

Reaktive Programmierung
Vorlesung 10 vom 24.06.14: The Actor Model

Christoph Lüth & Martin Ring

Universität Bremen

Sommersemester 2014

1 [17]

Fahrplan

- ▶ Teil I: Grundlegende Konzepte
- ▶ Teil II: Nebenläufigkeit
 - ▶ Futures and Promises
 - ▶ Reaktive Datenströme I
 - ▶ Reaktive Datenströme II
 - ▶ Funktional-Reaktive Programmierung
 - ▶ Das Aktorenmodell
 - ▶ Aktoren und Akka
- ▶ Teil III: Fortgeschrittene Konzepte

2 [17]

Das Aktorenmodell



- ▶ Eingeführt von Carl Hewitt, Peter Bishop und Richard Steiger (1973)
- ▶ Grundlage für nebenläufige Programmiersprachen und Frameworks. (Unter anderem Akka)
- ▶ Theoretisches Berechnungsmodell

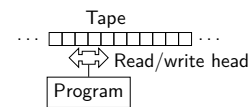
Warum ein weiteres Berechnungsmodell? Es gibt doch schon die Turingmaschine!

3 [17]

Die Turingmaschine



“the behavior of the computer at any moment is determined by the symbols which he [the computer] is observing, and his ‘state of mind’ at that moment.” — Alan Turing

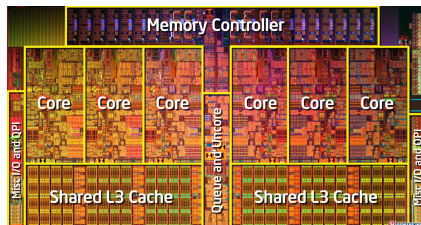


It is “absolutely impossible that anybody who understands the question [What is computation?] and knows Turing’s definition should decide for a different concept.” — Kurt Gödel



4 [17]

Die Realität



- ▶ $3\text{GHz} = 3'000'000'000\text{Hz} \implies$ Ein Takt = $3,333 \cdot 10^{-10}\text{s}$
- ▶ $c = 299'792'458 \frac{\text{m}}{\text{s}}$
- ▶ Maximaler Weg in einem Takt $< 0,1\text{m}$ (Physikalische Grenze)

5 [17]

Synchronisation

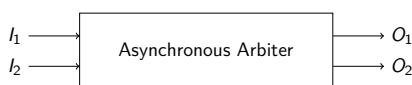


- ▶ Während auf ein Signal gewartet wird, kann nichts anderes gemacht werden
- ▶ Synchronisation ist nur in engen Grenzen praktikabel! (Flaschenhals)

6 [17]

Der Arbitrer

- ▶ Die Lösung: **Asynchrone Arbitrer**



- ▶ Wenn I_1 und I_2 fast ($\approx 2fs$) gleichzeitig aktiviert werden, wird entweder O_1 oder O_2 aktiviert.
- ▶ Physikalisch unmöglich in konstanter Zeit. Aber Wahrscheinlichkeit nimmt mit der Zeit exponentiell ab (**Unbounded indeterminacy**)
- ▶ Nicht modellierbar mit Turingmaschinen
- ▶ kommen aber in modernen Computern überall vor

7 [17]

Das Aktorenmodell

Quantum mechanics indicates that the notion of a universal description of the state of the world, shared by all observers, is a concept which is physically untenable, on experimental grounds. — Carlo Rovelli

- ▶ Frei nach der relationalen Quantenphysik

Drei Grundlagen

- ▶ Verarbeitung
- ▶ Speicher
- ▶ **Kommunikation**

- ▶ Die Turingmaschine ist ein Spezialfall des Aktorenmodells
- ▶ Ein **Aktorensystem** besteht aus **Aktoren** (Alles ist ein Akteur!)

8 [17]

Aktoren

- ▶ Ein Aktor verarbeitet Nachrichten (sequenziell) in FIFO Reihenfolge

Während ein Aktor eine Nachricht verarbeitet kann er

- ▶ neue Aktoren erzeugen
- ▶ Nachrichten an bekannte Aktor-Referenzen versenden
- ▶ festlegen wie die nächste Nachricht verarbeitet werden soll

- ▶ Aktor \neq (Thread | Task | Channel | ...)

9 [17]

Kommunikation

- ▶ Nachrichten sind unveränderliche Daten oder **Futures**
- ▶ Die Zustellung von Nachrichten passiert höchstens einmal (Best-effort)
- ▶ Wenn z.B. die Netzwerkverbindung abbricht, wird gewartet, bis der Versand wieder möglich ist
- ▶ Wenn aber z.B. der Computer direkt nach Versand der Nachricht explodiert (oder der Speicher voll läuft), kommt die Nachricht möglicherweise niemals an
- ▶ Über den Zeitpunkt des Empfangs kann keine Aussage getroffen werden (Unbounded indeterminacy)
- ▶ Über die Reihenfolge der empfangenen Nachrichten wird im Aktorenmodell keine Aussage gemacht (In vielen Implementierungen allerdings schon)
- ▶ Nachrichtenversand \neq (Queue | Lock | Channel | ...)

10 [17]

Identifikation

- ▶ Aktoren werden über **Identitäten** angesprochen

Aktoren kennen Identitäten

- ▶ aus einer empfangenen Nachricht
- ▶ aus der Vergangenheit
- ▶ von Aktoren die sie selbst erzeugen

- ▶ Nachrichten können weitergeleitet werden
- ▶ Eine Identität kann zu mehreren Aktoren gehören, die der Halter der Referenz äußerlich nicht unterscheiden kann
- ▶ Eindeutige Identifikation bei verteilten Systemen nur durch Authentisierungsverfahren möglich

11 [17]

Location Transparency

- ▶ Eine Aktoridentität kann irgendwo hin zeigen
 - ▶ Gleicher Thread
 - ▶ Gleicher Prozess
 - ▶ Gleicher CPU Kern
 - ▶ Gleiche CPU
 - ▶ Gleicher Rechner
 - ▶ Gleiches Rechenzentrum
 - ▶ Gleicher Ort
 - ▶ Gleiches Land
 - ▶ Gleicher Kontinent
 - ▶ Gleicher Planet
 - ▶ ...

12 [17]

Beispiel

13 [17]

Inkonsistenz

- ▶ Ein Aktorsystem hat keinen globalen Zustand (Pluralismus)
- ▶ Informationen in Aktoren sind global betrachtet **redundant**, **inkonsistent** oder **lokal**
- ▶ Konsistenz \neq Korrektheit
- ▶ Wo nötig müssen duplizierte Informationen konvergieren, wenn "**längere Zeit**" keine Ereignisse auftreten (**Eventual consistency**)

14 [17]

"Let it Crash!"

- ▶ Indeterminismus ist statisch kaum analysierbar
- ▶ **Unschärfe** beim Testen von verteilten Systemen
- ▶ Selbst wenn ein Programm fehlerfrei ist kann Hardware ausfallen
- ▶ Je verteilter ein System umso wahrscheinlicher geht etwas schief
- ▶ Deswegen:
 - ▶ Offensives Programmieren
 - ▶ Statt Fehler zu vermeiden, Fehler behandeln!
 - ▶ Teile des Programms kontrolliert abstürzen lassen und bei Bedarf neu starten



15 [17]

Das Aktorenmodell in der Praxis

- ▶ Erlang (Aktor-Sprache)
 - ▶ Ericsson - GPRS, UMTS
 - ▶ T-Mobile - SMS
 - ▶ WhatsApp (2 Millionen Nutzer pro Server)
 - ▶ Facebook Chat (100 Millionen simultane Nutzer)
 - ▶ Amazon SimpleDB
 - ▶ ...
- ▶ Akka (Scala Framework)
 - ▶ ca. 50 Millionen Nachrichten / Sekunde
 - ▶ ca. 2,5 Millionen Aktoren / GB Heap
 - ▶ Amazon, Cisco, Blizzard, LinkedIn, BBC, The Guardian, Atos, The Huffington Post, Ebay, Groupon, Credit Suisse, Gilt, KK, ...

16 [17]

Zusammenfassung

- ▶ Das Aktorenmodell beschreibt **Aktorensysteme**
- ▶ Aktorensysteme bestehen aus **Aktoren**
- ▶ Aktoren kommunizieren über **Nachrichten**
- ▶ Aktoren können überall liegen (**Location Transparency**)
- ▶ Inkonsistenzen können nicht vermieden werden: **Let it crash!**
- ▶ Vorteile: Einfaches Modell; keine Deadlocks/Race Conditions; Sehr schnell in Verteilten Systemen
- ▶ Nachteile: Informationen müssen dupliziert werden; Keine vollständige Implementierung