Loosely Coupled Communication in Actor Systems

Raphael Hiesgen
raphael.hiesgen@haw-hamburg.de

iNET RG, Department of Computer Science
Hamburg University of Applied Sciences

January 6, 2014

Hochschule für Angewandte
Wissenschaften Hamburg
Hamburg University of Applied Sciences

Motivation

- Internet of Things (IoT)
  - Vulnerable
  - Low-powered & lossy connections
- Internet-wide systems
  - Software should scale on demand
  - No downtime due to failure of subsystems
  - Dynamic, world-wide deployment
    - Unknown when the code is written

Agenda

1. The Actor Model
2. `libcppa`
3. Loosely Coupled Communication
4. Use Cases
5. Challenges
6. Conferences & Key Players

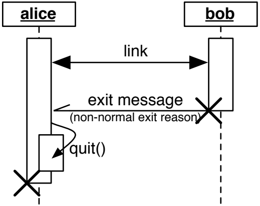
The Actor Model

Actors are concurrent entities, that ...

- Communicate via message passing
- Do not share state
- Can create ("spawn") new actors
- Can monitor other actors

Benefits

- High-level, explicit communication: no locks, no implicit sharing
- Applies to both concurrency *and* distribution
  - Divide workload by spawning actors
  - Interact via network-transparent messaging
  - Run transparently on heterogenous hardware
- Failure semantics
  - Links / Monitors
  - Hierarchical error management
  - Re-deployment at runtime



Implementations

- Erlang
  - Designed by Joe Armstrong at Ericsson in 1986
  - De-facto implementation of the actor model
  - Wide-spread use
  - Inspired other implementations
- Akka
  - Library for Scala based on the actor model
  - Initiated by Jonas Bonér in 2009
  - Developed by Typesafe Inc.
  - Included in the standard library
- `libcppa`

Agenda

1 The Actor Model

2 `libcppa`

3 Loosely Coupled Communication

4 Use Cases

5 Challenges

6 Conferences & Key Players

Features

- Developed by iNET working group at the HAW
- `libcppa` is an actor system based on C++11
- Efficient program execution
  - Low memory footprint
  - Fast, lock-free mailbox implementation
- Targets both low-end and high-performance computing
  - Embedded HW, e.g., running RIOT (http://www.riot-os.org)
  - Large multi-core systems
- Uses internal DSL for pattern matching of messages
- Transparent integration of OpenCL-based actors

Example

```
void math_server() {
  become (
    on(atom("plus"), arg_match) >> [](int a, int b) {
      return make_cow_tuple(atom("result"), a + b);
    }
  );
}
void math_client(actor_ptr ms) {
  sync_send(ms, atom("plus"), 40, 2).then(
    on(atom("result"), arg_match) >> [=](int result){
      cout << "40 + 2 = " << result << endl;
    }
  );
}
int main() {
  spawn(math_client, spawn(math_server));
  // ...
}
```

Linking Actors

```
void bob_fun(); // will fail
void alice_fun() {
  auto bob = spawn<linked>(bob_fun);
  send(bob, "hello bob");
  become ( /* will bob ever call back? */ );
}
void carl() {
  self->trap_exit(true);
  auto alice = spawn<linked>(alice_fun);
  become (
    on(atom("EXIT"), arg_match) >> [](uint32_t r) {
      if (r != exit_reason::normal)
        cout << "something went wrong..." << endl;
    }
  );
}
```

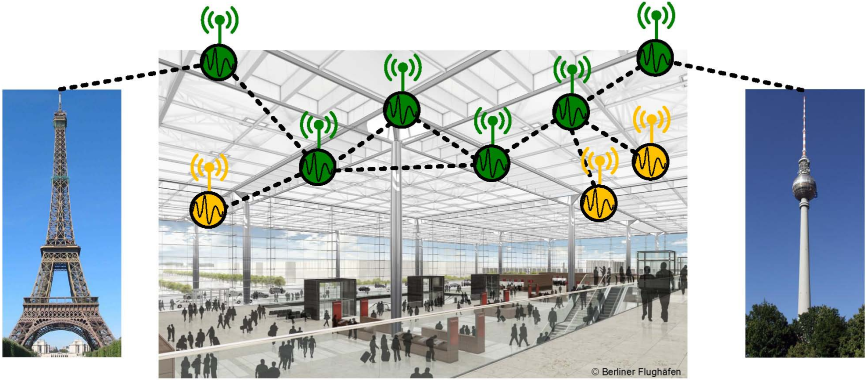
Agenda

Characteristics

- No global state
- Orchestration of independent systems
- Highly scalable
  - Performance depends on cores / nodes
  - Dynamic acquisition of resources based on demand
- Facade-like interfaces
  - Unknown implementation
  - Based on message content
- Suitable for long-distance traffic
  - Unreliable connections
  - Changing topology
  - Tolerant to interference
  - Asynchronous

Agenda

Safest



- Cooperation between:
  - France & Germany
  - Daviko, Forschungsforum Öffentliche Sicherheit, Fraunhofer FOKUS, FU Berlin, HAW Hamburg, INRIA, SAGEM
- Deployed at the airport Berlin Brandenburg

Safest

- Area surveillance system
- Resilient against malfunctions of components
- Respect privacy of citizens
- Distributed collection of anonymized data
- Stepwise evaluation towards central event processing
- Alerts the operator and/or the general public
- Nodes run RIOT & libcppa
    - High-Level modeling & development in the actor model
    - Development & tests on desktop systems
    - Seamless deployment to nodes

Publish/Subscribe

- Scalability
- Decoupled in space / time
- Easy way to deploy redundancy
- Examples:
  - Multicast
  - Information Centric Networking
  - Key-value storage

Agenda

Challenges
Security

- Verify identity of participants
- Ensure integrity of received messages
- Obtain robustness
- Find solutions for a decentralized system
- Changes have local impact

Challenges
Fault-tolerance

- Implement monitors/links for different scenarios
  - Multicast
  - Internet-wide systems
  - Low-powered & lossy connections
- Handle unreliable connections
- Deploy reliable services

Challenges

Communication

- Communicate with unknown systems
  - Rendezvous processes
- Minimize communication overhead
- Failures in pub/sub environments
- Error propagation in non-hierarchical actor systems

Agenda

Conferences

- C++Now
  - Focused on C++
  - May 2014 in Aspen, Colorado
- Splash → Agere!
  - Special Interest Group of ACM focused on Programming Languages
  - Actors, Agents, and Decentralized Control
- ICDCS
  - International Conference on Distributed Computing Systems
  - June / July 2014 in Spain
- PODC
  - ACM Symposium on Principles of Distributed Computing
  - July 2014 in Paris
- USENIX OSDI
  - USENIX Symposium on Operating Systems Design and Implementation
  - October 2014 in Broomfield, Colorado

Key Players

- Carl Hewitt
  - Initial paper on actors
  - Worked on actor model in 1973
- Gul Agha
  - Doctoral student of Hewitt
  - University of Illinois
  - On the Steering committee of the Agere
- Joe Armstrong
  - Developed Erlang in 1986

Key Players
Scala / Akka

- Martin Odersky
  - Professor at the École polytechnique fédérale de Lausanne (EPFL)
  - Developed Scala
- Philipp Haller
  - PhD in Computer Science from EPFL
  - Developed original actor implementation for Scala
- Jonas Bonér
  - Developed Akka
- Viktor Klang
  - Technical lead for the Akka project

Odersky, Bonér and Klang are part of Typesafe, a company focused on "Reactive Programming".

References I

Agha, G. (1986).
Actors: A Model Of Concurrent Computation In Distributed Systems.
Technical Report 844, MIT, Cambridge, MA, USA.

Armstrong, J. (2003).
Making Reliable Distributed Systems in the Presence of Software Errors.
PhD thesis, Department of Microelectronics and Information Technology, KTH, Sweden.

References II

Charousset, D., Schmidt, T. C., Hiesgen, R., and Wählisch, M. (2013).
Native Actors – A Scalable Software Platform for Distributed, Heterogeneous Environments.
In *Proc. of the 4rd Annual Conference on Systems, Programming, and Applications (SPLASH '13), Workshop AGERE!*, New York, NY, USA. ACM.

Hewitt, C., Bishop, P., and Steiger, R. (1973).
A Universal Modular ACTOR Formalism for Artificial Intelligence.
In *Proceedings of the 3rd IJCAI*, pages 235–245, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Thank you for your attention.
Questions?

iNET: `http://inet.cpt.haw-hamburg.de`