Nebenläufigkeit

Marvin Frommhold

Seminar Software Design Patterns (SS09)

Betriebliche Informationssysteme
Institut für Informatik
Universität Leipzig

8. Juli 2009

Gliederung

Worum geht es?

Nebenläufigkeit

Menge von Verfahren und Mechanismen, die es mehreren Threads/Prozessen ermöglichen, ihnen zugeteilte Aufgaben gleichzeitig auszuführen.

Prozess vs. Thread

Prozess

Ansammlung von Betriebsmitteln, welche Programminstruktionen ausführt

Thread

konkrete Folge von Programmschritten innerhalb eines Prozesses

Thread

. . . auch Ablauffaden gennant

- (deutlich) leichtgewichtiger als ein Prozess
- Ausführungseinheit innerhalb eines Prozesses
- teilt sich Adressraum und Ressourcen mit anderen Threads

Thread

Vorteile

- transparente Performanzsteigerung
- explizite Performanzsteigerung
- Verbesserung der Antwortzeiten
- Vereinfachung des Entwurfs von Anwendungen

Thread

Nachteile

- Vorteile schwer zu realisieren
- Zugriffskonflikte
- Verklemmungen
- Bestimmung einer effizienten Nebenläufigkeitsarchitektur

Jetzt gleich . . .

Nebenläufigkeit

Muster

- Active Object
- Monitor Object
- Leader/Followers

Zusammenfassung

Quellen/Abbildungen

Active Object

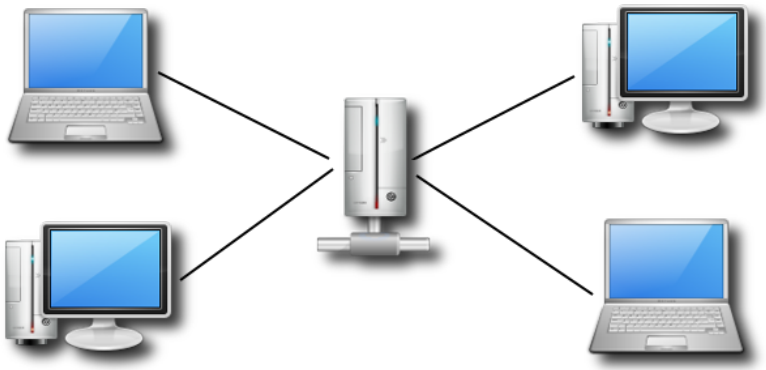
Kontext

. . . oder auch **Concurrent Object**

Active Object

Kontext

. . . oder auch **Concurrent Object**

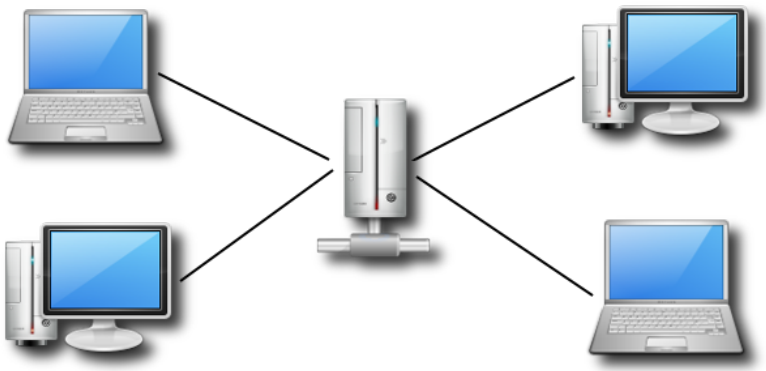


- ein Server
- viele Clients
- parallele Verarbeitung komplexer Anfragen

Active Object

Kontext

. . . oder auch **Concurrent Object**



- ein Server
- viele Clients
- parallele Verarbeitung komplexer Anfragen

Nutzung von Nebenläufigkeit

Active Object

Problem

Zugriff auf nebenläufiges Objekt aus mehreren Threads

Active Object

Problem

Zugriff auf nebenläufiges Objekt aus mehreren Threads

$\longrightarrow$ Synchronisation

Active Object

Problem

Zugriff auf nebenläufiges Objekt aus mehreren Threads
$\longrightarrow$ Synchronisation

Anforderungen

- Synchronisation transparent für Client
- Anwendung soll nicht (lange) blockieren

Trennung von Methodenaufruf und Methodenausführung

Active Object

Aufbau

Proxy

repräsentiert Schnittstelle des Active Object

Active Object

Aufbau

Proxy

repräsentiert Schnittstelle des Active Object

Servant

Implementierung des Active Object

Active Object

Aufbau

Proxy
: repräsentiert Schnittstelle des Active Object

Servant
: Implementierung des Active Object

Method Request
: stellt Methodenaufruf des Clients dar

Active Object

Aufbau

Proxy
: repräsentiert Schnittstelle des Active Object

Servant
: Implementierung des Active Object

Method Request
: stellt Methodenaufruf des Clients dar

Activation List
: Liste zum Speichern der Method Requests

Active Object

Aufbau

Proxy
: repräsentiert Schnittstelle des Active Object

Servant
: Implementierung des Active Object

Method Request
: stellt Methodenaufruf des Clients dar

Activation List
: Liste zum Speichern der Method Requests

Scheduler
: Verwaltung der Method Requests

Active Object

Aufbau

Proxy
: repräsentiert Schnittstelle des Active Object

Servant
: Implementierung des Active Object

Method Request
: stellt Methodenaufruf des Clients dar

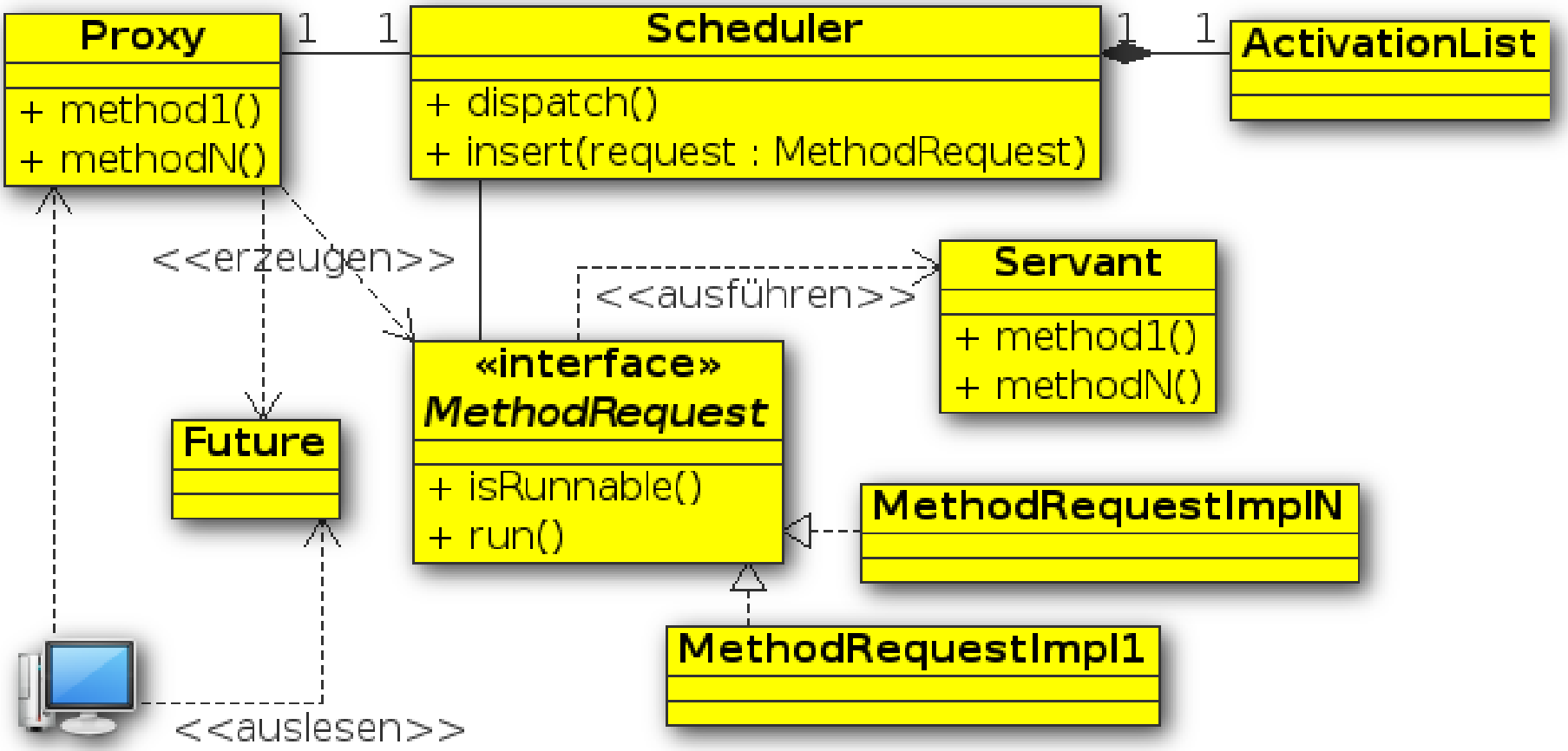
Activation List
: Liste zum Speichern der Method Requests

Scheduler
: Verwaltung der Method Requests

Future
: Rendezvous-Punkt für Client, speichert Resultat des Methodenaufrufs

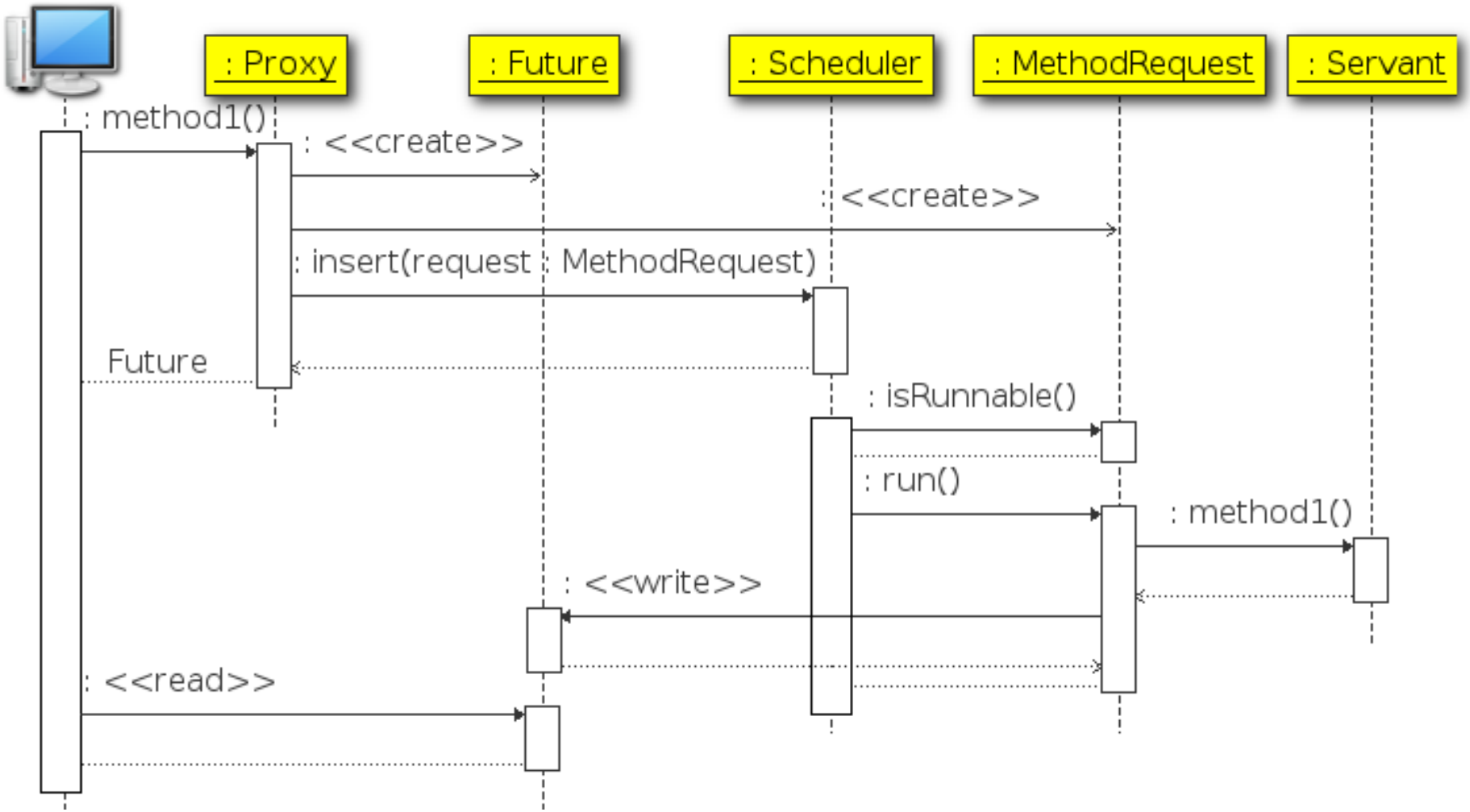
Active Object

Struktur



Active Object

Verhalten



Active Object

Vor- und Nachteile

- verbesserte Nebenläufigkeit auf Anwendungsebene
- Vereinfachung der Synchronisationskomplexität
- Reihenfolge der Aufrufe kann sich zu der der Ausführung unterscheiden
- transparentes Nutzen verfügbarer Parallelität

Active Object

Vor- und Nachteile

- verbesserte Nebenläufigkeit auf Anwendungsebene
- Vereinfachung der Synchronisationskomplexität
- Reihenfolge der Aufrufe kann sich zu der der Ausführung unterscheiden
- transparentes Nutzen verfügbarer Parallelität

- Performanzaufwand
- komplizierte Fehlerbereinigung

Als nächstes . . .

Nebenläufigkeit

Muster

- Active Object
- Monitor Object
- Leader/Followers

Zusammenfassung

Quellen/Abbildungen

Monitor Object

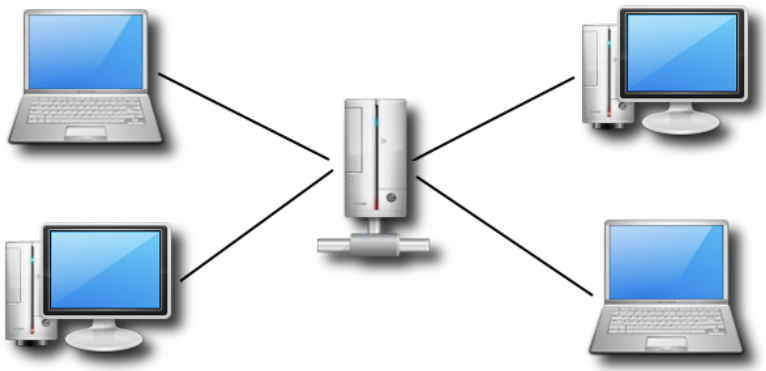
Kontext

. . . oder auch **Thread-Safe Passive Object**

Monitor Object

Kontext

. . . oder auch **Thread-Safe Passive Object**

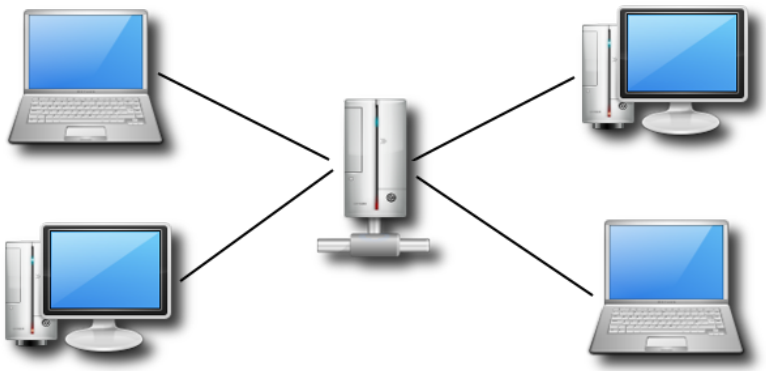


- ein Server
- viele Clients
- parallele Verarbeitung leichtgewichtiger Anfragen

Monitor Object

Kontext

. . . oder auch **Thread-Safe Passive Object**



- ein Server
- viele Clients
- parallele Verarbeitung leichtgewichtiger Anfragen

Active Object?

Monitor Object

Problem

Zugriff auf ein leichtgewichtiges Objekt aus mehreren Threads

Monitor Object

Problem

Zugriff auf ein leichtgewichtiges Objekt aus mehreren Threads

$\longrightarrow$ Synchronisation

Monitor Object

Problem

Zugriff auf ein leichtgewichtiges Objekt aus mehreren Threads
$\longrightarrow$ Synchronisation

Anforderungen

- Infrastruktur von Active Object zu überdimensioniert
  $\longrightarrow$ unnötiger Performanz- und Programmieraufwand
- Zugriff auf Objekt muss thread-safe sein
- zu jeder Zeit stabiler Objektzustand

Monitor Object

Lösung

**Zugriff synchronisieren,
so dass nur eine Methode gleichzeitig
ausgeführt werden kann**

Monitor Object

Aufbau

Monitor Objekt

besitzt synchronisierte Methoden

Monitor Object

Aufbau

Monitor Objekt

besitzt synchronisierte Methoden

Monitor Lock

sperrt den Zugriff auf das Objekt

Monitor Object

Aufbau

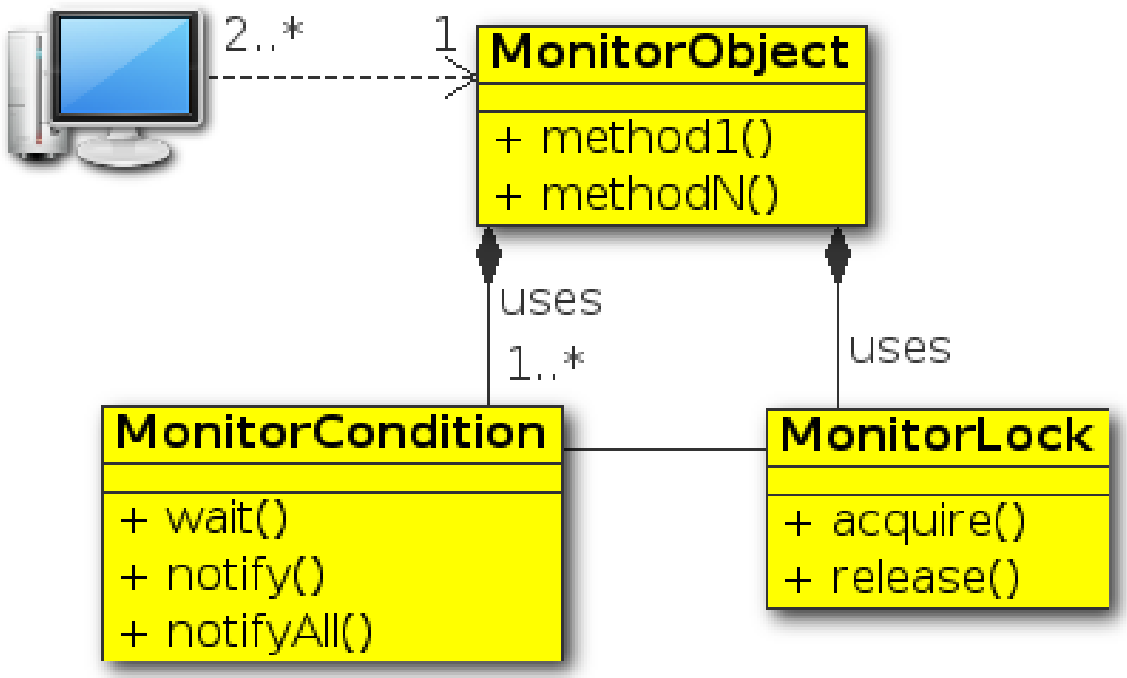
Monitor Objekt
: besitzt synchronisierte Methoden

Monitor Lock
: sperrt den Zugriff auf das Objekt

Method Condition
: bestimmt, wann synchronisierte Methode Ausführung aussetzt bzw. wieder aufnimmt

Monitor Object

Struktur



Monitor Object

Vor- und Nachteile

- vereinfachte Steuerung der Nebenläufigkeit
- vereinfachte Steuerung der Methodenausführung

Monitor Object

Vor- und Nachteile

- vereinfachte Steuerung der Nebenläufigkeit
- vereinfachte Steuerung der Methodenausführung

- schlechte Erweiterbarkeit
- Nested Monitor Lockout

```
1  class Inner {
2    protected boolean condition c = false;
3
4    synchronized void awaitCondition() {
5      while (!condition) {
6        try { wait(); } catch (InterruptedException ie) {}
7        // weiterer Code
8      }
9    }
10
11   synchronized void notifyCondition(boolean c) {
12     this.condition = c; notifyAll();
13   }
14 }
15
16 class Outer {
17   protected Inner inner = new Inner();
18
19   synchronized void process() {
20     inner.awaitCondition();
21   }
22
23   synchronized void set(boolean c) {
24     inner.notifyCondition(c);
25   }
26 }
```

Nun kommt ...

Leader/Followers

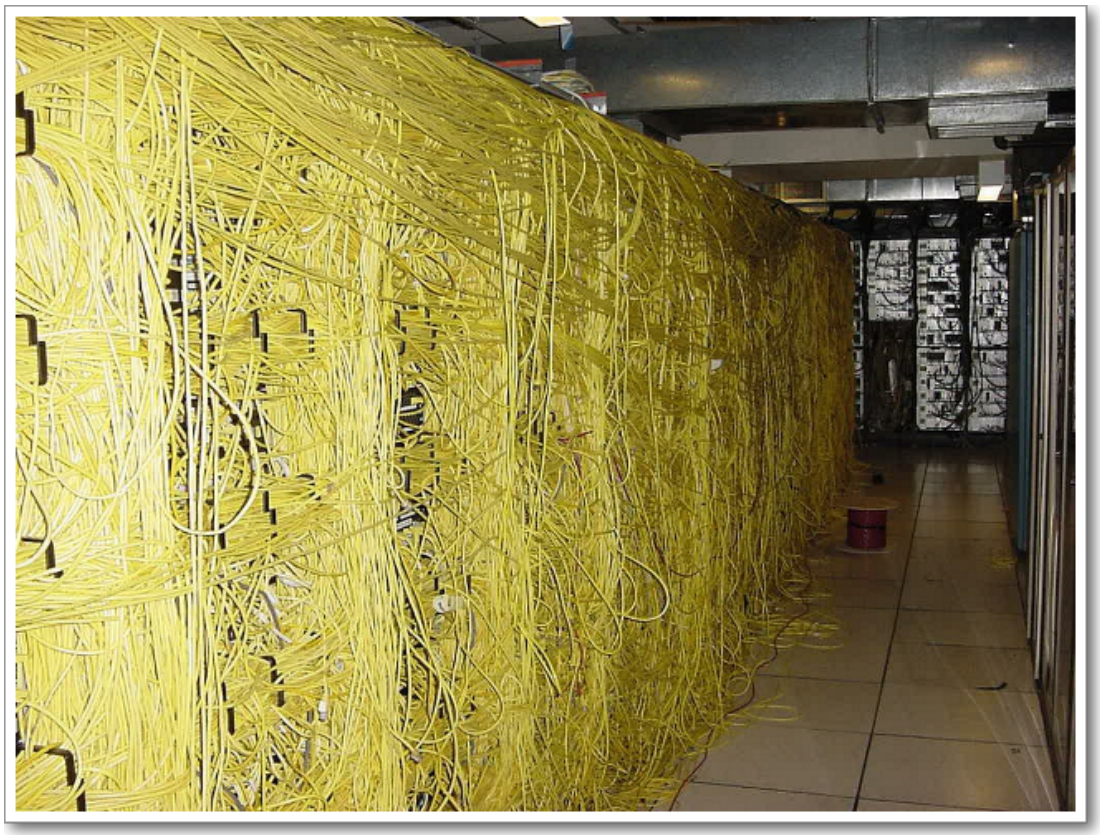
Kontext

- hochperformanter nebenläufiger Server mit vielen Ereignisquellen

Leader/Followers

Kontext

- hochperformanter nebenläufiger Server mit vielen Ereignisquellen



Leader/Followers

Problem

- Anfragen treffen bei mehreren Ereignisquellen ein
  $\longrightarrow$ Ein Thread pro Quelle? Skalierbarkeit?
- maximale Performanz auch unter hoher Last
  $\longrightarrow$ Kontextwechsel, Synchronisationsaufwand
- Threads nutzen gemeinsam Menge von Ereignisquellen
  $\longrightarrow$ Zugriffskonflikte

Leader/Followers

Lösung

**Thread-Pool,
dessen Threads sich gegenseitig abwechseln, um eintreffende Ereignisse von gemeinsamen Ereignisquellen zu verarbeiten**

Leader/Followers

Aufbau

Handle

Identifikator, welcher Ereignisquelle bezeichnet

Leader/Followers

Aufbau

Handle
: Identifikator, welcher Ereignisquelle bezeichnet

Handle Set
: Menge von Handles

Leader/Followers

Aufbau

Handle
: Identifikator, welcher Ereignisquelle bezeichnet

Handle Set
: Menge von Handles

Event Handler
: definiert Schnittstelle zur Verarbeitung (Hook-Methoden) von Ereignissen, die bei einem Handle eingehen

Leader/Followers

Aufbau

Handle
: Identifikator, welcher Ereignisquelle bezeichnet

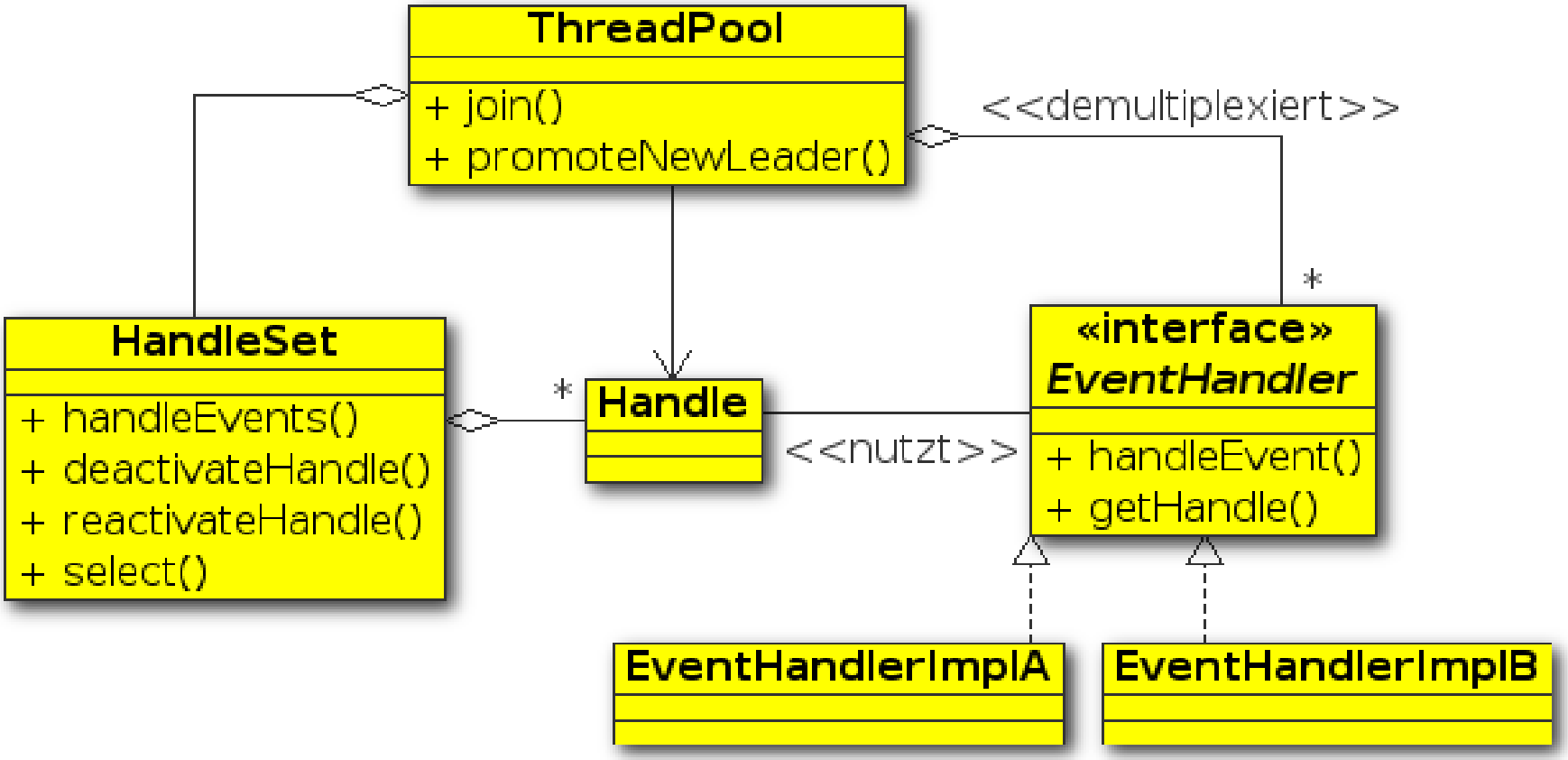
Handle Set
: Menge von Handles

Event Handler
: definiert Schnittstelle zur Verarbeitung (Hook-Methoden) von Ereignissen, die bei einem Handle eingehen

Thread-Pool
: Menge von Threads, enthält Synchronisationsmechismus

Leader/Followers

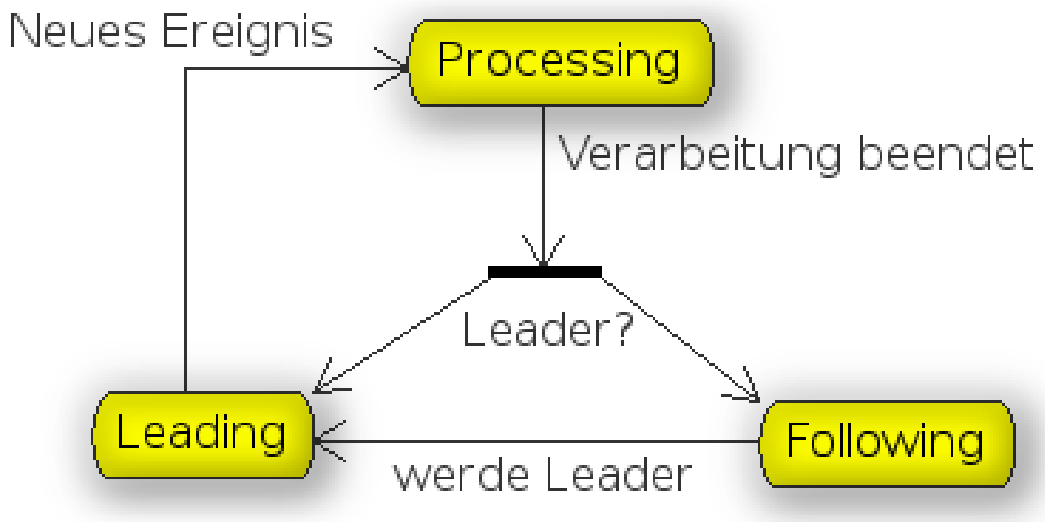
Struktur



Leader/Followers

Verhalten

Zustände eines Threads



Leader/Followers

Vor- und Nachteile

- Performanz
- einfache Programmierung

Leader/Followers

Vor- und Nachteile

- Performanz
- einfache Programmierung

- Implementierungskomplexität
- fehlende Flexibilität
- Engpässe bei Netzwerkein-/ausgabe

Zum Schluss ...

Zusammenfassung

Active Object

Active Object

- entkoppelt Aufruf einer Methode von deren Ausführung
- ermöglicht dadurch nebenläufiges Ausführen von Objekten
- Zugriff wird vereinfacht

Zusammenfassung

Monitor Object

Monitor Object

- synchronisiert nebenläufigen Zugriff auf Methoden eines Objekts
- garantiert, dass immer nur eine Methode ausgeführt wird
- Objekt kann Reihenfolge der Ausführung selbst bestimmen

Zusammenfassung

Leader/Followers

- Architekturmuster
- spezifiziert Nebenläufigkeitsarchitektur
- Threads wechseln sich selbstständig beim Zugriff aus Ereignisquellen ab
- eintreffende Ereignisse werden entdeckt, entmultiplexiert, verteilt und verarbeitet

Quellen

Schmidt, D., M. Stal, H. Rohnert und F. Buschmann: *Pattern-orientierte Software-Architektur*. dpunkt.verlag, 2002.

Abbildungen

- *Sprintlauf*
  `http://www.flickr.com/photos/texaseagle/3335901360/`
- *Boxer*
  `http://www.flickr.com/photos/24443965@N08/2649932391/`
- *Kabelsalat*
  `http://www.flickr.com/photos/jpconstantineau/2101768154/`
- *Noch Fragen?*
  `http://www.flickr.com/photos/bettina-braun/277386361/`

Vielen Dank!