Inhaltsverzeichnis

1 System, Konsistenz, Lösungsstrategien

1.1 Grundlagenzeugs

1.1.1 Begriffsdefinition System

- abgegrenzter Bereich der objektiven Realität - Umwelt / Umgebung gehört nicht zum System - Systemrand grenzt System von der Umgebung ab
- mathematische Sicht: Endlicher Automat - Endliche Menge von Zuständen - Endliches Alphabet - Übergangsfunktion von Zustand x Alphabet -> Zustand - Startzustand - endliche Menge von Endzuständen

1.1.2 Begriffsdefinition Konsistenz im Zusammenhang mit dem System

- System muss als DFA modelliert sein - Durchführung einer Aktion ist eine Übergangsrelation - Übergangsrelation kann folge von atomaren Transaktionen sein

1.1.3 Atomarität

- Folge von Anweisungen - Alles-Oder-Nichts Prinzip

1.2 Lösungsstrategien

1.2.1 Instrumente für Sicherstellung von Konsistenz in Monolithischen Anwendungen

- Definieren von Transaktionen - Folge von Anweisungen, die zusammen ausgeführt werden müssen - Commit oder Rollback - Unterstützung von Lokalen Transaktionen durch Datenbank-Transaktionen

1.2.2 Instrumente für Sicherstellung von Konsistenz in Verteilten Systemen

- Transaktion beinhaltet Aktion, die eine Abhängigkeit aufruft (zB Aufruf einer Http-Schnittstelle) - Zentrales Problem: Wie stelle ich sicher, dass ein Aufruf geklappt hat? Wie gehe ich vor, wenn eine Aktion einer Transaktion nicht geklappt hat? - 2 Phasen Commit als verteilte Umsetzung des Transaktionsvorgehens - Beschreibung - Nachteile: sehr hohe Chattines, sehr langsam, blockierend, geringer Throughput, komplexe Implementierung

1.2.3 Saga-Pattern

- Fehlerbehandlungsstrategie für monolithische und verteilte Systeme - Auflösen der atomarität der Transaktionen in einzelne lokale Transaktionen T - Definieren von Kompensationsaktionen C - jedes T hat ein C - sequentielle Ausführung der Ts - schlägt ein T fehl, kann ein entsprechendes C ausgeführt werden

2 Saga Pattern

2.1 Grundprinzipien

- Saga ist erfolgreich, wenn alle Ts erfolgreich ausgeführt werden -> Übergang von: Ausgangszustand -> Ausführung T1 -> Ausführung T2 -> Endzustand - Saga ist fehlgeschlagen, wenn ein T fehlschlägt - Ausführung der Cs - Systemzustand ist danach immernoch konsistent - Backward Recovery: Alle Ts, die ausgeführt wurden, werden durch Ausführung des entsprechenden Cs gerollbackt - Forward Recovery: Einführung von Save Points zwischen den Ausführungen (zB T1-T5, Checkpoint nach T2 und Checkpoint nach T3) - Fehler in T3 führt zu Rollback bis letztem Save Point S1 (entspricht Ausführung von C3, Zustand nach Ausführung nach T1 und T2) - Wiederaufnehmen der Saga: Ausführung von T3 - T5 - Kompletter Rollback, falls es nicht geht (Backward Recovery)

2.1.1 Ts und Cs

2.1.2 Forward Recovery

2.1.3 Backward Recovery

2.1.4 Saga Execution Component

2.1.5 Transaktionslog

2.2 Orchestration und Choreographie

2.3 Asynchronität

3 Design

3.1 Design Choreografie

3.2 Design Orchestration

4 Implementierung

Abbildungsverzeichnis