Seminar Advanced Computer Architecture

# Actors

Paul Preissner
Technische Universität München

06.07.2017

**Abstract**

The actor model is a programming paradigm developed specifically with concurrent and distributed computing in mind. It defines a system of self-contained actors that only communicate with each other through dedicated messages and adhere to the basic semantic properties of encapsulation, fairness and location transparency. This paper will elaborate on the history, fundamental concepts and aforementioned semantic properties of the model and discuss the issues the reality of implementation entails as well as current usage of the model in commercial projects.

## 1 Introduction

Ever since the introduction of multi-processor systems in the 1970s [S] and even more so the introduction of multi-core processors in the early 2000s, one of the main topics in computer science has been the question of how these systems can be programmed, used and exploited to utilize them to full capacity. Now quite naturally the usage and utilization of any computer and particularly how easy it is for the programmer to do it is very much dependent on the programming language employed in a program. Over the years, most programming languages have either been adapted to enable concurrent computing in at least a basic form or efforts have been made to enable concurrent computing through additional libraries and frameworks. But since "[c]oncurrency can be solved by a good programmer in many languages, [...] it's a tough problem to solve" as former Twitter engineer Alex Payne describes [S, 5-15], it would seem counter-intuitive to use a programming language defined at a time when sequential execution was the status quo. As such in the 1970s research first delved into the task of defining languages or rather programming paradigms that are meant to primarily deal with concurrent computing tasks. One of these paradigms is the so-called actor model, as message-passing based paradigm for concurrent and distributed computing, and in that the underlying topic of this paper. To latch onto Payne again, he continues that the Actor model is "commonly used to solve concurrency problems, and it makes that problem a lot easier to solve" [S, 5-15].

Subsequently, this paper will first take a brief look at the history and a longer look at the fundamentals and basic semantics of the Actor model, followed by a discussion of the reported and expected issues the model brings with it in an actual implementation and concluded by a rundown of its most promising current usage scenarios. In this effort it is based primarily on the work of Gul Agha on the topic since the 1980s as well as his and his colleagues research in recent years.

## 2 Actor model history

Ideas vaguely resembling parts of the Actor model can be found in early computational models like Petri nets, Lambda Calculus or Smalltalk in that they are influenced by the message passing concept as well. According to Carl Hewitt, some of these as well as physics actually influenced him in his work on the initial definition of Actors. In the following few years, Hewitt and his colleagues, notably Irene Greif and Henry Baker, developed operational semantics and axioms for the model. In a debate between Hewitt and Edsger Dijkstra in the late 1970s the topic of unbounded nondeterminism came up, which describes the property that the delay for servicing requests can be unbounded due to contention for shared resources yet the model guarantees that the requests will eventually be serviced. Dijkstra argued that according to his model of concurrency, this property could not be fulfilled by Actors. In 1981 however, William Clinger using domain theory proved that the Actor model does indeed fulfill this property. Another milestone was Gul Agha's work on Actors in 1985 forward, completing the Actor model theory as a transition-based semantic model which still holds as the basis for most if not all discussions and implementations regarding Actors. Since then Agha with his colleagues and the groups at the California Institute of Technology, Massachusetts Institute of Technology and other international universities have researched further into the theoretical application of the Actor model in modern systems, while the MIT Message Passing Semantics Group as well as commercial projects have done a lot of practical implementation work. After years of being regarded as sort of a niche topic, the latter especially sparked up again in recent time as industries recognized the relevance of the Actor model for concurrent computing in embedded and distributed systems, especially in regards to automation as well as the establishment of the internet as a core technology of modern society and industry. [SOURCE THIS STUFF.]

## 3 Fundamentals & semantics of the Actor model

### 3.1 Fundamental concept

- paradigm: ”everything” is an actor (thread, process, core, socket, node, system, ...) -¿ one actor encapsulates one computation unit

- an actor may send messages to actors it knows by name
- an (idling) actor receiving a message will accept it and execute the computation defined within, resulting in the possible actions:
  - sending new messages
  - creating new actors
  - updating its local state
- an actor can only influence its own local state

### 3.2 Semantics

- Actor semantics have three main properties
  - Encapsulation & atomicity (actors don't share state, process one message at a time)
  - Fairness (every actor makes progress, every message delivered eventually)
  - Location transparency (physical location not bound to identifier, hidden migration)
- in reality, some aspects aren't implemented faithfully (for efficiency, complexity)
- concerns about scalability & performance

## 4 Reality and issues of implementation

### 4.1 Tools and languages

Blob.

### 4.2 Peculiarities in implementation

Issues.

## 5 Current usage of actor systems

Focus: current usage - embedded systems, cloud/distributed computing and microservices

## 6 Versus other models of concurrency

Expansion: comparison to other models/paradigms of concurrent computation

## 7 Summary

Summary.