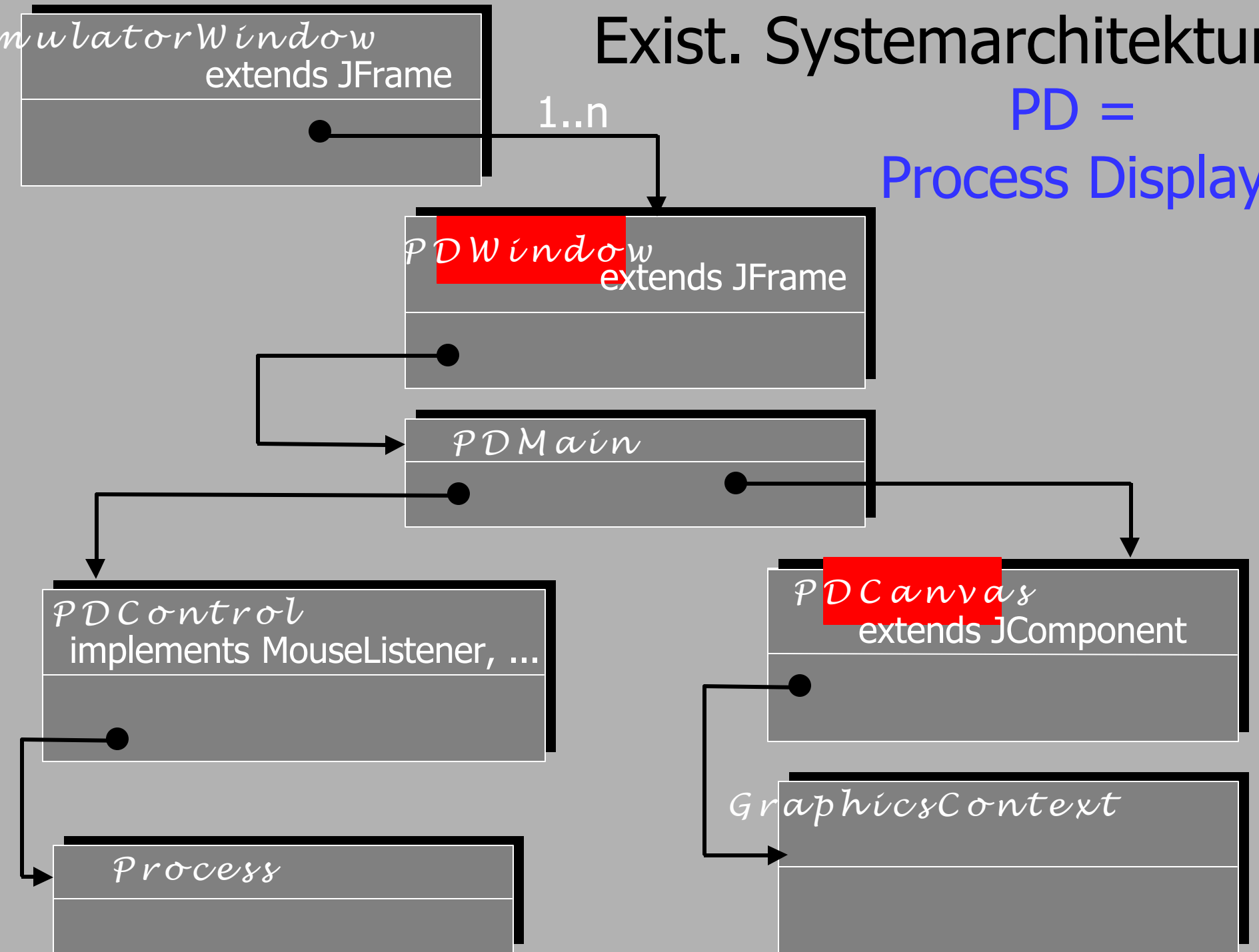
http://www.docs.uu.se/uppaal.shtml

(b) “alte” Darstellung

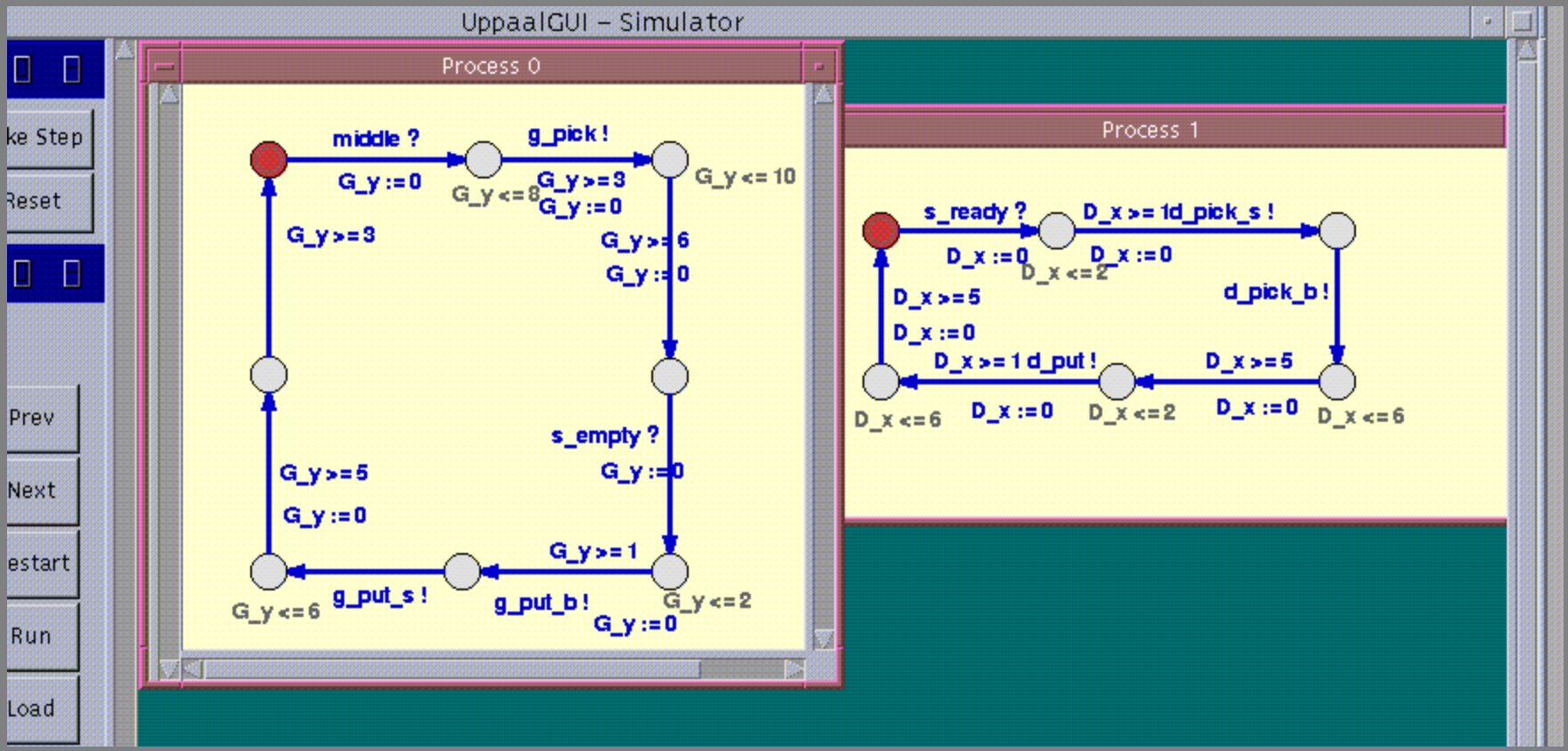


Exist. Systemarchitektur

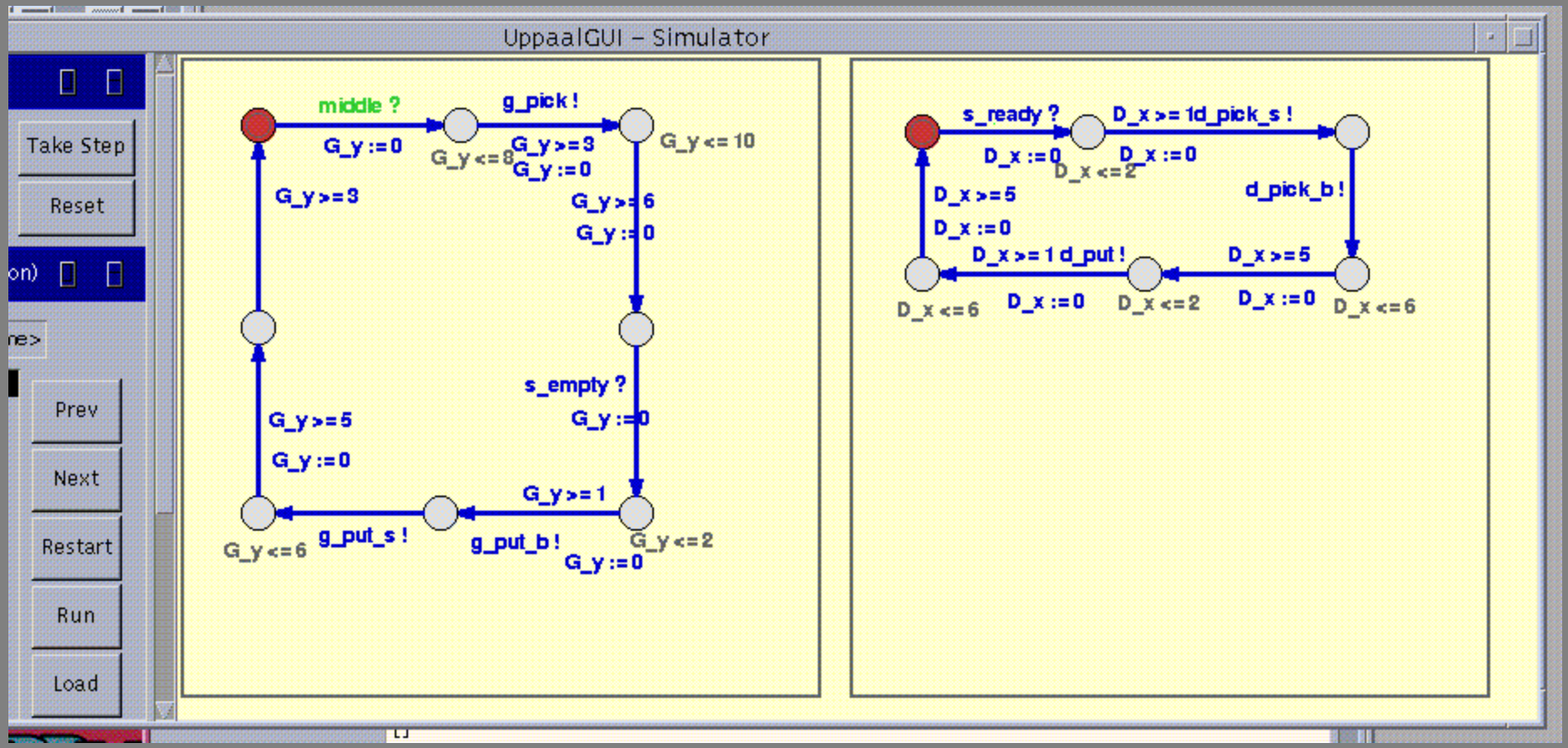
PD =
Process Display



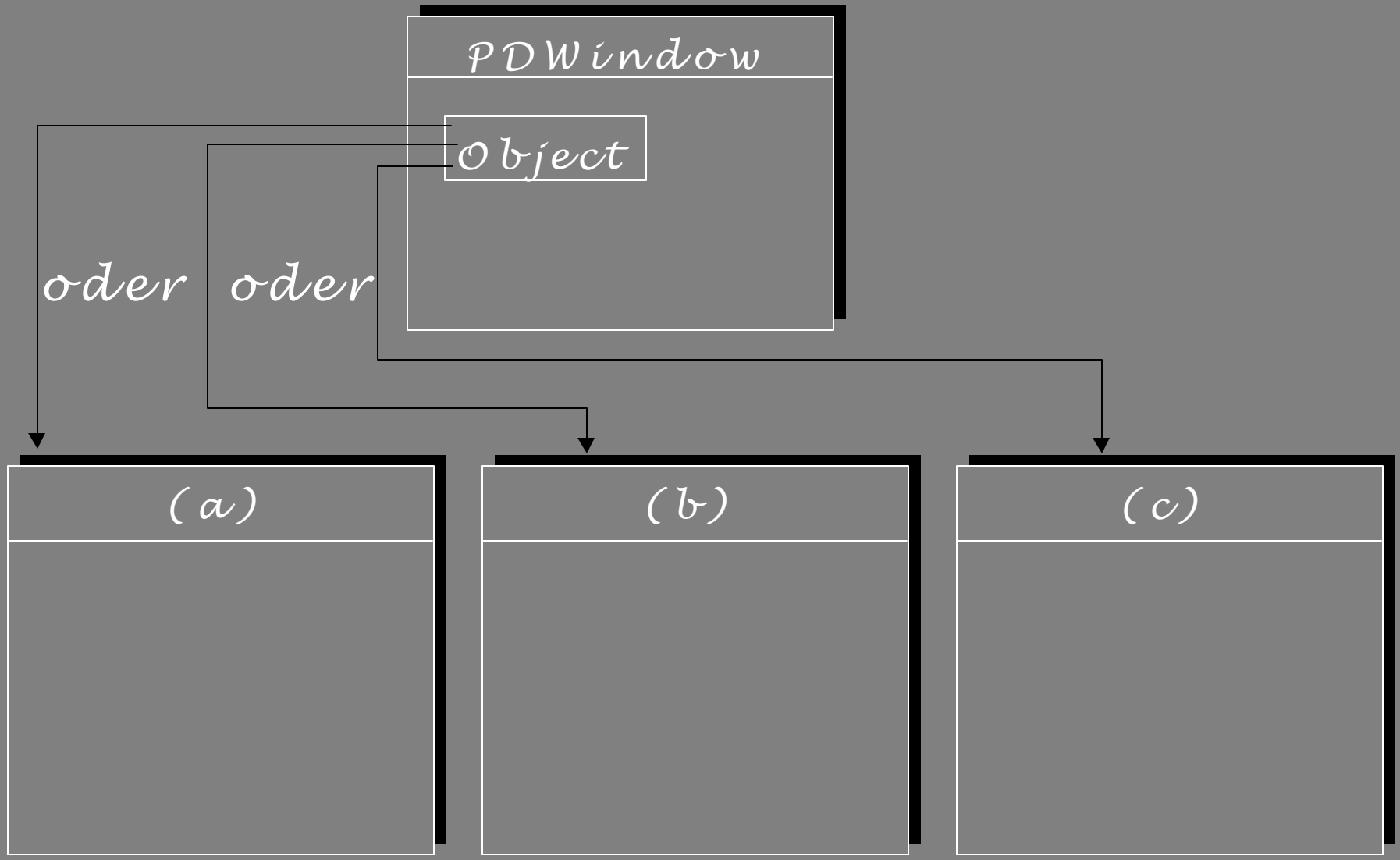
Darstellung (c)



Darstellung (a)



Flexible Implementierung



Analog für PDCanvas

isolierte Implementierung

- (b) Fenster/Prozeß: trivial
- (c) internes Fenster/Prozeß: einfach, ersetze JFrame durch JInternalFrame

Bestehende Systemarchitektur

Simulator = n Prozesse

1 zu 1

JFrame
+ Canvas
+ Control

Prozeß

Isolierte Implementierung (a)

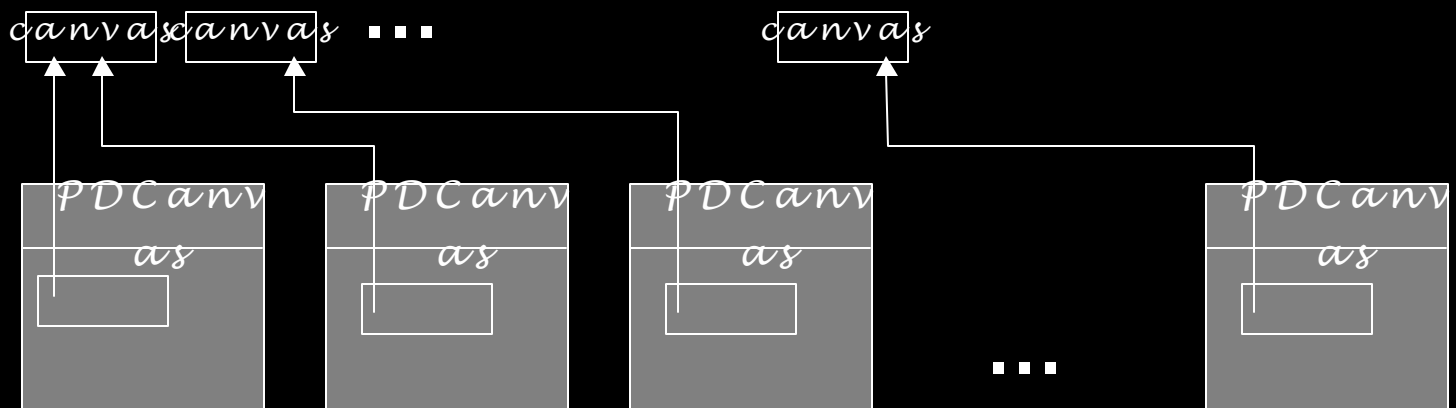
Simulator = JFrame
+ Canvas
+ Control

DisplayArea — 1 zu 1 — Prozeß

Canvas — 1 zu n — Prozeß 1
...
Prozeß n

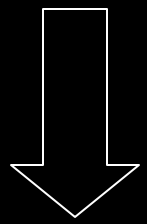
Implementierung von (a)

- Ersetze Objekt-Member (Canvas) durch Liste von Class-Member



Weiteres Problem

Art des PDCanvas hängt von PDWindow ab



PDWindow erhält Methode zum Erzeugen des “richtigen” PDCanvas

=

Factory-Methode

Schnittstellenkompatibilität

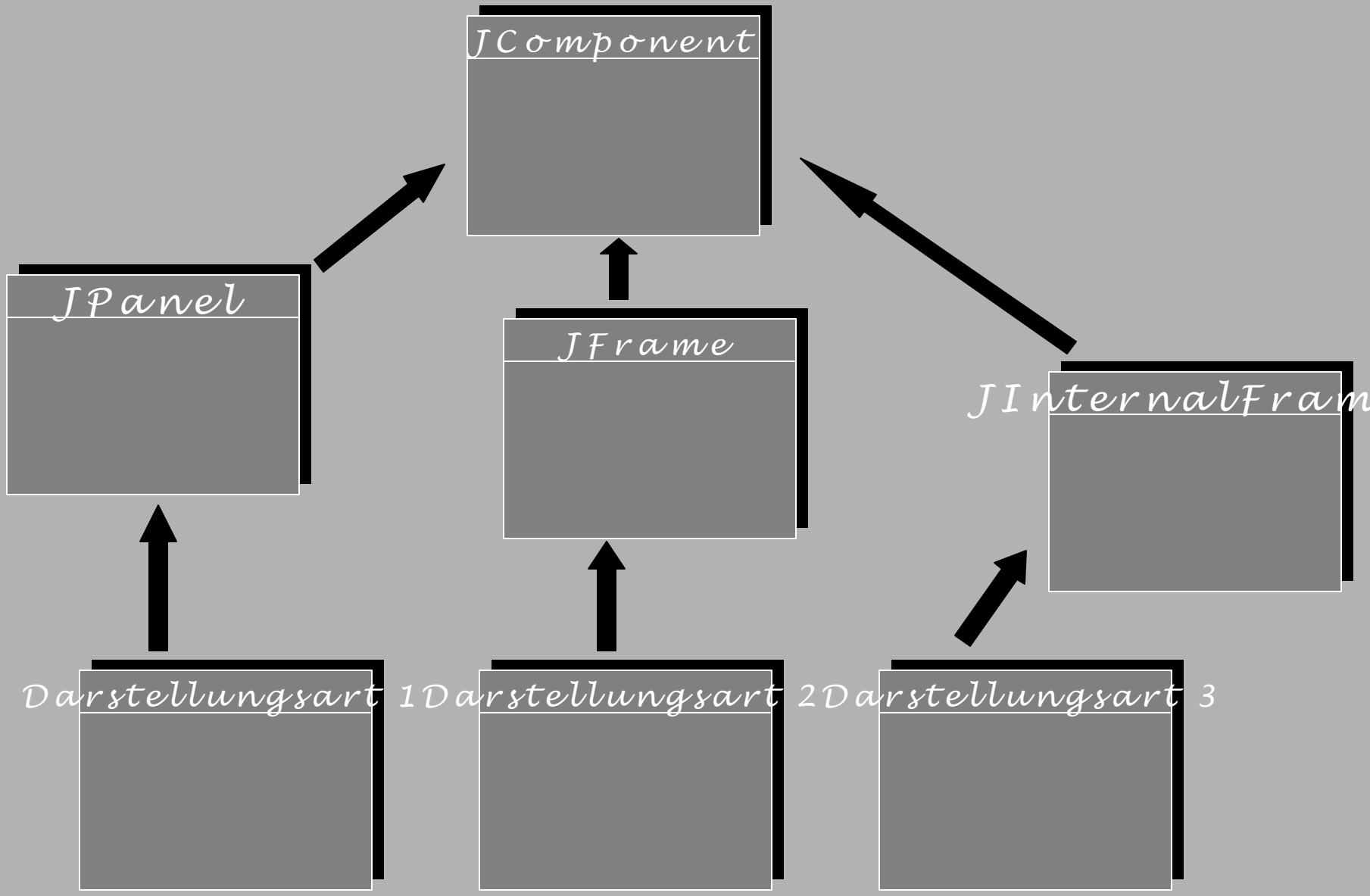
kann immer gewährt werden

Typkompatibilität

Basisklassen der ursprünglichen Implementierung?

ok, wenn alle Darstellungsformen auch diese Basisklasse besitzen

Situation hier



Typkompatibilität

Basisklassen der ursprünglichen Implementierung?

ok, wenn alle Darstellungsformen auch diese Basisklasse besitzen

sonst braucht man komplexere Design Pattern (z.B. Proxy)

Anwendung

- Erweiterung mit neuen Darstellungsformen
- Verzahnung mehrerer Ebenen
- verschieben einer 1-zu-1 zu einer n-zu-m Beziehung

Kombination von Design Patterns

- einfach
- robust
- erweiterbar

verallgemeinerbar?