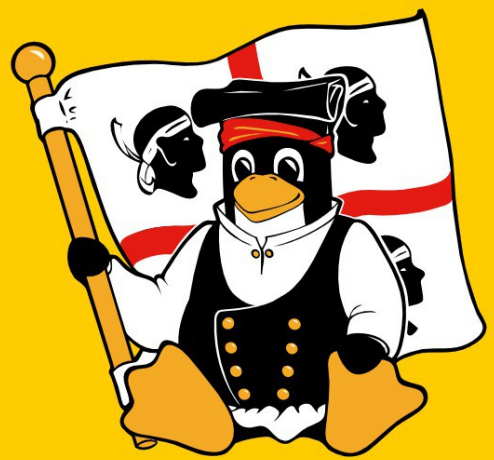
Reactive Applications

di Massimiliano Dessì



GULCh
Gruppo Utenti Linux Cagliari h...?

Speaker

@desmax74

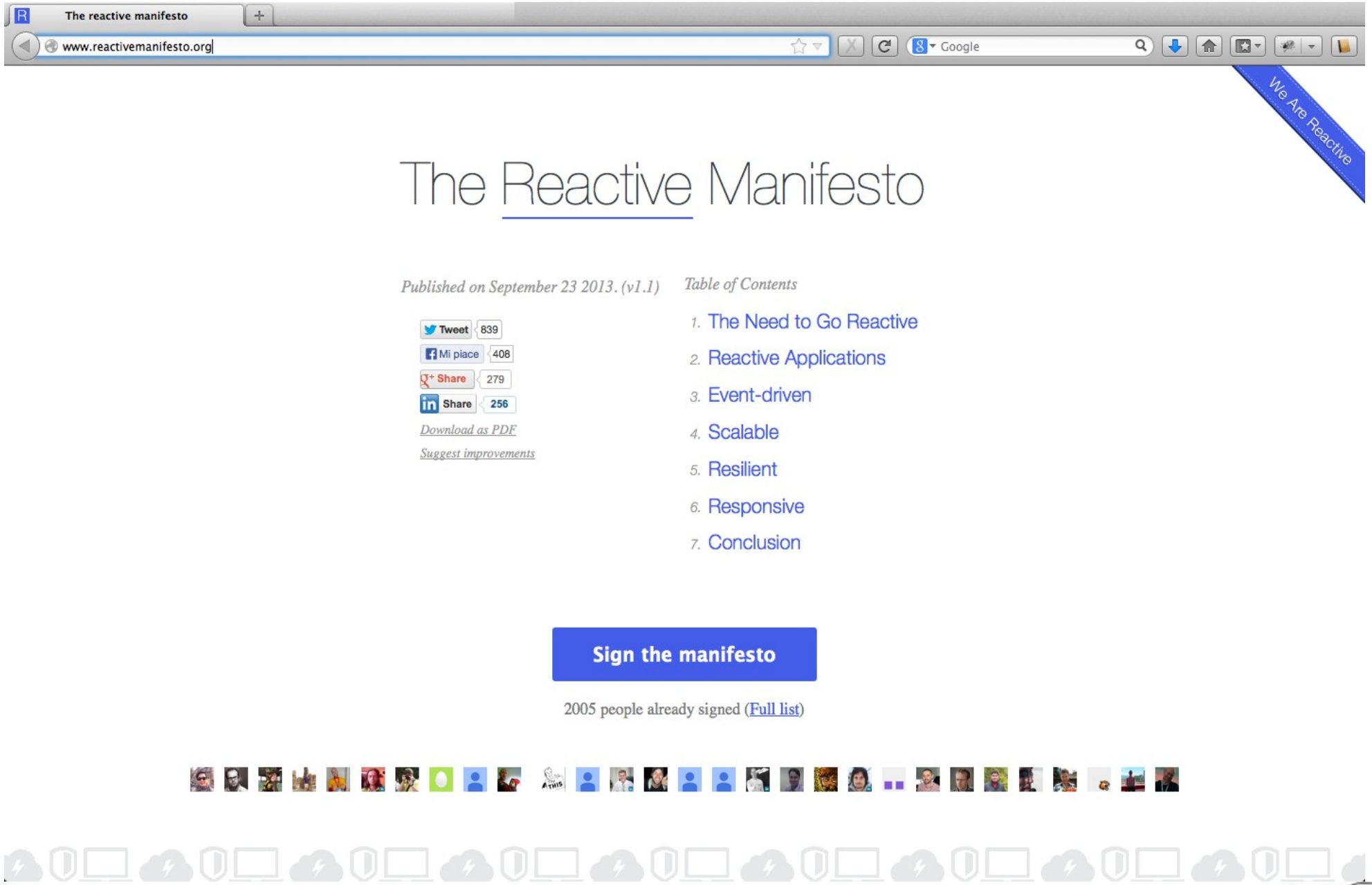
Massimiliano Dessì has more than 13 years of experience in programming.

He's a proud father of three.

Manager of GDG Sardegna, Founder of S SpringFramework IT, co-founder of Jug Sardegna. Author of Spring 2.5 AOP.

He works and lives in Cagliari, Italy.

Reactive applications



Software requirements nowadays

Highly Responsive, Real Time

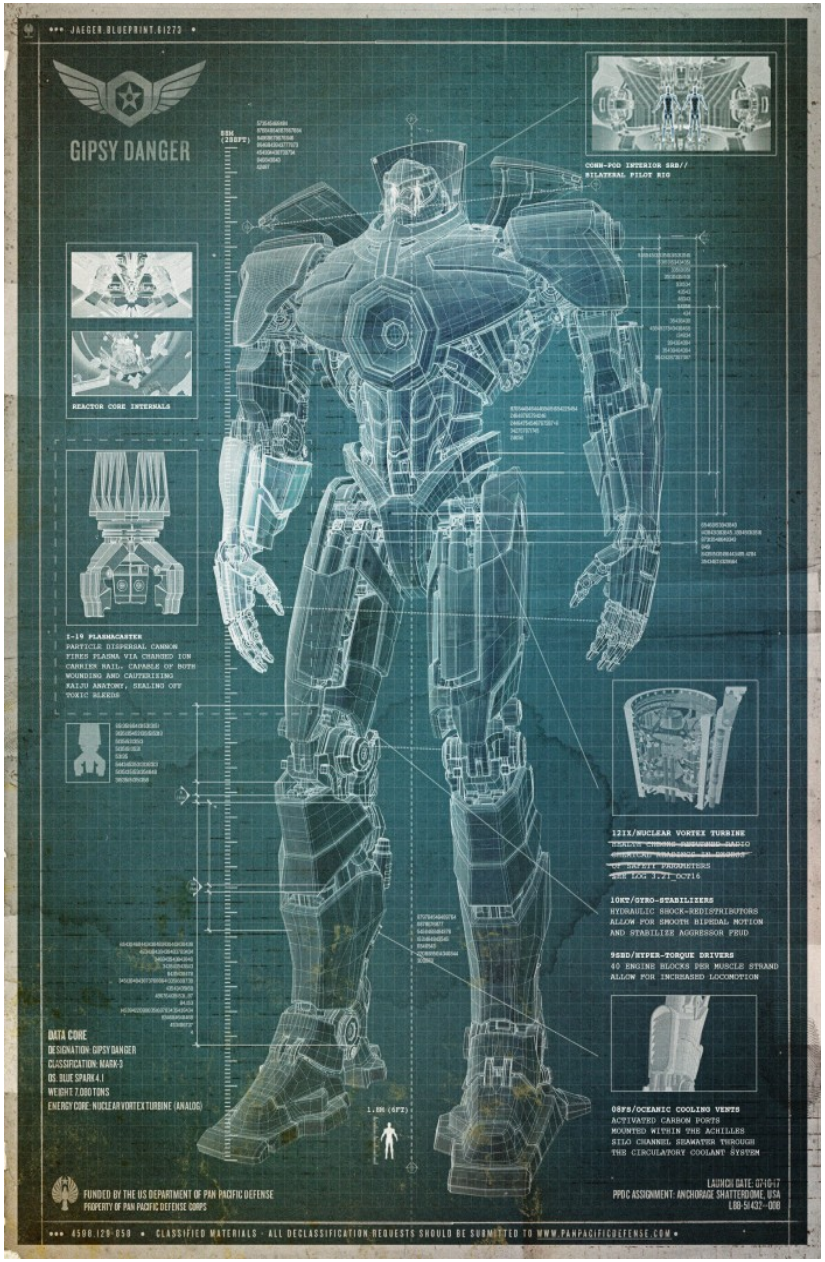
Scalable

Resilient

Petabytes

New problems

We need better tools



http://www.pacificrimmovie.net/

Reactive

"readily responsive to a stimulus"

Component active and ready to respond to event

Event Driven

Reactive

React to events → Event Driven

React to failure → Resilient

React through a UI → Interactive

React to load → Scalable

React to event - Event driven

Asyncronous and loosely coupled

+

Non blocking

=

lower latency and higher throughput

React to event - Event driven

Productors push asyncronously data

towards consumers (message passing)

better resource usage

instead of

having consumers continually ask for data

React to event - Event driven

Non blocking operation

mean have all the time

the application responsive

even under failure

Event driven

Actors , No shared mutable state

Promise , Composable

Message passing asyncronous, non Blocking

Lock free concurrency

Scalable

"Capable of being easily expanded or upgraded on demand"

Event driven and message passing are the foundations

Scalable

"loose coupling and location independence between components and subsystems make it possible to scale out the system onto multiple node"

Location transparency

!=

transparent distributed computing

Resilient

“the capacity to recover quickly from difficulties”

In a reactive application, resilience is part of the design from the beginning

Resilient

Bulkheads and Circuit Breaker patterns

Isolate failure

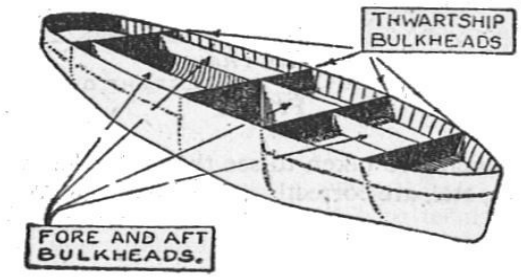
Manage Failure locally

Avoid cascading failure

Resilient

Bulkheads

Failure is modeled explicitly in order to be compartmentalized, observed, managed and configured in a declarative way, and where the system can heal itself and recover automatically



Resilient

Actor lightweight isolated process

(400 bytes of heap space)

Each process has its own supervisor

In case of failure the supervisor receive as async msg the error

The supervisor can choose the recovery strategy

`kill, restart, suspend/resume`

Resilient

Actor decoupling business logic from handling error

Mailbox Guaranteed Delivery

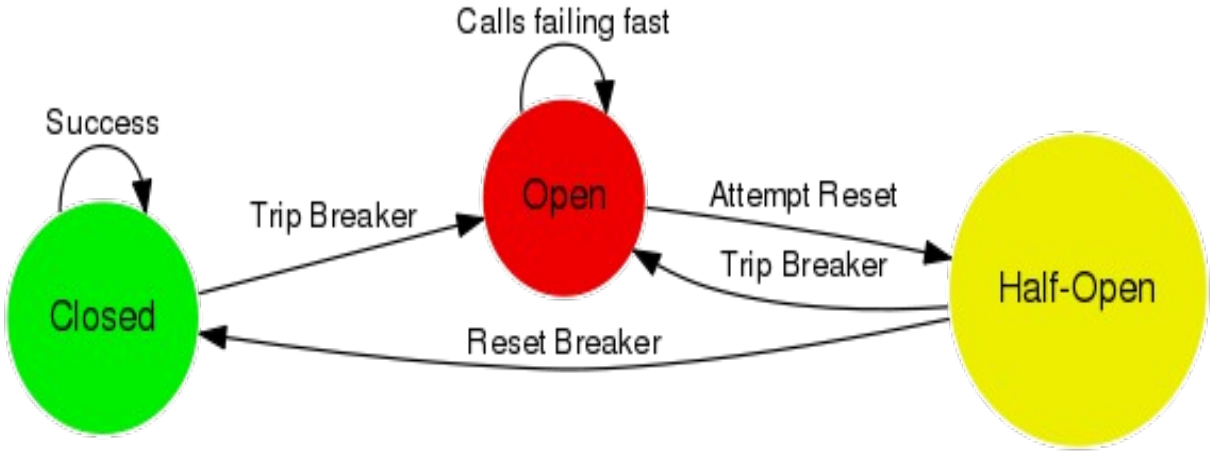
```
my-dispatcher {
        mailbox-type = akka.actor.mailbox.filebased.FileBasedMailboxType
}
```

Resilient

```
override val supervisorStrategy =
OneForOneStrategy(maxNrOfRetries = 10,
withinTimeRange = 1 minute) {

        case _: java.sql.SQLException => Resume

        case _: NullPointerException => Restart

       case _: Exception => Escalate

}
```

Resilient

A circuit breaker is used to provide stability and prevent cascading failures in distributed systems.




```
val breaker =
        new CircuitBreaker(
                context.system.scheduler,
                maxFailures = 5,
                callTimeout = 10.seconds,
                resetTimeout = 1.minute).onOpen(notifyMeOnOpen())
```

doc.akka.io/docs/akka/current/common/circuitbreaker.html

Responsive

quick to respond or react appropriately

Reactive applications use observable models, event streams and stateful clients.

Tools

Actors

Agent

Future

Functional Reactive Programming

Tools

Actors

Share nothing

Each actor has a Mailbox (message queue)

Comunicates through async and non blocking message passing

Location transaparent

Tools

Agents

Reactive memory cells

Send an update function to the agent which:

1) add to an ordered queue, to be

2)applied to the agent async and non blocking

Reads are “free”

Composes

http://clojure.org/agents

Agent

```
import scala.concurrent.ExecutionContext.Implicits.global
import akka.agent.Agent

 val agent = Agent(5)

 agent.send(7); //Update (atomically and asyncronously)

 val result = agent.get
 //Reading an Agent's current value happens immediately
```

If an Agent is used within an enclosing Scala STM transaction, then it will participate in that transaction

Tools

Futures

Span concurrency with not yet computed result

Write once, read many

Freely sharable

Allows non blocking composition

Monadic

Built in model for managing failure

Future

Future Read-only placeholder

```
val f1 = Future {
   "Hello" + "World"}

val f2 = f1 map {
     x ⇒  x.length
}
val result = Await.result(f2, 1 second)
```

Promise

Writeable, single-assignment container, which completes a Future

```
import scala.concurrent.{ future, promise }

val p = promise[T]
val f = p.future
val producer = future {
val r = produceSomething()
p success r
ContinueDoingSomethingUnrelated()}

val consumer = future {
startDoingSomething()
f onSuccess {
   case r => doSomethingWithResult()   }
}
```

Tools

Functional reactive programming

Extends futures with concept of stream

Functional variation of the observer pattern

A signal attached to a stream of events

The signal is reevaluated for each event

Model events on a linear timeline deterministic

Compose nicely

Rx, RXJava, Scala.RX, Reactive.js, Knockout.js

References

- http://www.reactivemanifesto.org/
- http://akka.io/ (JAVA and SCALA API)
- Deprecating the observer pattern

  http://lampwww.epfl.ch/~imaier/pub/DeprecatingObserversTR2010.pdf

Q &A

Thanks for your attention