Beobachter

Ricco Hamm, André Martin & Stephan Dorer

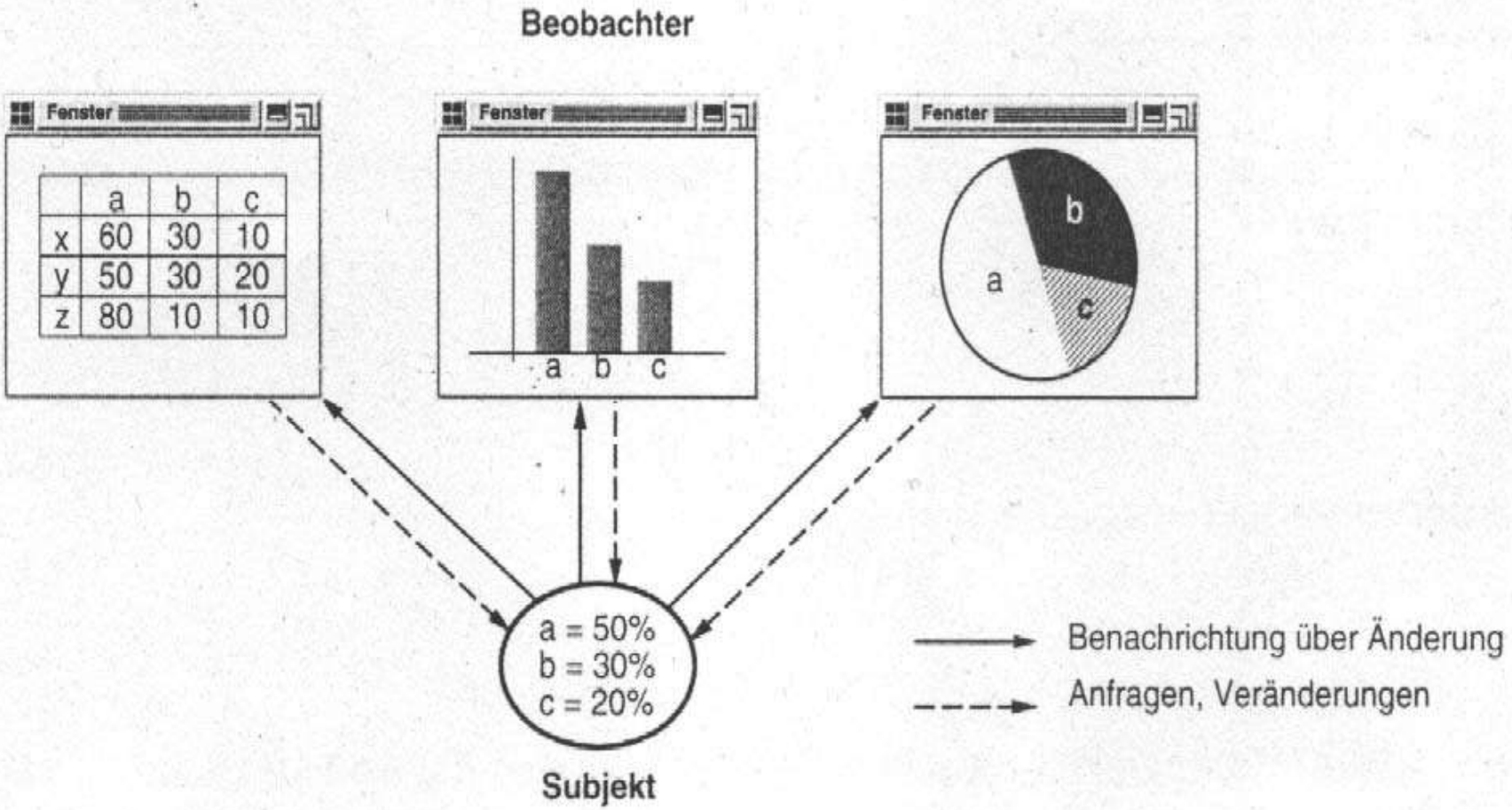
Zweck

- 1-zu-n Abhängigkeiten zwischen Objekten
- Änderung des Zustands eines Objekts => Benachrichtigung und Aktualisierung aller abhängigen Objekte

Motivation

- Aufrechterhaltung der Konsistenz zwischen mit einander in Beziehung stehenden Objekten bei Änderungen
- Enge Kopplung der Klassen wegen Wiederverwendbarkeit nicht wünschenswert

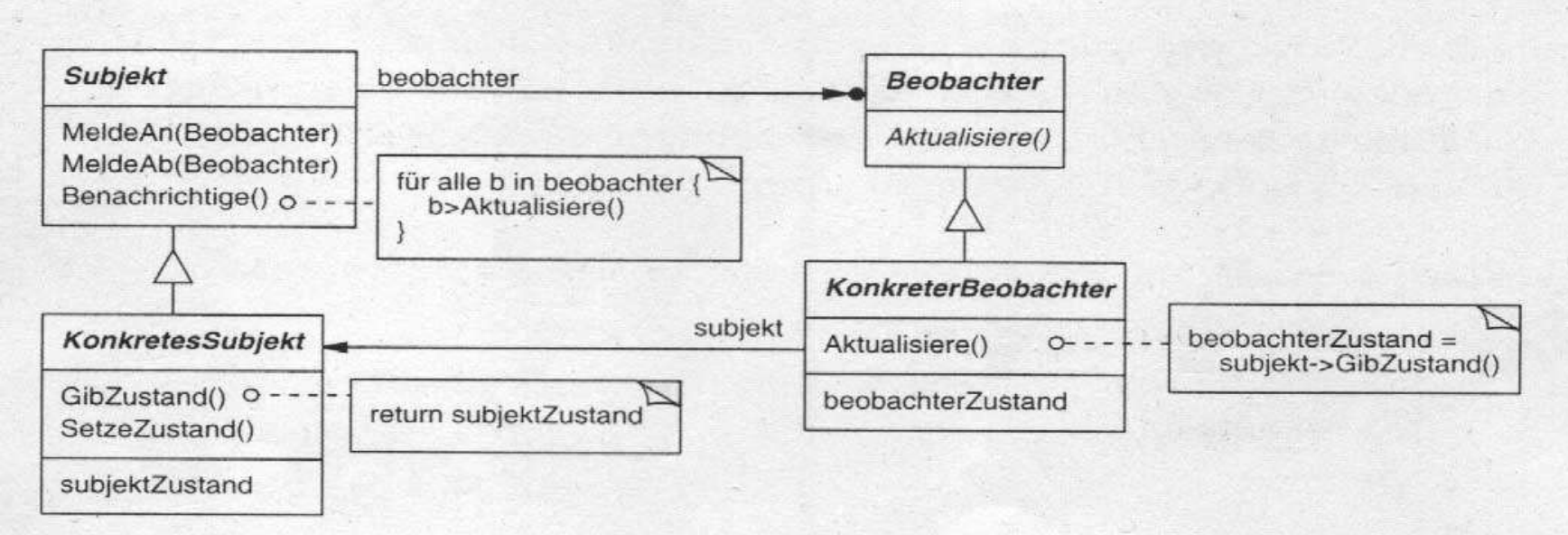
Motivation


Beobachter
Fenster
Fenster
Fenster
a b c
x 60 30 10
y 50 30 20
z 80 10 10
a b c
b
a
c
a = 50%
b = 30%
c = 20%
Subjekt
Benachrichtung über Änderung
Anfragen, Veränderungen

Anwendbarkeit

- Wenn man 2 Aspekte besitzt, von denen einer vom Anderen abhängt
- Wenn Änderung eines Objektes die Änderung anderer Objekte nach sich zieht
- Wenn ein Objekt andere Objekte benachrichtigen soll ohne sie explizit zu kennen (keine enge Kopplung der Objekte)

Struktur


Subjekt
MeldeAn(Beobachter)
MeldeAb(Beobachter)
Benachrichtige()
beobachter
Beobachter
Aktualisiere()
für alle b in beobachter {
b>Aktualisiere()
}
KonkreterBeobachter
Aktualisiere()
beobachterZustand
beobachterZustand =
subjekt->GibZustand()
subjekt
KonkretesSubjekt
GibZustand()
SetzeZustand()
subjektZustand
return subjektZustand

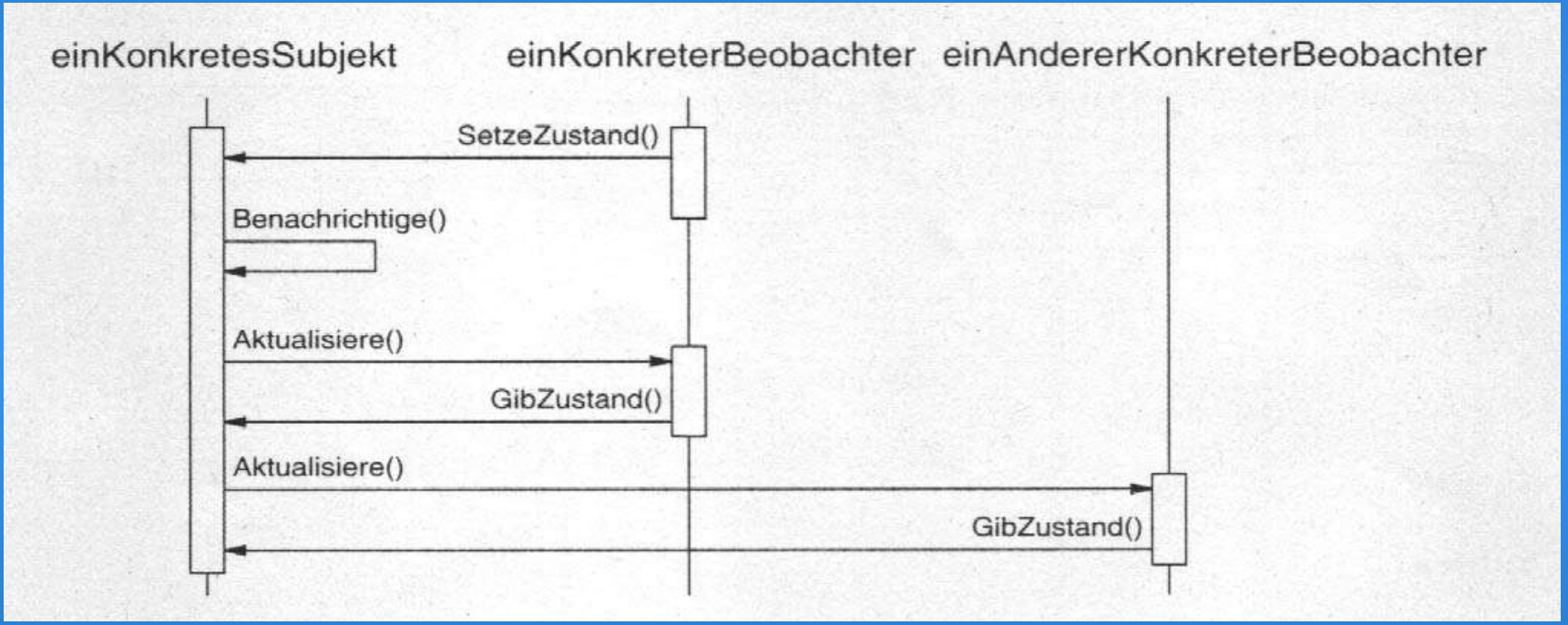
Teilnehmer

- Subjekt:
  - kennt seine Beobachter (beliebig viele)
  - bietet Schnittstelle zum An- und Abmelden von Beobachtern
- Beobachter:
  - definiert Schnittstelle für Objekte, die über Änderungen eines Subjektes benachrichtigt werden sollen

Teilnehmer

- konkretes Subjekt:
  - speichert für konkreten Beobachter relevanten Zustand
  - benachrichtigt seine konkreten Beobachter, wenn sich sein Zustand ändert
- konkreter Beobachter:
  - verwaltet eine Referenz auf konkretes Subjekt
  - speichert den Zustand, der mit dem des Subjekts im Einklang stehen soll
  - implementiert die Aktualisierungsschnittstelle

Interaktion



Konsequenzen

- Abstrakte Kopplung zwischen Subjekt und Beobachter
  - Kennt Liste der Beobachter & Schnittstelle
- Unterstützung von Broadcastkommunikation
  - Automatische Benachrichtigung an registrierte Beobachter
- Unerwartete Aktualisierungen
  - Harmlose Änderung kann zu vielen Änderungen bei den Beobachtern führen
  - Nachteil: Aktualisierung sagt nichts darüber aus, was sich geändert hat.

Implementation

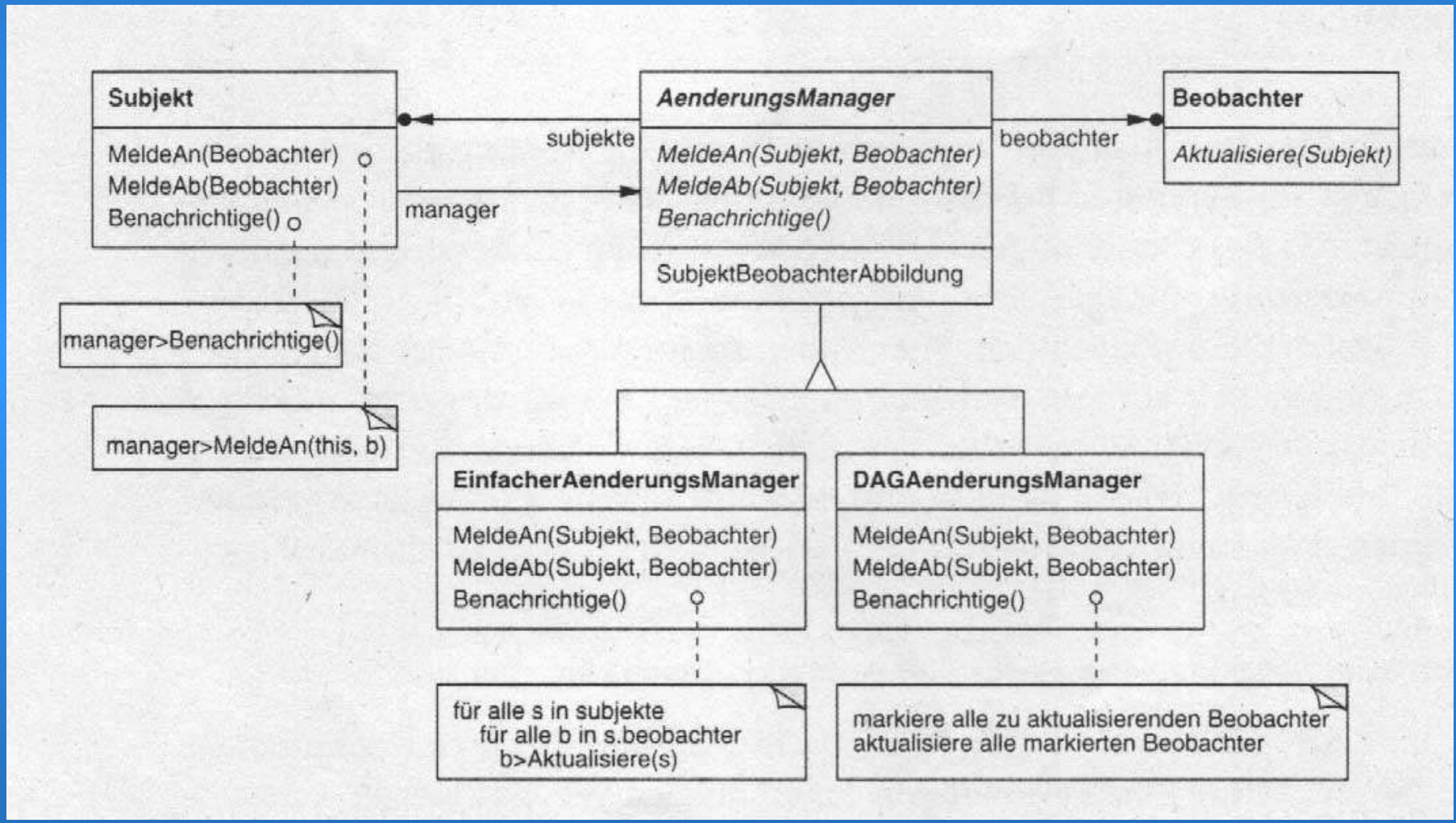
1. Abbildung von Subjekten auf ihre Beobachter
   - Problem: Jedes Subjekt hat Liste von Beobachtern
   - Lösung: Zugriff auf Beobachter durch assoziatives Lookup (z.B. Hashtabelle)

2. Beobachten von mehr als einem Subjekt
   - Erweiterung der Aktualisierungsschnittstelle
   - Vermeidung unnötiger Subjektanfragen

3. Auslösen der Aktualisierung
   a) zustandsändernde Operationen rufen Benachrichtige() auf
   b) Klient muss selbst Benachrichtige() aufrufen
4. Fehlerhafte Referenzen auf gelöschte Subjekte
   - Subjekte teilen Beobachter mit, wenn sie gelöscht werden
5. Sicherstellen, daß der Subjektzustand vor der Benachrichtigung konsistent ist
   - Benachrichtige() wird erst am Ende der Änderungen des Subjektes aufgerufen

6. Vermeiden von beobachterspezifischen Aktualisierungsschnittstellen: Push- und Pull-Modell
   - Push-Modell: detaillierte Informationen über Änderungen
   - Pull-Modell: minimale Informationen über Änderungen (Nachfrage notwendig)
7. Explizites Festlegen der interessierenden Änderungen
   - Aspekt-Konzept
   - Beobachter übermitteln Interessen bei Anmeldung

8. Kapseln komplexer Aktualisierungssemantik



9. Kombinieren von Subjekt- und Beobachterklassen

- Bei Sprachen, die keine Mehrfachvererbung unterstützen werden Subjekt und Beobachter in eine Klassen gepackt

- z.B. Smalltalk: Subjekt- und Beobachterschnittstellen in Wurzelklasse Object definiert