Eine Entwicklungsplattform für die architekturorientierte Programmierung

Peter Tabeling

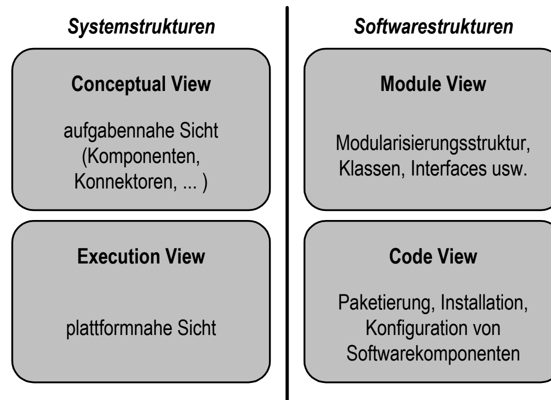
HASSO–PLATTNER-INSTITUT
für Softwaresystemtechnik an der Universität Potsdam

Überblick

- **Hintergrund und Motivation**
- **Grundelemente des Programmiermodells**
- **Praktische Umsetzung**
- **Abschließende Bemerkungen und Ausblick**

Architekturelle Sichten – „Siemens Four Views“

- **Die Beschreibung komplexer Systeme erfordert die Unterscheidung verschiedener architektureller Sichten (z.B. nach Hofmeister, Nord, Soni)**



Verteilungsaspekte

- **„Große“ Systeme sind i.d.R. verteilt**
- **Inkonsistenz von Daten ist inherántes Merkmal**
- **Inkonsistenz gemeinsam genutzter Daten**
  - Zugriffskonflikte bei nebenläufig agierenden Komponenten
  - Bedarf nach Koordination der Zugriffe
  - Lösungen: Sperren, Transaktionen, ...
- **Inkonsistenz verteilter Daten**
  - (echte) Nebenläufigkeit, nicht vernachlässigbare Laufzeiten
  - Beobachtung des Zustandes nur “lokal” möglich
  - Lösung: “Snapshots” – kausal konsistente Sicht

Hintergrund und Motivation

Idee der „architekturorientierten Programmierung“

- **Unterstützung der architekturellen Sichten im Programmiermodell**
  - Sprachelemente zur Beschreibung architektureller Systemstrukturen (Conceptual View bis Execution View)
  - Mittel zur Erstellung der Softwarestrukturen (Module View und Code View)
- **Berücksichtigung von Verteilungsaspekten**
  - Unterstützung verschiedener Kommunikationsszenarien (synchron, asynchron, Broadcast, One-Way, …)
  - Annahme der Nichtbeobachtbarkeit des Gesamtzustandes
  - (In-) Konsistenzmodelle und transaktionale Verarbeitung als integrale Bestandteile des Programmiermodells
- **Nicht objektorientiert!**

Programmiermodell

Basis: Fundamental Modeling Concepts (FMC)

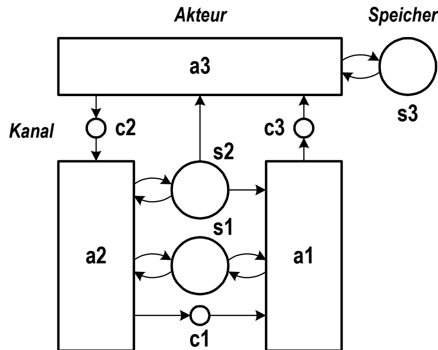
FMC unterscheidet drei Systemstrukturen

- **Aufbaustruktur des Systems aus interagierenden Komponenten**
  - **aktive** Komponenten: **Akteure**
  - **passive** Komponenten: **Speicher** und **Kanäle** → sind **abstrakte Orte**
- **Ablaufstruktur, d.h. Verhalten des Systems**
  - **Operationen**: elementare Aktivitäten von **Akteuren**
  - **Ereignisse**: elementare Auswirkungen von Operationen auf **Orten**
- **Wertestruktur**
  - **Werte**: Informationen auf Orten, können strukturiert sein

Ergänzende Konzepte für die Erfassung der Softwarestrukturen

- **Module View: Definitions, Modules**
- **Code View: Ressources, Bundles**

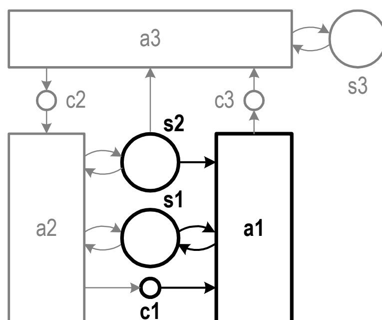
Beispiel für Aufbaustruktur



Beispiel für Operation – Zugriffe auf Orte

- **s1, s2** enthalten zweidimensionale Vektoren: **s1 = (2,4); s2 = (2,1)**
- Nachricht **„add“** über Kanal **c1** stößt Operation an: **<u>s1 := s1 + s2</u>**

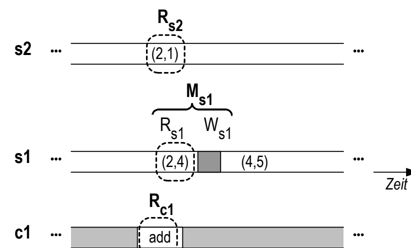
→ ***Für jeden betroffenen Ort ergibt sich ein <u>Zugriff</u>:***
lesend, **schreibend** oder **modifizierend** (kombiniert Lesen und Schreiben)



<u>hier:</u>

zwei Lesezugriffe: R_{c1}, R_{s2}

ein modifizierender Zugriff: M_{s1} (R_{s1},W_{s1})



Verteilungsaspekte – Integration von Konsistenzannahmen

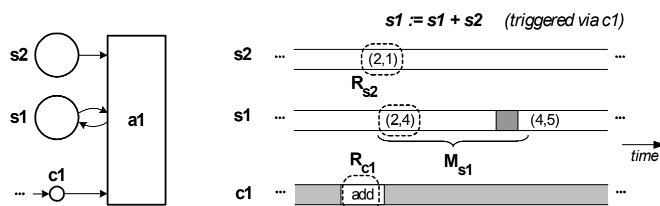
- **Bei der Durchführung einer Operation**
  - liest der durchführende Akteur Werte von mehreren Orten
  - verarbeitet diese Werte und
  - schreibt das Ergebnis auf den Zielort der Operation

  → ***Das Operationsergebnis beruht auf einer Beobachtung***

- **Grundsätzliche Konsistenzannahmen:**
  - ***Temporale Konsistenz***
    Eine Folge von Lesezugriffen auf einem Ort liefert eine temporal konsistente Wertefolge
  - ***Kausale Konsistenz***
    Das Lesen mehrerer Orte im Rahmen einer Operation liefert eine kausal konsistente Wertemenge

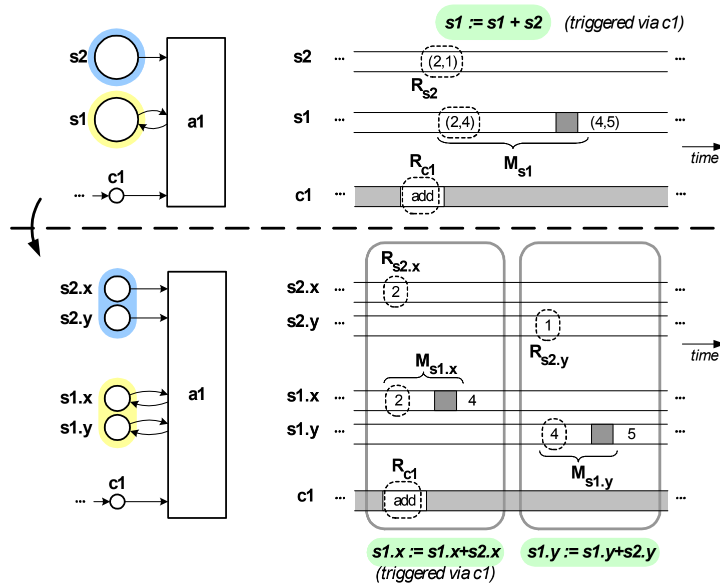
  → ***Keine Annahme eines beobachtbaren Gesamtzustandes!***

Zugriffsabbildung bei Implementierung von Orten bzw. Operationen (1)



Programmiermodell

Zugriffsabbildung bei Implementierung von Orten bzw. Operationen (2)



Programmiermodell

→ ***Transaktionen ergeben sich implizit aus der Abbildung von Operationen und Orten!***

höheres Modell		tieferes Modell
• **Ein Zugriff**	→	**transaktionale** Zugriffsmenge
• **Temporale Konsistenz**	→	**Atomicity, Consistency, Isolation**
• (Forderung: Fehlertoleranz)	→	(Durability)

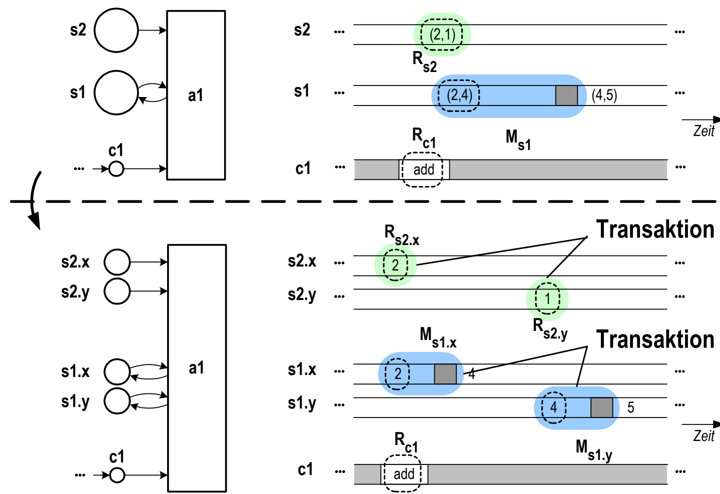
"Ableitung" verschiedener Transaktionstypen:

• **Eine Operation wird aufgeteilt**	→	**Flache Transaktion**
• **Eine Ort wird aufgeteilt**	→	**Verteilte Transaktion**
• **Mehrstufige Abbildung**	→	**Geschachtelte Transaktion**

→ ***Bedarf nach Snapshots ergibt sich aus der kausalen Konsistenz beim Lesen mehrerer Orte***

Programmiermodell

Im Beispiel:



Praktische Umsetzung

Laufzeit- und Entwicklungsplattform STAGE

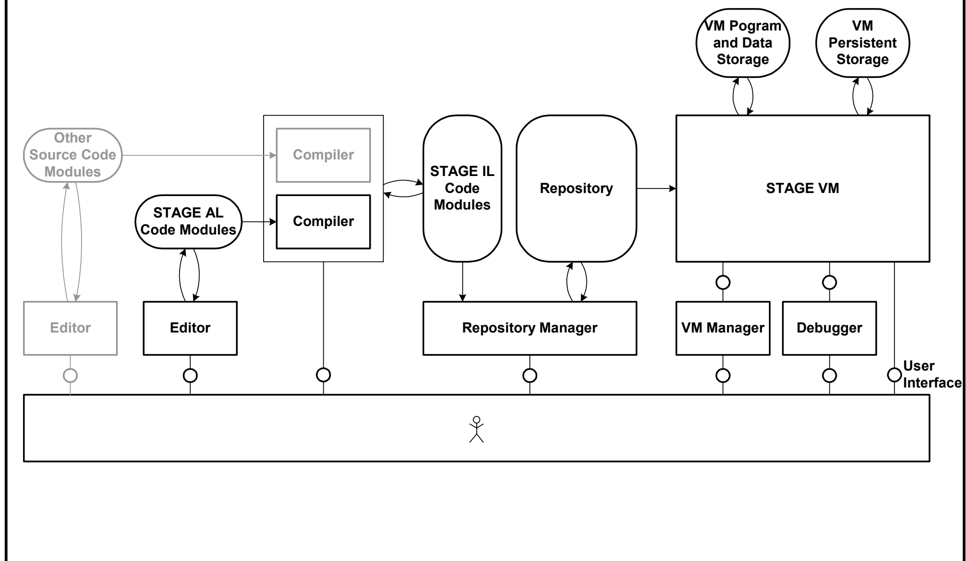
- **„Architekturorientierte“ Programmierung**
  - auf Basis des erweiterten FMC-Metamodells
  - Programme beschreiben Systemmodelle von Conceptual View bis Execution View
  - Strukturierungsmittel für Module View und Code View
- **Beschreibung von**
  - Akteurstypen, Ortstypen, Operationstypen, Aktivitätstypen, Prädikatstypen
  - Implementierungsbeziehungen
- **Übersetzung in Zwischensprache (STAGE IL)**
- **Transparent bzgl. der physikalischen Verteilung**
- **Verteilte virtuelle Maschine**
- **Implizite Transaktionen und Snapshots werden erkannt und durchgeführt**

→ ***kein Bedarf nach expliziten Synchronisationsmechanismen im Programmcode!***

Praktische Umsetzung

Peter Tabeling, 2.12.04

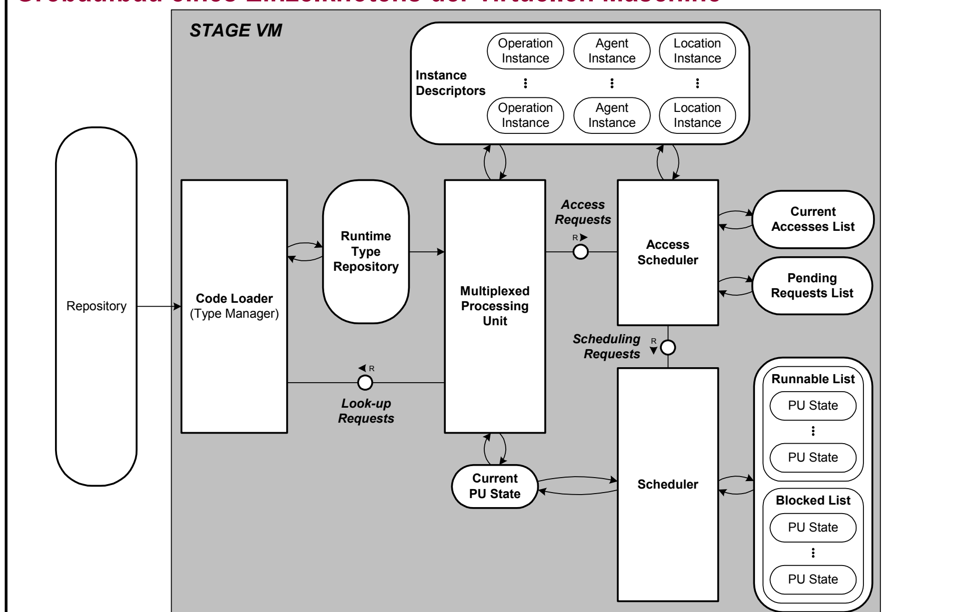
Überblick STAGE-Plattform



Praktische Umsetzung

Peter Tabeling, 2.12.04

Grobaufbau eines Einzelknotens der virtuellen Maschine



Abschließende Bemerkungen und Ausblick

Peter Tabeling, 2.12.04

Stand des Projektes

- Einfache, assemblerartige Hochsprache (STAGE AL)
- Compiler: Hochsprache → Zwischensprache (STAGE IL)
- Erste, noch nicht verteilte Version der virtuellen Maschine

Laufende Arbeiten

- Erweiterung, Anpassung der Sprache
- Erweiterung, Verteilung der virtuellen Maschine
- Debugger, Repository Manager

Ausblick

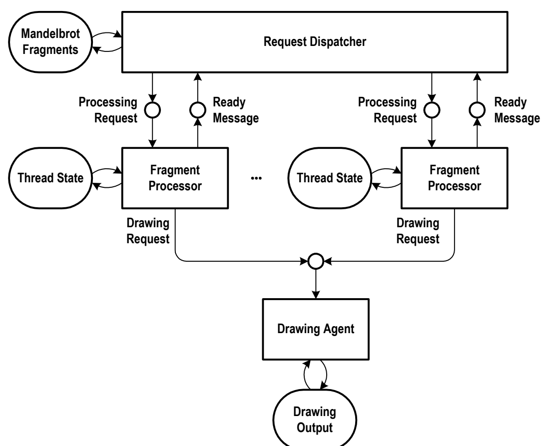
- Werkzeuge für die grafische Programmerstellung
- Ausbau der Plattform für die modellbasierte Entwicklung

Vielen Dank!

Beispiel
STAGE-Anwendung

Peter Tabeling, 2.12.04

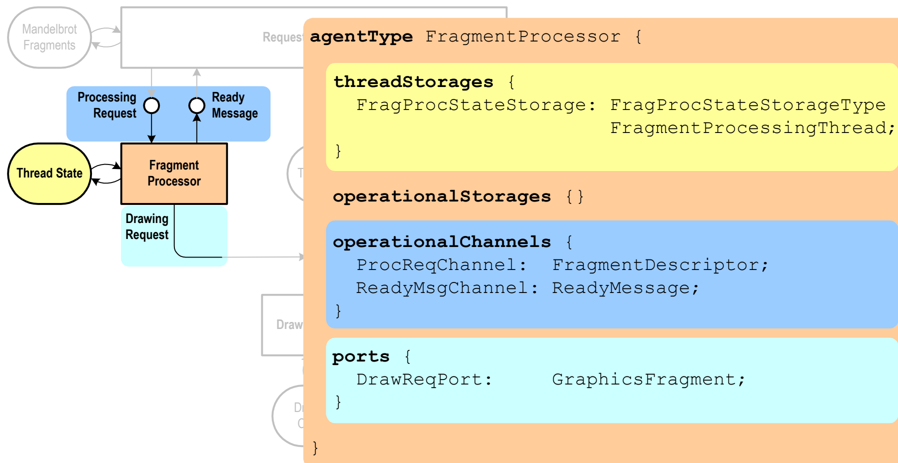
Beispiel: verteilter Mandelbrot-Berechner



FMC-basierte Entwicklungsplattform STAGE

Peter Tabeling, 2.12.04

Beispiel: verteilter Mandelbrot-Berechner, Fragment Processor



FMC-basierte Entwicklungsplattform STAGE

Peter Tabeling, 2.12.04

Beispiel: verteilter Mandelbrot-Berechner, Fragment Processor