

S-BPM Groupware

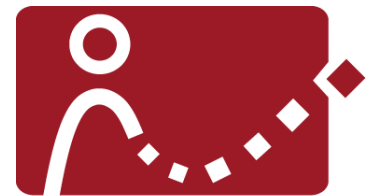
Internet-Praktikum TK SS12



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorstellung Akka

André Röder



Telecooperation Lab

- Grundprinzipien Akka
- Actors und dazugehörige Konzepte
- Messaging
- Logging
- Futures

- **Actors**
 - Einfache, high-level Abstraktion für Nebenläufigkeit (concurrency) und Parallelität (parallelism)
 - Asynchron, non-blocking und event-driven
- **Location Transparency**
 - Alle Interaktionen erfolgen über Messaging und sind vollkommen von der darunterliegenden Umgebung unabhängig
- Verwendung als library oder microkernel

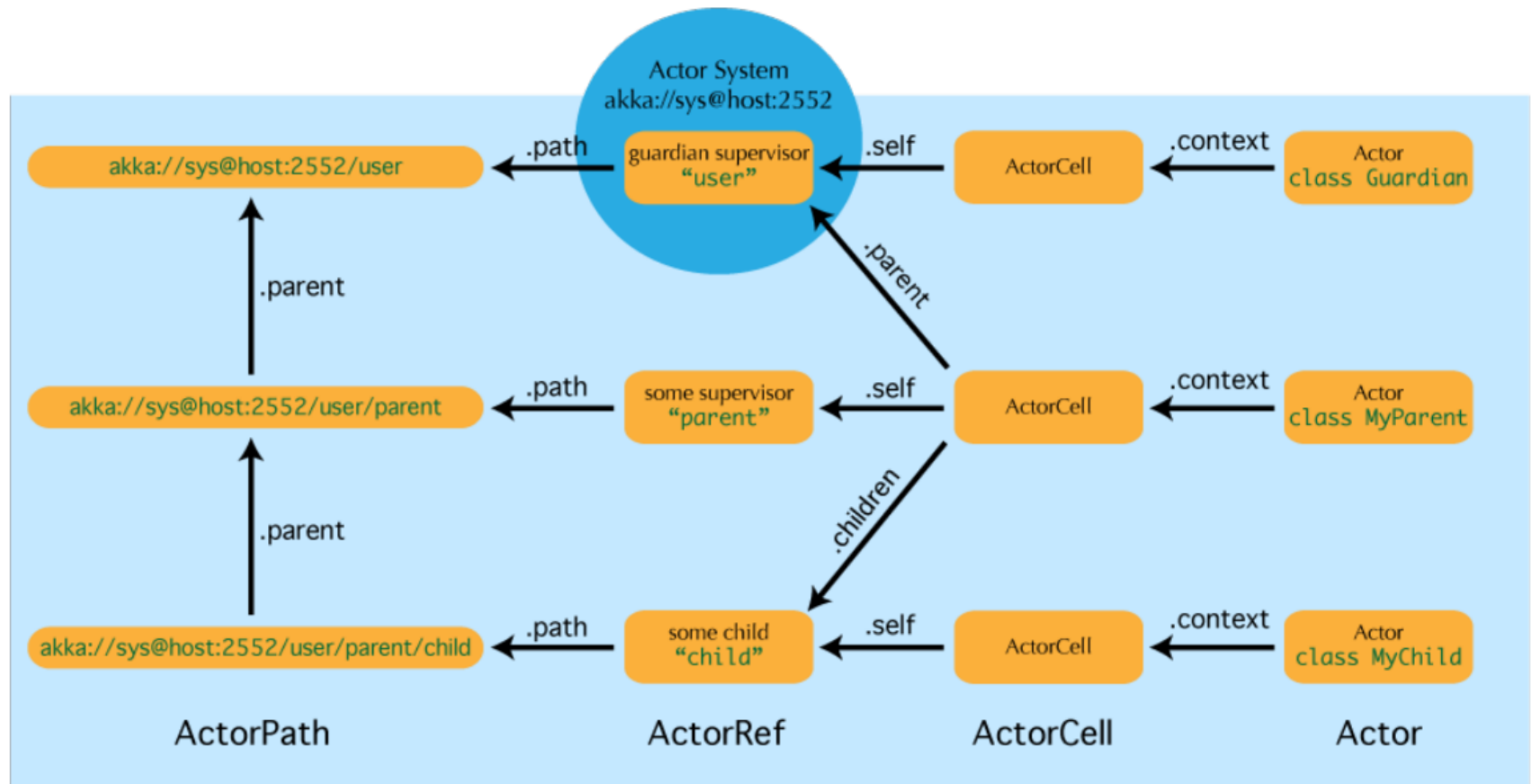
Actors

- Grundgedanke: Divide & Conquer – Tasks werden so lange ge- und verteilt bis ein Actor sie alleine lösen kann
- Actors sind Container (Objekte) für Zustand, Verhalten, Kinder, Mailbox und eine Supervisor Strategie
- Actors folgen, wie eine reale Organisation, einer Hierarchie – jeder Actor hat genau einen Supervisor

- **State & Behavior**
 - Der Zustand eines Actors ist nach außen abgeschirmt (wie ein Objekt)
 - Ein Neustart des Actors erzeugt ihn in seinem Ursprungsverhalten, in der Zwischenzeit veränderte Verhaltensweisen gehen verloren (become/unbecome)
- **Mailbox**
 - Jeder Actor hat eine Mailbox deren Nachrichten nach ihrer Sendezeit gequeued werden
 - Wenn eine Actor mehrere Nachrichten an einen anderen sendet, sind diese genau in der Reihenfolge in der Warteschlange
 - Senden mehrere Actoren Nachrichten an einen Actor, kann durch Threading nicht sichergestellt werden, welche Nachricht zuerst ankommt
 - Jeder Actor MUSS die nächste Nachricht verarbeiten, es gibt kein scannen der Queue nach passenden Nachrichten

- **Children**
 - Jeder Actor ist ein potentieller Supervisor und hat eine Liste seiner Children, welche dem ausgeführten Kontext zur Verfügung steht
 - Die möglichen Operationen sind Erzeugung oder Beendung von Children

Akka Hierarchie



Best Practice

Immer die ActorRef verarbeiten (self), nicht den Actor selbst (über this)

Code: Creating an Actor

```
import akka.actor.Actor
import akka.actor.Props
import akka.event.Logging

class MyActor extends Actor {
  val log = Logging(context.system, this)
  def receive = {
    case "test" => log.info("received test")
    case _      => log.info("received unknown message")
  }
}
```

```
object Main extends App {
  val system = ActorSystem("MySystem")
  val myActor = system.actorOf(Props[MyActor], name = "myactor")
}
```

Ein eigener Actor erweitert immer die Actor Klasse und muss die Funktion receive definieren

actorOf erzeugt eine Actor Instanz für das actorSystem. Rückgabe Typ ist actorRef

Hinweis

Actors werden automatisch gestartet. Dabei wird der Hook preStart() automatisch ausgeführt und kann überschrieben werden.

Code: Messaging



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
case object Tick
case object Get

class Counter extends Actor {
  var count = 0

  def receive = {
    case Tick => count += 1
    case Get  => sender ! count
  }
}

object AkkaProjectInScala extends App {
  val system = ActorSystem("AkkaProjectInScala")

  val counter = system.actorOf(Props[Counter])

  counter ! Tick
  counter ! Tick
  counter ! Tick

  implicit val timeout = Timeout(5 seconds)

  (counter ? Get) onSuccess {
    case count => println("Count is " + count)
  }

  system.shutdown()
}
```

Es gibt zwei Möglichkeiten zum senden:

- ! für tell (fire & forget)
- ? für ask (auf Antwort warten und reagieren)

Eine weitere Möglichkeit ist das weiterleiten mit **forward**

Future Callback

Best Practice

Immer tell benutzen wenn möglich, da ask aufwändiger (Performance)

Logging

```
import akka.event.Logging

class MyActor extends Actor {
  val log = Logging(context.system, this)
  override def preStart() = {
    log.debug("Starting")
  }
  override def preRestart(reason: Throwable, message: Option[Any]) {
    log.error(reason, "Restarting due to [{}] when processing [{}]",
      reason.getMessage, message.getOrElse(""))
  }
  def receive = {
    case "test" => log.info("Received test")
    case x      => log.warning("Received unknown message: {}", x)
  }
}
```

Log Level: debug

Log Level: error

Log Level: warning

- Logging wird asynchron über ein Event-Bus ausgeführt
- Darüber können Event Handler definiert werden, die weitere Aktionen auslösen

- Ein Future ist eine Struktur die einem den Zugriff auf ein zukünftiges (nebenläufige), möglicherweise eintretendes Ergebnis ermöglicht
- Wird häufig als Rückgabewert eines Asks (?) genutzt, dies kann auch außerhalb eines Actors eingesetzt werden

```
implicit val timeout = Timeout(3000)
val future = (processInstanceActor ? ReadSubject(id))
    .mapTo[SubjectAnswer]
val result = Await.result(future, timeout.duration)
```

Await ist blockierend und sollte daher nur in Ausnahmefällen genutzt werden

Future liefert immer Typ Any zurück, daher mapTo

Hinweis: um ask nutzen zu können benötigt man den folgenden import:

```
import akka.pattern.ask
```

Non-blocking Futures

- Listener auf Future Events sind: onComplete, onSuccess, onFailure
- Non-blocking Futures:

```
val f1 = ask(actor1, msg1)
val f2 = ask(actor2, msg2)

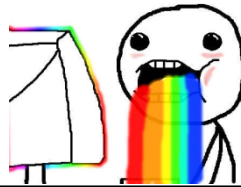
val f3 = for {
  a ← f1.mapTo[Int]
  b ← f2.mapTo[Int]
  c ← ask(actor3, (a + b)).mapTo[Int]
} yield c

val result = Await.result(f3, 1 second).asInstanceOf[Int]
```

Warnung

Futures werden ab Version 2.1
Scala Bestandteil und gehören
wohl nicht mehr zu Akka.

Wie installieren?



- Akka ist in Typesafe enthalten und Beispielprojekte können mit g8 ausgecheckt werden
 - z.B. g8 typesafehub/akka-first-tutorial-scala
- Vorsicht: Die Projekte können nicht in Eclipse importiert werden, sie müssen vorher umgewandelt werden
 - Es muss das Eclipse SBT Plugin ausgeführt werden um die Ordner in Eclipse Projekte umzuwandeln
 - Hierzu im Beispielprojektordner unter \project (z.B. akka-project-in-scala\project) die Datei **plugins.sbt** anlegen oder erweitern

```
// Comment to get more information during initialization
logLevel := Level.Warn

// The Typesafe repository
resolvers += "Typesafe repository" at "http://repo.typesafe.com/typesafe/releases/"

// Use the eclipse sbt plugin
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "2.0.0")
```



- Dann in den Hauptordner navigieren, **SBT** starten in der Konsole und einfach „**eclipse**“ ausführen
- Schließlich über Import als Projekt in Eclipse importieren