Actors

Paul Preißner

Fakultät für Informatik
Technische Universität München

July 6th 2017

Brief history

- C. Hewitt et al. '73 onward: first theory of actor model, operational semantics, axioms
- W. Clinger '81: proved unbounded nondeterminism property
- G. Agha '85: formalization of semantic model
- Theoretical/Practical research by MIT, CalTech, industry, etc.
- Recent resurgence (strong relevance to distributed/cloud computing)

“A Model of Concurrent Computation in Distributed Systems“

- actors encapsulate computation (technically at any level)
- an actor may only 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

$\rightarrow$”self-contained, autonomous, interactive, asynchronously operating components” [Karmani, Agha]

Example structure

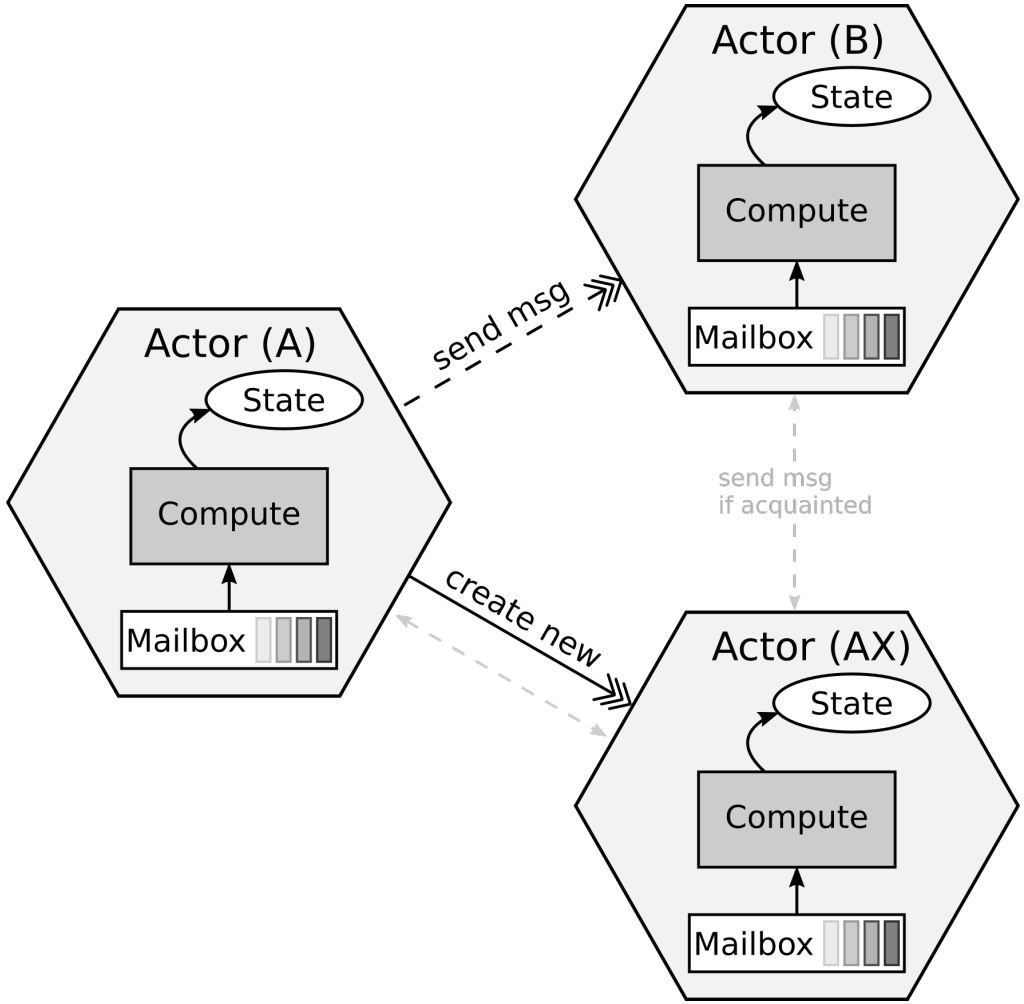

Figure: Any actor may send messages to other known actors, create new actors or update its own state.

Hello ...

```
1  [includes, usings]
2
3  behavior pong(event_based_actor* self, string selfname) {
4    return {
5      //if the message contains a string, proceed
6      [=](const string& what) -> string {
7        aout(self) << selfname << ": " << what << endl;
8        // reply Pong
9        return string("Pong!");
10     }
11   };
12 }
```

Specify behavior

Hello ...

```
1  void ping(event_based_actor* self, const actor& buddy,
       string selfname) {
2    // send Ping to buddy (timeout for reply = 10s)
3    self->request(buddy, std::chrono::seconds(10),
         "Ping!").then(
4      //if the message contains a string, proceed
5      [=](const string& what) {
6        aout(self) << selfname << ": " << what << endl;
7            //if reply is as expected, restart ping again
8            if(what.compare("Pong...") == 0)
9                    ping(self, buddy, selfname);
10     }
11   );
12 }
```

Specify actions

... World!

```
1 int main() {
2   [caf setup]
3   // create a new actor that calls 'pong()'
4   auto actor_B = system.spawn(pong, "B");
5   // create another actor that calls 'ping(actor_B)';
6   auto actor_A = system.spawn(ping, actor_B, "A"); }
```

Spawn actors and start something

```
1 B: Ping...
2 A: Pong...
3 B: Ping...
4 A: Pong...
5 B: Ping...
6 A: Pong...
7 [...]
```

Output

Main semantic properties

- Encapsulation & atomic execution: actors don't share state; process one message at a time; arrivals mid-computation need to be buffered;
- Fairness every actor makes progress; every message is delivered eventually; assumes fair scheduler; $\rightarrow$subsystems cannot stall entire program
- Location transparency physical location not bound to identifier; $\rightarrow$(hidden) migration possible, i.e. *mobility* $\rightarrow$allows load-balancing, efficiency optimization

Synchronization ...

RPC-like local sync cons

... and abstraction

Patterns

Worst practices

Worst practices

Support

Distributed systems

DS server HPC microservices

Embedded systems

system level appl design

Versus

???

Q&A