Programación Reactiva Actores

Paradigmas de la programaación
FaMAF-UNC 2021

Problema de la concurrencia (esp. distribuida)

condiciones de carrera en las componentes no declarativas (orientación a objetos, imperativo)

> *shared mutable state*
>
> *stateful objects whose state can be changed by multiple parts of your application, each running in their own thread*

Concurrencia

pocos locks - condiciones de carrera

demasiados locks - poca eficiencia

> *low-level synchronization constructs like locks and threads are very hard to reason about. As a consequence, it's very hard to get it right (race conditions to deadlocks or just strange behaviour)*

Solución:
Declaratividad

Algunas referencias

https://en.wikipedia.org/wiki/Reactive_programming

https://en.wikipedia.org/wiki/Actor_model

https://danielwestheide.com/blog/the-neophytes-guide-to-scala-part-14-the-actor-approach-to-concurrency/

https://doc.akka.io/docs/akka/2.5/guide/introduction.html

programación reactiva

Programación Reactiva

paradigma de programación declarativa

data streams: estáticos (arreglos) o dinámicos (generadores de eventos)

propagación del cambio: dependencia entre componentes

esto se dibuja como un data flow

¿Cómo es la propagación del cambio?

a := b + c

imperativo – a recibe el resultado de b + c en el momento en que se evalúa la expresión, si después cambian los valores de b o c, no tiene efecto en a

reactivo – el valor de a se actualiza automáticamente cada vez que cambian los valores de b o c, sin re-ejecutar nada

Sin re-evaluacion constante
Los procesos que no estan activos no consumen recursos

¿Cómo es la propagación del cambio?

a := b + c

imperativ

que se ev

de b o c, r

qué horror!!!
por qué haría algo así????

reactivo – el valor de a se actualiza automáticamente cada vez que cambian los valores de b o c, sin re-ejecutar nada

Ejemplos de uso

- descripción de hardware (Verilog) – propagación de los cambios en circuitos
- arquitectura model-view-controller (MVC): se propagan los cambios del modelo a la vista

es un paradigma

hay que pensar y escribir el programa con la concurrencia en mente

modelo de actores

Motivación

1973 – paper fundacional

1975 – semántica operacional

inspirado en la física (relatividad, cuántica)

> *motivated by the prospect of highly parallel computing machines consisting of dozens, hundreds, or even thousands of independent microprocessors, each with its own local memory and communications processor, communicating via a high-performance communications network*

Modelo de actores

modelo matemático de cálculo concurrente

actor: primitiva de cálculo concurrente

cuando un actor recibe un mensaje, **reacciona** con:

- decisión local
- crear más actores
- enviar más mensajes
- determinar cómo responder al siguiente mensaje

Propiedades de los actores

- componentes de software **livianas** (light-weight)
  - muy buen rendimiento
- cada actor se encarga de **una sola tarea simple** (por convencion)
  (composicional vs. monolítico)
  - más fácil de razonar
- **determinísticas**: se comunican a través de mensajes
- la lógica más compleja emerge de la **interacción** entre actores: delegar tareas, pasar mensajes

Los actores son declarativos, pero…

Los actores pueden modificar su estado interno (privado) de forma explícita :-O

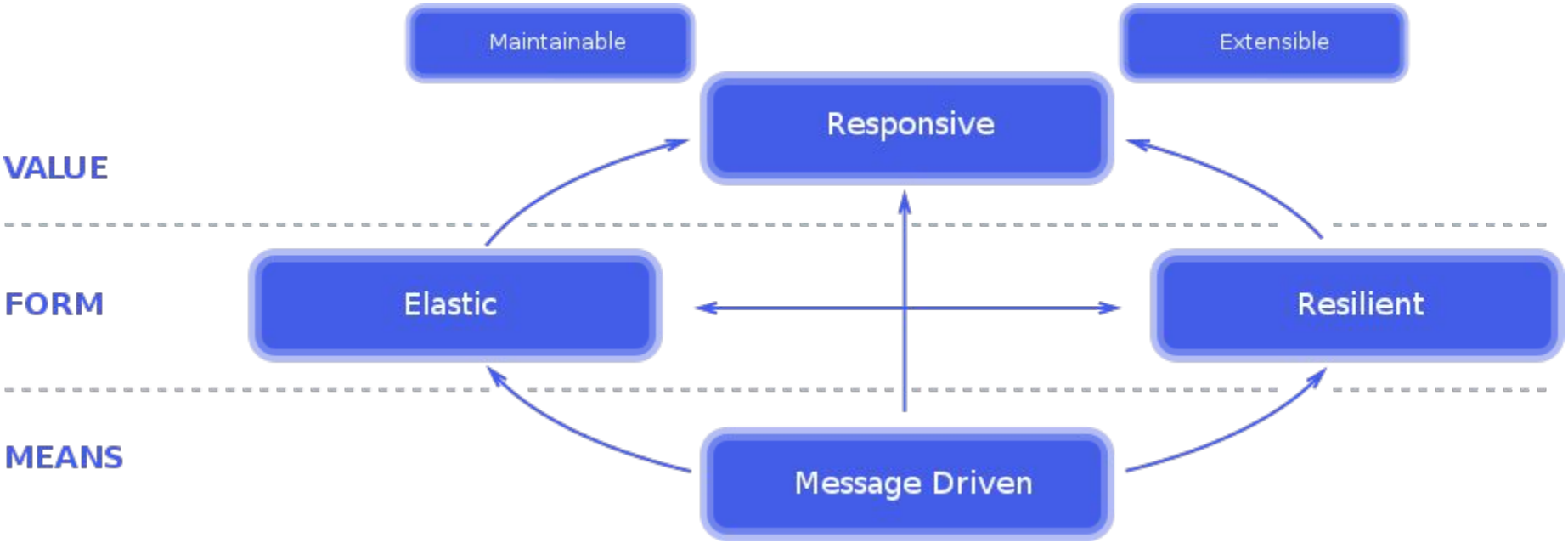
sólo pueden afectar a otros actores a través de mensajes

- determinísticos
- no hay sección crítica
- no se necesitan locks

Pasaje de mensajes

- asíncrono
- sin orden
- en buffer

Decisiones de diseño de orientación a actores



Aplicaciones (casos de uso)

- mensajería: email, telefonía, whatsapp
- Web services: Simple Object Access Protocol (SOAP), endpoints.

Akka

¿Qué es Akka?

Construccion lingüistica para Scala

Akka is a free and open-source toolkit and runtime simplifying the construction of concurrent and distributed applications on the JVM. Akka supports multiple programming models for concurrency, but it emphasizes actor-based concurrency, with inspiration drawn from Erlang.[2]

Language bindings exist for both Java and Scala. Akka is written in Scala and, as of Scala 2.10, the actors in the Scala standard library are deprecated in favor of Akka.

Qué tiene Akka que no tengan los demás

- Concurrency is message-based and asynchronous: typically no mutable data are shared and no synchronization primitives are used; Akka implements the actor model.

Qué tiene Akka que no tengan los demás

- The way actors interact is the same whether they are on the same host or separate hosts, communicating directly or through routing facilities, running on a few threads or many threads, etc. Such details may be altered at deployment time through a configuration mechanism, allowing a program to be scaled up (to make use of more powerful servers) and out (to make use of more servers) without modification.

Qué tiene Akka que no tengan los demás

- Actors are arranged hierarchically with regard to program failures, which are treated as events to be handled by an actor's supervisor (regardless of which actor sent the message triggering the failure). In contrast to Erlang, Akka enforces parental supervision, which means that each actor is created and supervised by its parent actor.

Qué tiene Akka que no tengan los demás

- Akka has a modular structure, with a core module providing actors. Other modules are available to add features such as network distribution of actors, cluster support, Command and Event Sourcing, integration with various third-party systems (e.g. Apache Camel, ZeroMQ), and even support for other concurrency models such as Futures and Agents.