The Power of ~~LOVE~~... Actors

Bernhard Stöcker

Recogizer Group GmbH

bernhard.stoecker@recogizer.de

November 5, 2016

Overview

What are Actors?

What are Actors?

What are Actors?

Act-or: to act: **"to do something for a particular purpose or to solve a problem"** (From the Cambridge dictionary)

What are Actors?

Wikipedia:

- A mathematical model of concurrent computation
- Actors can hold and modify private state
- Affect each other through messages only
- In response to a message that it receives, an actor can:
  - Make local decisions
  - Create more actors
  - Send more messages
  - Respond to the incoming message

WhatsApp

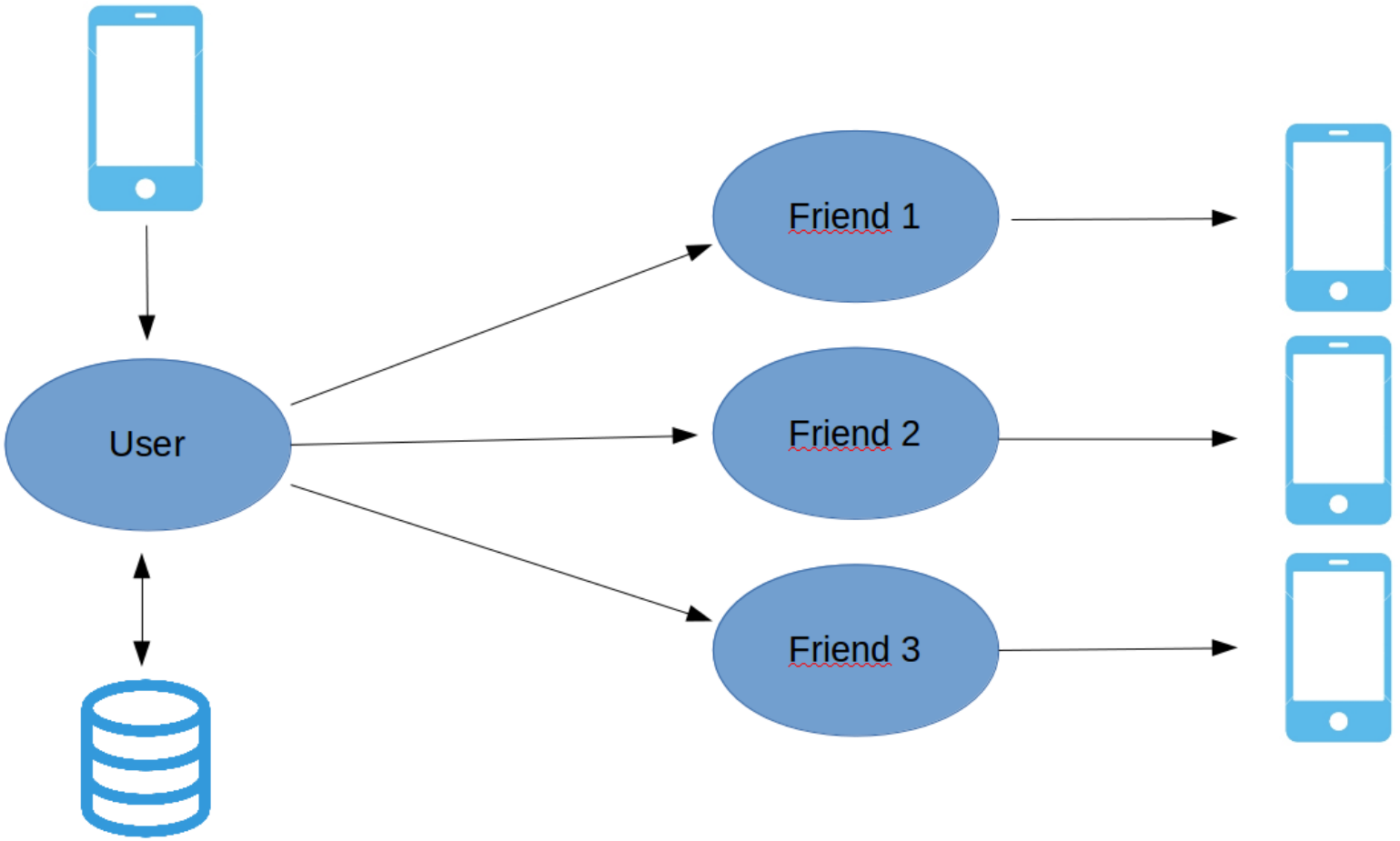
WhatsApp

WhatsApp???

WhatsApp

- Every user has an actor representing her
- When sending a message my actor sends a message to all related users
- The users receiving a message ensure the message is delivered.

WhatsApp



Actors in Elixir

Actors in Elixir

Actors in Elixir

```
bernhard@bernhards-thinkpad ~ $ iex
Erlang/OTP 19 [erts-8.1] [source-4cc2ce3] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-po

Interactive Elixir (1.3.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> spawn fn -> IO.puts("Hello World") end
Hello World
#PID<0.83.0>
iex(2)>
```

Actors in Elixir

```
bernhard@bernhards-thinkpad ~ $ iex
Erlang/OTP 19 [erts-8.1] [source-4cc2ce3] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-po

Interactive Elixir (1.3.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> func = fn ->
...(1)> receive do
...(1)> name -> IO.puts("Hello #{name}")
...(1)> end
...(1)> end
#Function<20.52032458/0 in :erl_eval.expr/5>
iex(2)> pid = spawn func
#PID<0.88.0>
iex(3)> send(pid, "Kira")
Hello Kira
"Kira"
iex(4)>
```

Actors in Elixir

```
iex(1)> defmodule Value do
...(1)> def current(x) do
...(1)> receive do
...(1)> add -> IO.puts(x + add); current(x + add)
...(1)> end
...(1)> end
...(1)> end
{:module, Value,
 <<70, 79, 82, 49, 0, 0, 5, 40, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 148
   131, 104, 2, 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 1
   95, 118, 49, 108, 0, 0, 0, 4, 104, 2, ...>>, {:current, 1}}
iex(2)> pid = spawn fn -> Value.current(0) end
#PID<0.95.0>
iex(3)> send pid, 17
17
17
iex(4)> send pid, 16
33
16
iex(5)> send pid, 9
42
9
iex(6)>
```

Actors in Elixir

```elixir
defmodule Stack do
  use GenServer

  # Callbacks

  def handle_call(:pop, _from, []) do
    {:reply, nil, []}
  end
  def handle_call(:pop, _from, [h | t]) do
    {:reply, h, t}
  end

  def handle_call(:top, _from, []) do
    {:reply, nil, []}
  end
  def handle_call(:top, _from, [h | t]) do
    {:reply, h, [h | t]}
  end

  def handle_cast({:push, item}, state) do
    {:noreply, [item | state]}
  end
end

```

Actors in Elixir

```
^Cbernhard@bernhards-thinkpad ~/Dokumente/Officetalk $ iex
Erlang/OTP 19 [erts-8.1] [source-4cc2ce3] [64-bit] [smp:4:4] [async-threads:

Interactive Elixir (1.3.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> c("stack.ex")
[Stack]
iex(2)> {:ok, pid} = GenServer.start_link(Stack, ["Hello", "World"])
{:ok, #PID<0.89.0>}
iex(3)> pid
#PID<0.89.0>
iex(4)> GenServer.call(pid, :pop)
"Hello"
iex(5)> GenServer.call(pid, :pop)
"World"
iex(6)> GenServer.call(pid, :top)
nil
iex(7)> GenServer.cast(pid, {:push, "Hello Kira"})
:ok
iex(8)> GenServer.call(pid, :top)
"Hello Kira"
iex(9)>
```

Actors in Scala

Actors in Scala

Actors in Scala

```scala
package example

import akka.actor.{ Props, Actor, Terminated }

final case class Hello(var name: String)

object Example {
  def props() :Props = Props(classOf[Example])
}

class Example extends Actor {
  def receive = {
    case Hello(name) => { println("Hello " + name) }
    case _ => println("Example received unknown message")
  }
}
```

Actors in Scala

```
package runtime

import akka.actor._
import example._

object Main extends App {

  val system = ActorSystem("KidsActorSystem")
  val exampleActor = system.actorOf(Example.props())

  exampleActor ! Hello("World")

  system.terminate
}
```

Actors in Scala

```
bernhard@bernhards-thinkpad ~/Dokumente/Officetalk/scala_
[info] Set current project to Actors (in build file:/home
[info] Compiling 1 Scala source to /home/bernhard/Dokumen
[info] Running runtime.Main
Hello World
[success] Total time: 4 s, completed 03.11.2016 16:53:15
bernhard@bernhards-thinkpad ~/Dokumente/Officetalk/scala_
```

Actors in Scala

```scala
package family

import akka.actor._

trait BaseParent extends Actor {
  def spawnChild(context: ActorContext) :ActorRef

  var child = respawnChild

  def receive = {
    case MeasureKidSize => child ! TellMeSize
    case FeedKid => child ! Feed
    case KillKid => child ! PoisonPill
    case KidSize(size) => println("The child is " + size + "cm tall!")
    case Terminated(childActor) => {
      println("Child actor died. Respawn!")
      child = respawnChild
    }
    case _ => println("Example received unknown message")
  }

  def respawnChild = {
    val childActor = spawnChild(context)
    context.watch(childActor)
    childActor
  }
}
```

Actors in Scala

```scala
package family

import akka.actor._

object Parent {
  def props() :Props = Props(classOf[Parent])
}

class Parent extends BaseParent {
  def spawnChild(context: ActorContext) = {
    context.system.actorOf(Child.props())
  }
}
```

Actors in Scala

```
package family

import akka.actor.{ Props, Actor }

object Child {
  def props() :Props = Props(classOf[Child])
}

class Child extends Actor {
  var currentSize = 55
  def receive = {
    case Feed => {
      currentSize += 1
      sender() ! KidSize(currentSize)
    }
    case TellMeSize => sender() ! KidSize(currentSize)
    case _ => println("Example received unknown message")
  }
}
```

Actors in Scala

```
package family

case object FeedKid
case object Feed
case object KillKid
final case class KidSize(val size: Int)
case object TellMeSize
case object MeasureKidSize
```

Actors in Scala

```
package runtime

import akka.actor._
import family._

object Main extends App {
  val system = ActorSystem("KidsActorSystem")
  val parentActor = system.actorOf(Parent.props())

  parentActor ! MeasureKidSize
  parentActor ! FeedKid
  parentActor ! FeedKid
  parentActor ! KillKid
  Thread.sleep(100)
  parentActor ! MeasureKidSize
  Thread.sleep(1000)
  system.terminate
}
```

The End