Sommersemester 2021

Wolfgang Berger

Software Paradigmen

Behavioral Patterns

Verhaltensmuster

Verhaltensmuster

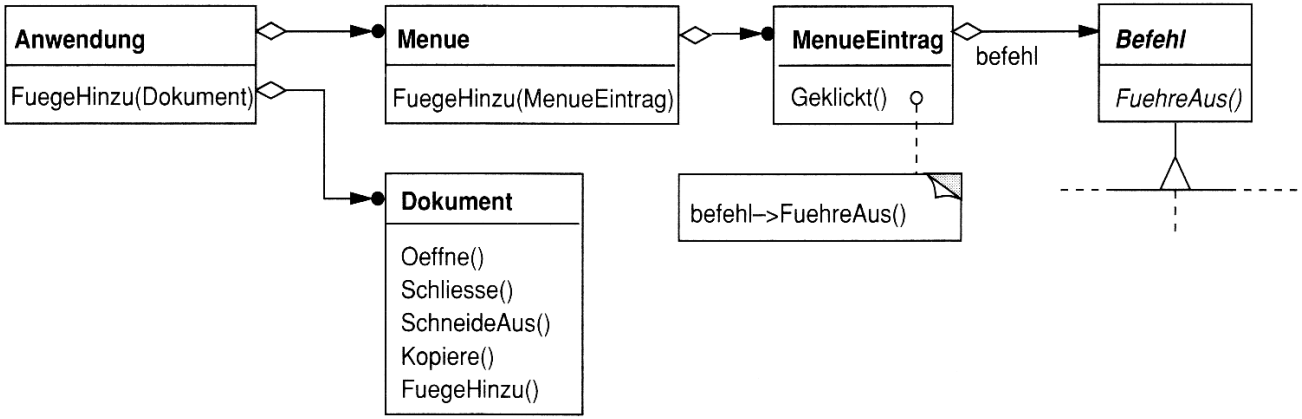
- Klassenbasierte
  - Interpreter
  - Template Method
- Objektbasierte
  - Observer
  - State
  - Command
  - Visitor
  - Iterator
  - Strategy
  - Mediator
  - Chain of Responsibility

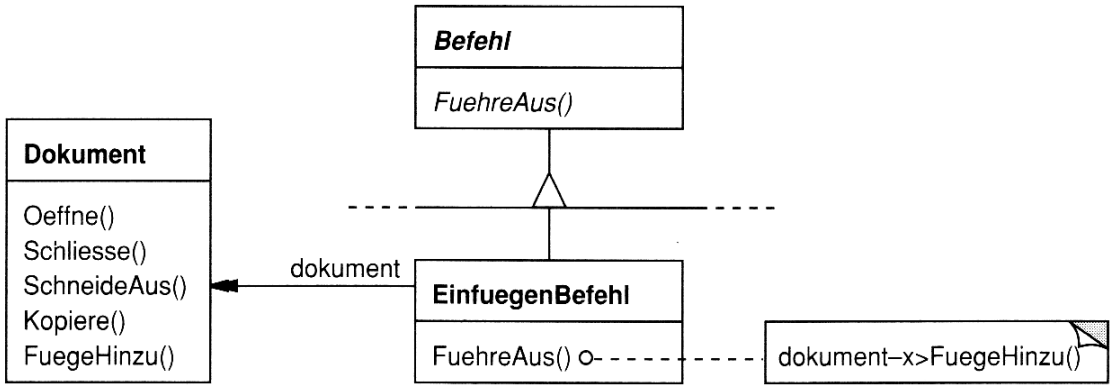
Command

- Zweck
  - Command, Action, Transaction
  - Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Queue zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.

Command

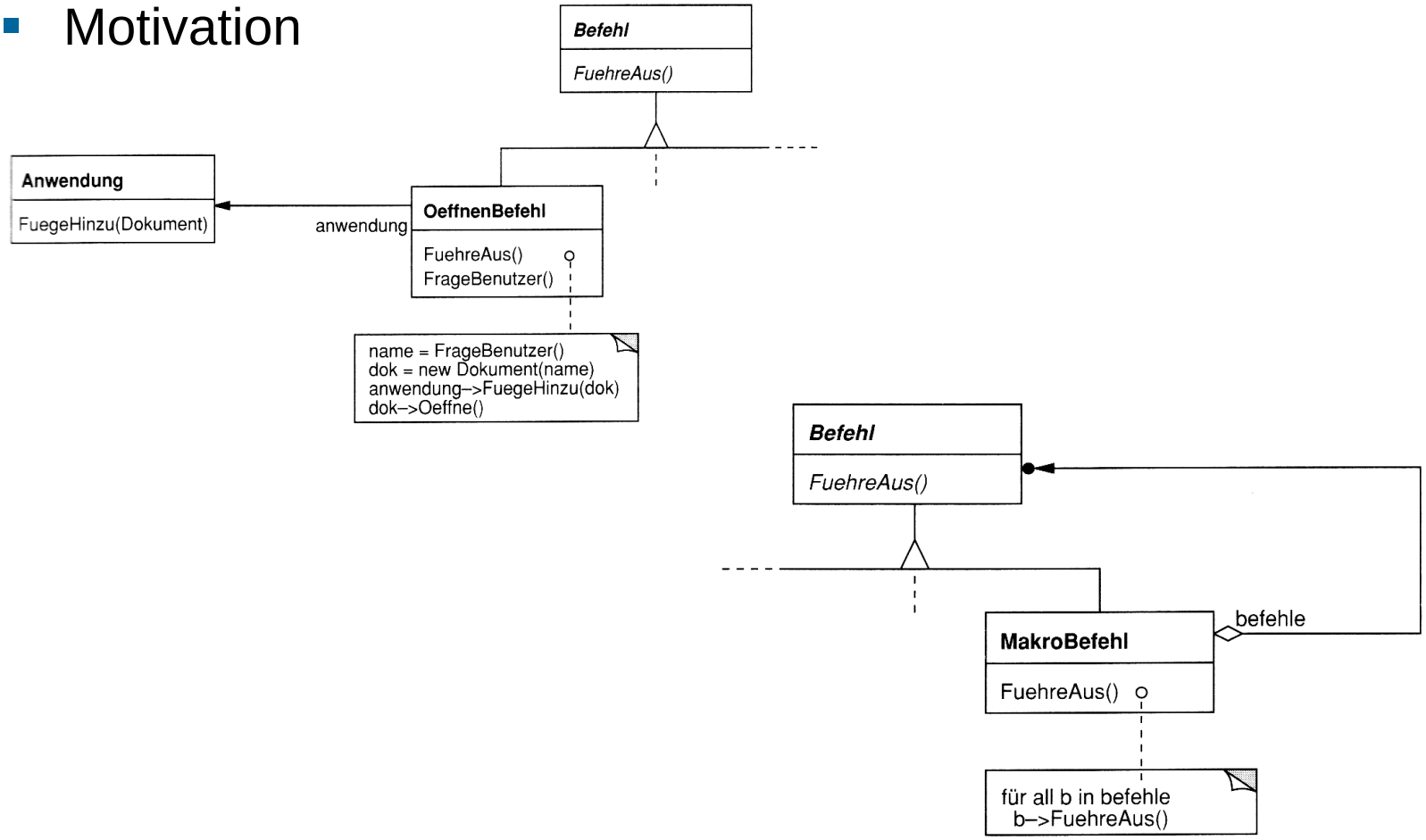
- Motivation





Command

- Motivation

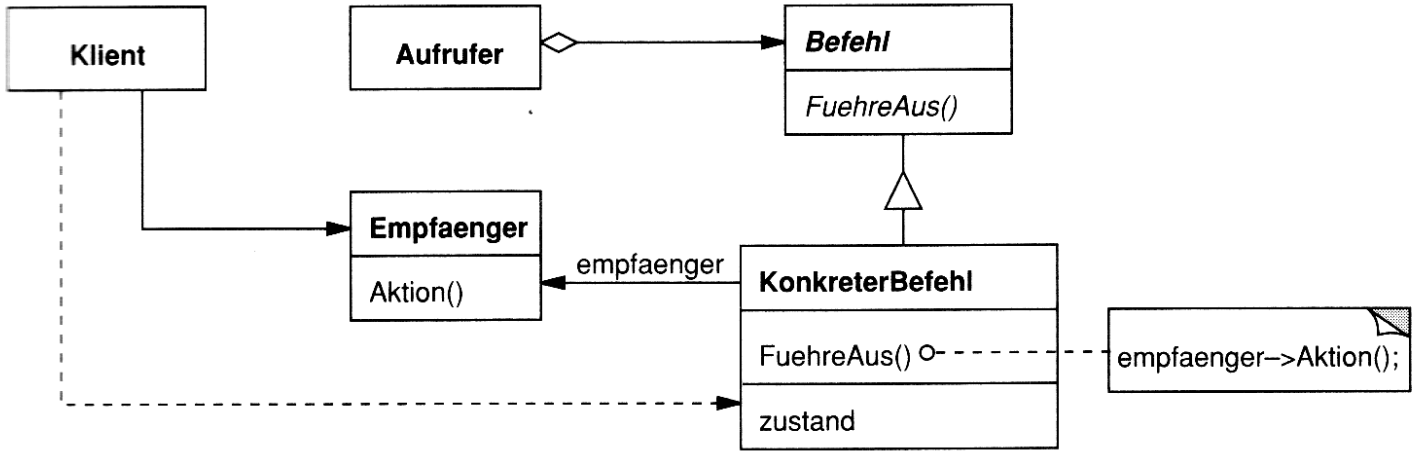


Command

- Anwendbarkeit
  - Objektorientierter Ersatz für Callbacks
  - Befehle zu unterschiedlichen Zeitpunkten und Orten aus der Anwendung spezifizieren, aufreihen und ausführen
  - Befehlsgeschichte mit Do-Undo Funktionalität
  - Mitprotokollieren von Änderungen
  - Strukturierung von komplexen Operationen, die aus primitiven Operationen bestehen.
  - Transaktionen – kapseln eine Menge von Datenänderungen.

Command

- Struktur



Command

- Teilnehmer
  - Befehl
    - Deklariert eine Schnittstelle zum Ausführen einer Operation
  - Konkreter Befehl
    - Definiert die Anbindung eines Empfängers an eine Aktion
    - Implementiert FuehreAus() durch Aufrufen der entsprechenden Operation(en) beim Empfänger
  - Klient
    - Erzeugt ein KonkreterBefehl-Objekt und übergibt ihm den Empfänger

Command

- Teilnehmer
  - Auslöser
    - Befiehlt dem Befehlsobjekt, die Anfrage auszuführen
  - Empfänger
    - Weiß, wie die an die Ausführung einer Anfrage gebundenen Operationen auszuführen sind. Jede Klasse kann ein Empfänger sein.

Command

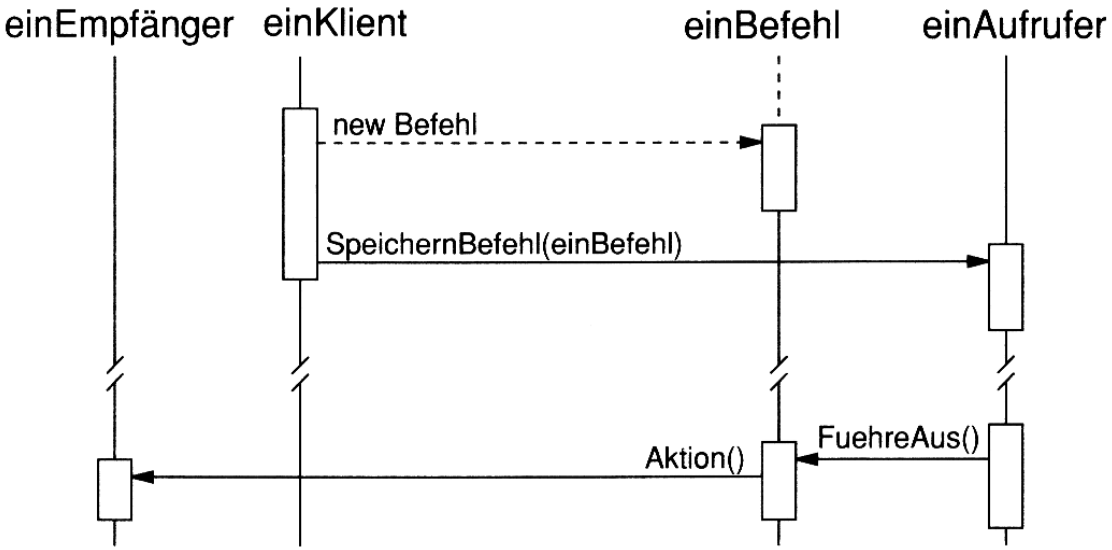
- Interaktionen
  - Der Klient erzeugt ein Befehlsobjekt einer konkreten Befehlsklasse und bestimmt ihren Empfänger
  - Ein Auslöser (z.B. ein Button oder ein Menüpunkt) speichert das Befehlsobjekt der konkreten Klasse.
  - Der Auslöser löst eine Anfrage aus, indem er die FuehreAus() Operation des Befehlsobjekts aufruft.

Command

- Interaktionen
  - Wenn Befehle rückgängig gemacht werden können, speichert das Befehlsobjekt vor dem Ausführen des Befehls den Zustand des Empfängers, um ihn später wiederherstellen zu können.
  - Das konkrete Befehlsobjekt ruft Operationen auf seinem Empfängerobjekt auf und setzt die Anfrage um.

Command

- Interaktionen



Command

- Konsequenzen
  - Das Befehlsmuster entkoppelt bei Änderungen den Klienten vom Empfänger.
  - Befehlsobjekte können manipuliert und erweitert werden wie jedes andere Objekt auch.
  - In einem Kompositum-Muster können Befehle auch zu Makros kombiniert werden.
  - Es ist einfach neue Befehlsobjekte hinzuzufügen, weil die bestehenden Befehlsobjekte nicht verändert werden müssen.

Command

- Implementierung
  - Intelligenz von Befehlsobjekten
    - Umsetzung der Änderungen im Befehl selbst oder im Empfänger?
    - Besser: im Befehl selbst um unabhängig vom Empfänger zu sein. Bei der Implementierung der Empfänger muss nicht auf das Befehlsmuster Rücksicht genommen werden.

Command

- Implementierung
  - Unterstützung von Do/Undo
    - Befehl muss sich den Zustand des Empfängers merken um eine Änderung wieder rückgängig machen zu können.
    - Befehlshistorie = Liste, die durchgeführte Befehle enthält
    - Befehlsobjekte, die in die Historie wandern, müssen kopiert werden, weil Ursprungsobjekt eventuell noch andere Abfragen abarbeiten muss.
    - Objekt Versionierung

Command

- Implementierung
  - Vermeiden von Fehlern im Undo-Prozess
    - durch mehrfaches Do-Undo (z.B. Mathematische Rundungsfehler)
    - Semantische Probleme beim Löschen – Ändern durch Befehle

Command

- Übung 4!