# Delegierende Programmierung als höheres Paradigma

## Ausgangsfrage

Wenn es neben imperativer und deklarativer Programmierung ein höheres, delegierendes Programmierparadigma mit Delegaten gäbe, in dem man Funktionen beziehungsweise Delegaten addieren, subtrahieren und multiplizieren kann, unter Einbeziehung von Arithmetik und Topologie:

> Zeige, dass dann manche Funktionsaufrufe mit Delegaten noch einmal einfacher sind als mit Haskell als deklarativ-funktionaler Programmiersprache und noch mehr als mit imperativer, prozeduraler Programmierung.

## 1. Vergleich der Paradigmen

### 1.1 Imperativ / prozedural

Imperative Programmierung beschreibt Programme primär als Sequenzen von Befehlen mit explizitem Kontrollfluss:

- Reihenfolge ist semantisch wesentlich,
- Zustand ist explizit,
- Kombination von Verhalten erfolgt durch Kontrollstrukturen.

Funktionsaufrufe sind an Ablauf, Variablen und Bedingungen gebunden. Die semantische Bedeutung ist über viele Codezeilen verteilt.

### 1.2 Deklarativ / funktional (z. B. Haskell)

Funktionale Programmierung beschreibt Programme als Transformationen von Werten:

$$f : A \to B$$

Kombination erfolgt durch:

- Funktionskomposition,
- Applicatives,
- Monaden.

Dies reduziert expliziten Zustand, führt jedoch zu:

- geschachtelten Abstraktionen,
- komplexen Typkonstruktionen,
- linear-sequentieller Semantik von Effekten.

## 2. Einführung des delegierenden Paradigmas

Das delegierende Paradigma hebt die Abstraktionsebene erneut an.

Ein Delegat ist kein bloßer Funktionswert, sondern ein *Bedeutungs- oder Wirkungsobjekt.*

$$D := \langle \text{Typ}, \text{Wirkung}, \text{Kontext} \rangle$$

Delegaten sind:

- erstklassig,
- algebraisch kombinierbar,
- topologisch strukturiert,
- ausführbar.

## 3. Algebraische Kombination von Delegaten

Im delegierenden Paradigma existieren wohldefinierte Operationen:

$$D_1 + D_2 \quad D_1 - D_2 \quad D_1 \cdot D_2$$

Diese Operationen wirken nicht auf Rückgabewerte, sondern auf die Bedeutung der Delegaten selbst.

Interpretation:

- Addition: Überlagerung von Wirkungen,
- Subtraktion: Ausschluss oder Maskierung,
- Multiplikation: Verschränkung oder Abhängigkeit.

Damit wird ein gesamter Verarbeitungsablauf zu einer einzigen algebraischen Formel.

## 4. Vergleich an einem Funktionsaufruf

Angenommen, ein Wert soll:

- validiert,
- geloggt,
- transformiert,
- eingeschränkt

werden.

### 4.1 Imperativ

Die Bedeutung verteilt sich auf:

- mehrere Kontrollstrukturen,
- Bedingungen,
- temporäre Variablen.

Entfernung oder Hinzufügung eines Aspekts erfordert strukturelle Codeänderungen.

### 4.2 Funktional (Haskell)

Die Bedeutung wird durch Monadenstapel oder Funktionskomposition ausgedrückt:

- semantisch dichter als imperativ,
- aber typologisch und syntaktisch komplex.

Kombination ist möglich, aber nicht algebraisch transparent.

### 4.3 Delegierend

Der gesamte Ablauf wird als Delegatenkombination geschrieben:

$$D = (V + L + T) \cdot R$$

Der Aufruf reduziert sich auf:

$$D(x)$$

Hinzufügen oder Entfernen eines Aspekts geschieht lokal:

$$D - L, \quad D + 2T$$

Keine Kontrollstruktur, keine Monaden, keine Reihenfolgeannahmen auf Quelltextebene.

## 5. Rolle der Topologie

Delegaten sind nicht nur algebraische Objekte, sondern liegen in einem Wirkungsraum mit topologischer Struktur.

- Nähe zwischen Delegaten entspricht ähnlicher Wirkung,
- Glättung ersetzt harte Kontrollentscheidungen,
- Randbedingungen ersetzen explizite Verzweigungen.

Damit wird Reihenfolge durch räumliche Struktur ersetzt.

## 6. Formale Überlegenheit

Für jede imperative Prozedur $P$ existiert eine funktionale Darstellung $F$ mit höherem Abstraktionsgrad.

Für jede funktionale Darstellung $F$ existiert eine delegierende Darstellung $D$ mit geringerer kombinatorischer Komplexität:

$$\text{AST-Tiefe}(D) < \text{AST-Tiefe}(F) < \text{AST-Tiefe}(P)$$

Der Grund ist, dass Delegaten zusätzliche algebraische und topologische Struktur tragen, die Funktionen nicht besitzen.

## 7. Schlussfolgerung

Delegierende Programmierung arbeitet auf einer höheren Ebene als imperative und deklarativ-funktionale Programmierung.

Sie kombiniert:

- Ausführbarkeit,
- Algebra,
- Topologie,
- Bedeutung.

| Manche Funktionsaufrufe sind delegierend strikter einfacher als funktional, und funktional einfacher als impe |
|---|