# Using the Actor Model in ACE

Jonathan Thaler

November 11, 2016

**Abstract**

The Actor-Model, a model of concurrency, has been around since the paper Hewitt et al. (1973) in 1973. It was a major influence in designing the concept of Agents and although there are important differences between Actors and Agents there are huge similarities thus the idea to use actors to build agent-based simulations comes quite natural. Although there are papers around using the actor model as basis for their ABMS unfortunately no proper theoretical treatment of using the actor-model in implementing agent-based simulations has been done so far. This paper looks into how the more theoretical foundations of the suitability of actor-model to ABMS and what the upsides and downsides of using it are.

## 1 Introduction

`http://www.grids.ac.uk/Complex/ABMS/`
Bezirgiannis (2013) describes in chapter 3.3 a naive clone of NetLogo in the Erlang programming language where each agent was represented as an Erlang process. The author claims the 1:1 mapping between agent and process to "be inherently wrong" because when recursively sending messages (e.g. A to B to A) it will deadlock as A is already awaiting Bs answer. Of course this is one of the problems when adopting Erlang/Scala with Akka/the Actor Model for implementing agents *but it is inherently short-sighted to discharge the actor-model approach just because recursive messaging leads to a deadlock*. It is not a problem of the actor-model but merely a very problem with the communication protocol which needs to be more sophisticated than Bezirgiannis (2013) described. The hypothesis is that the communication protocol will be in fact *very highly application-specific* thus leading to non-reusable agents (across domains, they should but be re-usable within domains e.g. market-simulations) as they only understand the domain-specific protocol. This is definitely NOT a drawback but can't be solved otherwise as in the end (the content of the) communication can be understand to be the very domain of the simulation and is thus not generalizable. Of course specific patterns will show up like "multi-step handshakes" but they are again then specifically applied to the concrete domain.

## 1.1 The Actor Model

Read in the following order

1. Hewitt et al. (1973)
2. Grief and Greif (1975)
3. Clinger (1981)
4. Agha (1986)
5. Agha et al. (1997)
6. Hewitt (2007)
7. Hewitt (2010)
8. Agha (2004)

upside: extreme huge number of agnts possible due to distributed and parallel technology downside: depends on system & hardware: scheduler, system time, systime resolution (not very nice for scientific computation), much more complicated, debugging difficult due to concurrency, no global notion of time appart from systemtime, thus always runs in real-time, but there is no global notion of time in the actor model anyway, no EDSL full of technical details, no determinism, no reasoning

## 1.2 Agents vs. Actors

Agents more a high-level concept, Actors low level, technical concurrency primitives

## 1.3 Actors are pure functional!

## 1.4 Hypothesis

This makes simulations very difficult and also due to concurrency implementing a sync conversation among agents is very cumbersome. I have already experience with the Actor Model when implementing a small version of my Master-Thesis Simulation in Erlang which uses the Actor Model as well. For a continuous simulation it was actually not that bad but the problem there was that between a round-trip between 2 agents other messages could have already interfered - this was a problem when agents trade with each other, so one has to implement synchronized trading where only messages from the current agent one trades with are allowed otherwise budget constraints could be violated. Thus I think Erlang/Akka/Actor Model is better suited for distributed high-tolerance concurrent/parallel systems instead for simulations. Note: this is definitely a major point I have to argue in my thesis: why I am rejecting the actor model.

AKKA: thus my prediction is: akka/actor model is very well suited to simulations which 1. dont rely on global time 2. dont have multi-step conversations: interactions among agents which are only question-answer. TODO: find some classical simulation model which satisfies these criterias.

# 2 Actor Model implementations

It is important to note that the actor-model is not tied to any particular programming language and could be (and is) implemented in different types of languages like Java (Object-Oriented), Haskell (pure functional) and Scala (mix of functional and Object-Oriented).

## 2.1 Erlang

TODO: erlang is an old implementation of the actor model

## 2.2 Akka

TODO: akka is a modern implementation of the actor model

## 2.3 Haskell

TODO: can implement actor-model using forkIO and STM

# 3 Theoretical reflections

## 3.1 The problem of time

how can we simulate global time? how can we implement multistep conversations (by futures)?
The real problem seems to be concurrency but i feel we can simulate concurrency by synchronizing to continuous time. computations are carried out after another but because time is explicitly modelled they happen logically at the same time. these rules hold: an agent cannot be in two conversations at the same time, the agent can be in only one or none conversation at a given time t.
What if time is of no importance and only the continuous dynamics are of interest?
To put it another way: real concurrency (with threads) makes time implicit which is what one does NOT want in simulation. Maybe FRP is the way to go because it allows to explicitly model continuous and discrete time, but I have to get into FRP first to make a proper judgement about its suitability.

### 3.2 Conversations

new concept: not single, async messages, but syncronous conversations which (can) take time = multiple synchronous messages between agents which (can) change the state of an agent in the end.

## 4 Example implementation

### 4.1 Wildfire with/without wind

"This is a model of a wildfire. Vegetation is modeled as grid cells - agents in discrete space. The burning time of a cell is proportional to the amount of "fuel" in the cell, which is randomly generated at the model startup. While burning, the cell may cause ignition in adjacent cells. You can cause the initial ignition by clicking a cell. The ignition may also be caused by a bomb dropped by the aircraft - an agent moving in continuous 2D space that overlaps the discrete space. This model, among other things, shows how the two types of space can be linked. The model is computationally very efficient because there are no time steps in this model; the behavior of vegetation cells and the aircraft is defined in the form of a statechart. Unlike in the full version, in this model wind is not taken into consideration."
"This is a model of a wildfire. Vegetation is modeled as grid cells - agents in discrete space. The burning time of a cell is proportional to the amount of "fuel" in the cell, which is randomly generated at the model startup. While burning, the cell may cause ignition in adjacent cells. The probability of ignition depends on the wind direction (which you can change on the fly). You can cause the initial ignition by clicking a cell. The ignition may also be caused by a bomb dropped by the aircraft – an agent moving in continuous 2D space that overlaps the discrete space. This model, among other things, shows how the two types of space can be linked. The model is computationally very efficient because there are no time steps in this model; the behavior of vegetation cells and the aircraft is defined in the form of a statechart."

1. No time steps.
2. One-Step interaction.

### 4.2 Discrete SIR/S

1. Discrete time steps.
2. One-Step interaction.

## 5 Decentralized bartering

1. Continuous time.

2. Multi-Step interactions.

# 6 Further Research

## 6.1 Pure functional approach

## 6.2 Explicit time

# 7 Conclusion

## 7.1 Real concurrency not needed in simulation

AKKA is nice but I think the actor model is not very well suited for simulations due to inherent concurrency where time is implicit.
real concurrency is not needed: simultaneous events can be modeled through explicit time but calculated sequential - when we reduce agents to process only one message after another and not multiple concurrently. thus true parallelism is only a technical detail for performance enhancement.

# References


Agha, G. (1986). *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA.

Agha, G. (2004). An algebraic theory of actors and its application to a simple object-based language. In *In Ole-Johan Dahl's Festschrift, volume 2635 of LNCS*, pages 26–57. Springer.

Agha, G. A., Mason, I. A., Smith, S. F., and Talcott, C. L. (1997). A foundation for actor computation. *J. Funct. Program.*, 7(1):1–72.

Bezirgiannis, N. (2013). Improving performance of simulation software using haskell's concurrency & parallelism. Master's thesis, Utrecht University - Dept. of Information and Computing Sciences.

Clinger, W. D. (1981). Foundations of actor semantics. Technical report, Cambridge, MA, USA.

Grief, I. and Greif, I. (1975). Semantics of communicating parallel processes. Technical report, Cambridge, MA, USA.

Hewitt, C. (2007). *What Is Commitment? Physical, Organizational, and Social (Revised)*, pages 293–307. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hewitt, C. (2010). Actor model for discretionary, adaptive concurrency. *CoRR*, abs/1008.1459.

Hewitt, C., Bishop, P., and Steiger, R. (1973). A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 235–245, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.