Actor-Based Programming

Jan Bessai

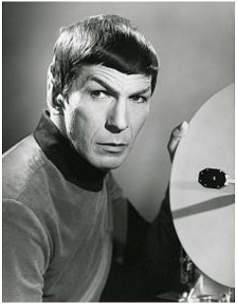
2018-10-02

technische universität
dortmund

fakultät für
informatik

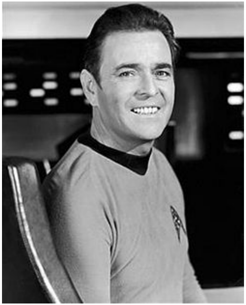
LEHRSTUHL 14
SOFTWARE ENGINEERING

Today: Actor-Based Programming



Theory

Spirit



Practice

Graphics: `en.wikipedia.org`

Concurrent and or Distributed Systems in Computer Science (Models)

- Petri Nets [23]
- Communicating Sequential Processes (CSP) [11]
  - used in Go [15]
- Communicating Concurrent Processes (CCS) [19], π-Calculus [20]
- Threads, Mutexes, Semaphores, Monitors [6]
  - all modern operating systems
- Kahn Process Networks [14]
  - designing embedded systems, network buffers
- Arrow-Calculus [13]
  - Functional Programming, Haskell [12]
- ...
- **Today:** Actors

Actors – Why?

- Simple old concept: Carl Hewitt et al. 1973 [9]
- Theoretically interesting:
  - nondeterminism vs. indeterminism
- Direct connection to (micro-) services
- Practically interesting
  - Foundation of Erlang
  - Libraries for almost every language
  - Adopted by industry: Intel, Paypal, Amazon, Zalando

Actors - Basics [5]

Definition (Actor)

An Actor is an object capable of receiving a message and then performing three operations:

1. create a finite number of new actors
2. send a finite number of messages
3. designate the behavior when receiving the next message

Definition (Actor System)

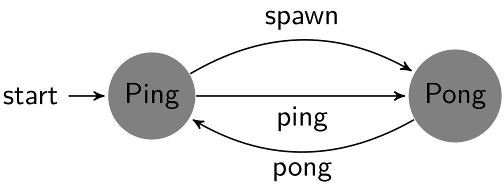
An Actor System

1. manages names by which actors address each other
2. provides message delivery guarantees:
    - arrival
    - duplicate freedom
    - **no order/time guarantees**

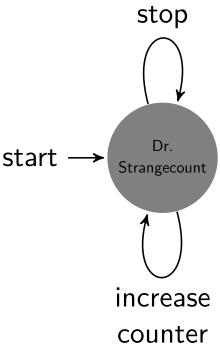
For a mathematical formalization see [8]

- e.g. to prevent Zenon machines [24]

Example



Indeterminism [5]



- On "start":
    - `counter := 0`
    - `Send "stop" to Dr. Strangecount`
    - `Send "increase counter" to Dr. Strangecount`
- On "increase counter":
    - `counter := counter + 1`
    - `Send "increase counter" to Dr. Strangecount`
- On "stop"
    - `Change behavior to ignore all messages`

- Does this system halt?
- If so: what will be the value of counter?

Excursion: Nondeterminism [17]

Definition (Non-deterministic Algorithm)

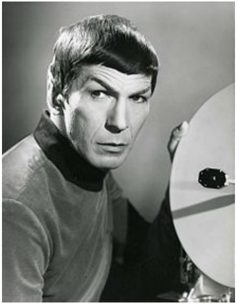
A non-deterministic algorithm may choose between a *finite* number of control flows

1. Return true if any possible control flow returns true
2. Otherwise reject or loop

a	b	c	$f(a, b, c)$
0	0	0	0
0	0	1	0
		...	
1	0	1	1
		...	
1	1	1	0

```
choose $a, b, c \in \{(0,0,0), (0,0,1), \dots, (1,1,1)\}$
if $f(a,b,c) = 1$ then return true
else return false
```

- Runtime: time of shortest returning control flow
- Equal in power to normal Turing Machines (programs)
- Important complexity classes, e.g. Nondeterministic Polynomial Time (NP)
- Actors: more power, infinite possible control flows [22]



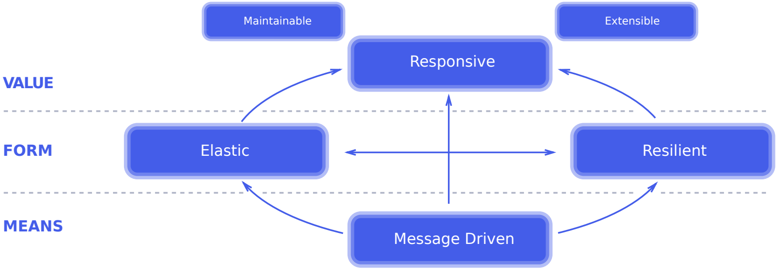
$\Longrightarrow$

Innovation Cycle

A typical innovation cycle in software engineering:

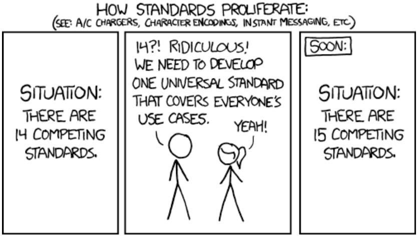
1. Scientists discover something [9, 8]
2. Things quiet down for years
3. Practitioners are frustrated with the state of the art
4. Some of them organize and declare principles
   - e.g. in a developer conference keynote
   - sometimes this results in a "manifesto" [3]
5. Everybody uses and implements new toys
6. Chaos [7] and disasters [2] have to be tamed
7. Things die out and/or solidify in standards [25]

The Reactive Manifesto (Bonér et al. [3])



- Motivational text with some requirements for software systems
- Not tied to any specific technology
    - actors are just one possibility out of many
- Inherent vagueness of terms
- Similar manifestos exist, e.g. for Agile [18]

Solidification and Standardization


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Source: `https://xkcd.com/927/`

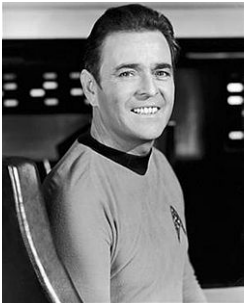
Some key technologies for reactive and actor-based programming have been standardized:

- Java Flow API [25]
- Server-Sent Events [10] (”push notifications”)
- JSON-Serialization format [4]

Others lack proper standards:

- Coupling services (actors) of different frameworks
- Java and .Net world are largely incompatible [21]
- Deployment to different cloud services

$\Longrightarrow$



Akka

Source: akka.io

Akka is a collection of libraries for actor-based systems:

- Open Source, available for Java, Scala, and .NET
- Provides abstractions to define actors and their behavior
- Encapsulates low-level message-passing protocols
- Support for monitoring, automatic restarting, and load balancing (Resilience, Elasticity)
- Much more [1]

Akka vs. AWS-λ, Azure, Heroku

- $+$ Deployment on local servers possible
  - $+$ compliance with data-protection laws [16]
  - $\sim$ security
  - $\sim$ operating costs
  - $\sim$ deployment on AWS via Docker
- $+$ API Updates at your own pace
- $+$ Accompanying technology options
  - Event Sourcing
  - Self-Healing
- $-$ Pre-configuration
- $-$ Cost-effectiveness under certain loads

Analyze: How long will your application live? Expected costs for infrastructure? Ok to handle user-data on foreign servers? Prior experience of available developers? ...

Example - Initialization

An anonymous voting service using Akka actors:

- Voting management service is initialized with a list of parties and names of people who can vote
- Start message triggers the management service to spawn a polling station
- Polling station prints ballots with the parties, its address, and random passwords for registered voters
- Management service forwards ballots, polling station address and individual passwords to voters

Example - Voting

- Voters send their vote, password and reply address to the station
- For each password sent, the vote is counted if the password wasn't previously used and the party is eligible
- Votes for non eligible parties are counted as invalid/abstained
- Reply addresses are informed about success of individual votes in a participation receipt
- When all votes are counted, results are sent back to the management service
- The polling station service shall only be informed about passwords, and never hold a map from passwords to registered voters

Code!

Bonus Questions

- Can we parallelize multiple voting stations?
- Can votes be forwarded in secret to gain further anonymity?
- Can we add persistence and restart crashed polling stations?
- Which actors could join clusters on different computers?

Conclusion

- Actors as a powerful model for distributed concurrent computation
- In- vs. Nondeterminism
- Social aspects of innovation in action
- Akka as a possible implementation of actors
- For a deeper dive check out material here: `https://doc.akka.io/docs/akka/2.5/additional/books.html`

Thanks! :)

Literature I

[1] *Akka Homepage*. 2018. URL: www.akka.io.

[2] Jimmy Bogard. *Avoiding Microservice Megadisasters*. 2017. URL: https://www.youtube.com/watch?v=gfh-VCTwMw8.

[3] Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. *The Reactive Manifesto*. 2013 - 2014. URL: https://www.reactivemanifesto.org/.

[4] T. Bray. *RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format*. 2014. URL: https://tools.ietf.org/html/rfc7159.

[5] Clemens Szyperski Carl Hewitt Erik Meijer. *The Actor Model*. 2012. URL: https://www.youtube.com/watch?v=1zVdhDx7Tbs.

Literature II

[6] E. W. Dijkstra. "Solution of a Problem in Concurrent Programming Control". In: *Commun. ACM* 8.9 (Sept. 1965), pp. 569–. ISSN: 0001-0782. DOI: 10.1145/365559.365617. URL: http://doi.acm.org/10.1145/365559.365617.

[7] Josh Evans. *Mastering Chaos - A Netflix Guide to Microservices*. 2016. URL: https://www.youtube.com/watch?v=CZ3wIuvmHeM.

[8] Carl Hewitt and Henry G. Baker. "Laws for Communicating Parallel Processes". In: *IFIP Congress*. 1977, pp. 987–992.

Literature III

[9] Carl Hewitt, Peter Boehler Bishop, Irene Greif, Brian Cantwell Smith, Todd Matson, and Richard Steiger. "Actor Induction and Meta-Evaluation". In: *Conference Record of the ACM Symposium on Principles of Programming Languages, Boston, Massachusetts, USA, October 1973*. Ed. by Patrick C. Fischer and Jeffrey D. Ullman. ACM Press, 1973, pp. 153–168. DOI: 10.1145/512927.512942.

[10] Ian Hickson. *Server-Sent Events (w3c standard)*. 2015. URL: https://www.w3.org/TR/2015/REC-eventsource-20150203/.

[11] C. A. R. Hoare. "Communicating Sequential Processes". In: *Commun. ACM* 21.8 (Aug. 1978), pp. 666–677. ISSN: 0001-0782. DOI: 10.1145/359576.359585. URL: http://doi.acm.org/10.1145/359576.359585.

Literature IV

[12] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. “Arrows, Robots, and Functional Reactive Programming”. In: *Advanced Functional Programming, 4th International School, AFP 2002, Oxford, UK, August 19-24, 2002, Revised Lectures*. Ed. by Johan Jeuring and Simon L. Peyton Jones. Vol. 2638. Lecture Notes in Computer Science. Springer, 2002, pp. 159–187. DOI: 10.1007/978-3-540-44833-4_6.

[13] John Hughes. “Generalising monads to arrows”. In: *Sci. Comput. Program.* 37.1-3 (2000), pp. 67–111. DOI: 10.1016/S0167-6423(99)00023-4.

[14] Gilles Kahn. “The Semantics of Simple Language for Parallel Programming”. In: *IFIP Congress*. 1974, pp. 471–475.

Literature V

[15] Thomas Kappler. *package csp*. 2017. URL: http://godoc.org/github.com/thomas11/csp.

[16] Phil Lee. *The differences between EU and US data protection laws*. 2017. URL: https://www.youtube.com/watch?v=-_zLeGKHOpc.

[17] Anil Maheshwari and Michiel Smid. *Introduction to Theory of Computation*. 2012. URL: http://cglab.ca/~michiel/TheoryOfComputation/TheoryOfComputation.pdf.

[18] Robert C. Martin, Ward Cunningham, Martin Fowler, et al. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org/.

Literature VI

[19] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: 3-540-10235-3. DOI: 10.1007/3-540-10235-3.

[20] Robin Milner, Joachim Parrow, and David Walker. “A Calculus of Mobile Processes, I”. In: *Inf. Comput.* 100.1 (1992), pp. 1–40. DOI: 10.1016/0890-5401(92)90008-4.

[21] Chris Ochs. *Remoting example on wiki not working?* 2014. URL: https://github.com/akkadotnet/akka.net/issues/132.

[22] Toby Ord. “The Many Forms of Hypercomputation”. In: *Applied Mathematics and Computation* 178.1 (2006), pp. 143–153. URL: https://doi.org/10.1016/j.amc.2005.09.076.

Literature VII

[23] Carl Adam Petri. “Communication with automata”. eng. PhD thesis. Universität Hamburg, 1966.

[24] Petrus H. Potgieter. “Zeno machines and hypercomputation”. In: *Theor. Comput. Sci.* 358.1 (2006), pp. 23–33. DOI: 10.1016/j.tcs.2005.11.040.

[25] Rahul Srivastava. *Reactive Programming with JDK 9 Flow API*. 2016. URL: https://community.oracle.com/docs/DOC-1006738.