Resurrecting Actors:

New Applications of an Old Paradigm in Engineering and Science

Tristan Aubrey-Jones <taj105@ecs.soton.ac.uk>; Supervisor: Bernd Fischer <bf@ecs.soton.ac.uk>

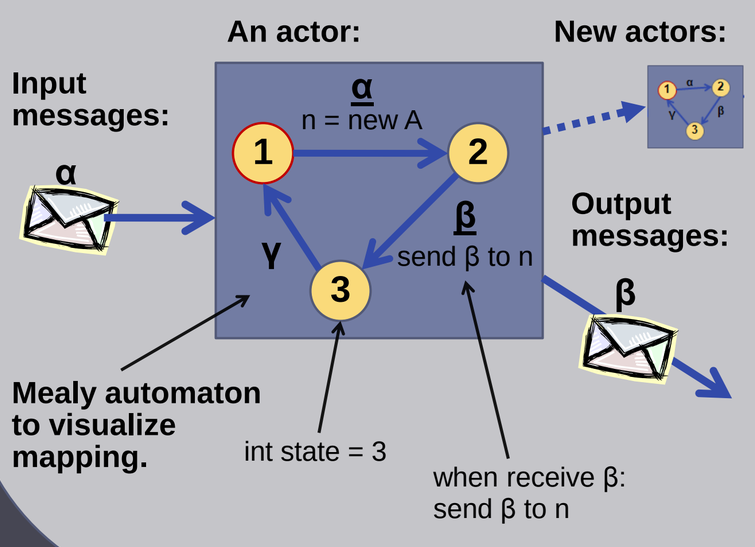
The Actor Model

- **Model of concurrent computation**
- Proposed in 1977 by C. Hewitt [1].
- Actor = a mapping between an input communication and a triple:
  - New state/behaviour,
  - Communications to send, and
  - New actors to create
- Communicate only via message passing
- Isolated (no shared state)
- Implicitly concurrent, with **no locks and no shared memory.**
- More flexible than shared memory model
- **Easily distributed and migrated**.
- Was **stillborn** as a niche AI interest due to:
  - Functional programming bias
  - Lack of highly distributed architectures
  - Inefficiency of message passing

- Now being **revived** in various fields due to:
  - Need to program new complex distributed architectures (multi-core, WSNs, clouds).
  - Usefulness of abstraction in automated code generation & optimizations.
  - Use of familiar imperative & OO syntaxes

```
class A: Actor { // Java
  A nghbor;
  int state = 1;
  receive(Msg m) {
    switch (state) {
     case 1: if (m == Msg.ALPHA) {
       state = 2;
       nghbor = new A();} break;
     case 2: if (m == Msg.BETA) {
       state = 3;
       nghbor.send(Msg.BETA);} break;
     case 3: if (m == Msg.GAMMA)
       state = 1; break;
  }}
}
```

```
(defun ActorA (state neighbor) ; Lisp
 (lambda (msg)
  (case state
   (1 (if (= msg 'alpha)
          (ActorA 2 (ActorA 1 nil))
          (ActorA state neighbor)))
   (2 (if (= msg 'beta)
          (ActorA 3 (neighbor 'beta))))
          (ActorA state neighbor)))
   (3 (if (= msg 'gamma)
          (ActorA 1 neighbor)
          (ActorA state neighbor))))))
```



Concurrency: Actors & Linear Types

- **"Kilim": Lightweight Java actors [2]**
- Implemented via a byte code weaver
- Offers:
  - 1000s of fast lightweight threads.
  - Efficient cooperative scheduling.
  - Actor memory isolation and efficient "zero-copy" message passing, via a **statically enforced linear type system**
- **Overcomes inefficiency of message passing via linear ownership passing of messages:**

```
import kilim.*;

class HtmlMsg implements Message {
  public String html; public HttpRequest req; }
class HttpRequest implements Message {
  public Mailbox<HtmlMsg> replyTo;
  public String url; public String[] cookies; }
class DatabaseConnection implements Message {
  public Object jdbcConnection; }

class RequestQueue extends Mailbox<HttpRequest> {}
class DBConnectionPool extends Mailbox<DatabaseConnection> {}

class HttpRequestHandler extends Actor {

  RequestQueue in; DBConnectionPool pool;
  Mailbox<HtmlMsg> cartmb = new Mailbox<HtmlMsg>(),
     searchmb = new Mailbox<HtmlMsg>();
  ShoppingCartControl cartControl = ... ;
  SearchResultsControl searchControl = ... ;

  @pausable
  public void execute() {
    for(;;) {
      HttpRequest req = in.get();
      HtmlMsg reply = new HtmlMsg();
      handle(req, reply);
      sendReply(req, reply);
  }}

  @pausable
  void handle(@safe HttpRequest req, @cuttable HtmlMsg reply) {
    HttpRequest r = req.clone(); r.replyto = cartmb;
    cartControl.put(r);
    r = req.clone(); r.replyto = searchmb;
    searchControl.put(r);
    reply.html = "<html>"+cartmb.get() + searchmb.get() + "</html>";
  }

  @pausable
  void sendReply(@free HttpRequest req, @free HtmlMsg reply) {
    reply.req = req;
    reply.req.replyTo.put(reply);
  }

  @pausable
  private Results query(@safe DatabaseConnection con,
                        @safe String sql) {…}
}

class ShoppingCartControl extends HttpRequestHandler {
  @pausable
  void handle(@safe HttpRequest req, @cuttable HtmlMsg reply){
    DatabaseConnection con = pool.get();
    Results r = query(con, "select * from carts where ...");
    reply.html = "<h2>Cart</h2><table>"+r.print()+"</table>";
    pool.put(con);
  }
}

class SearchResultsControl extends HttpRequestHandler {
  @pausable
  void handle(@safe HttpRequest req, @cuttable HtmlMsg reply) {
    DatabaseConnection con = pool.get();
    Results r = query(con, "select * from products where ...");
    reply.html = "<h2>Search results</h2><div>" + r.print() + "</div>";
    pool.put(con);
  }
}
```

Messages are public tree structures

Method may suspend

Actor entry point

May be structurally modified but not sent

May only be read

- *r is free*
- *r is invalid*
- *r is free*
- *r is invalid*

May be sent / destructively read

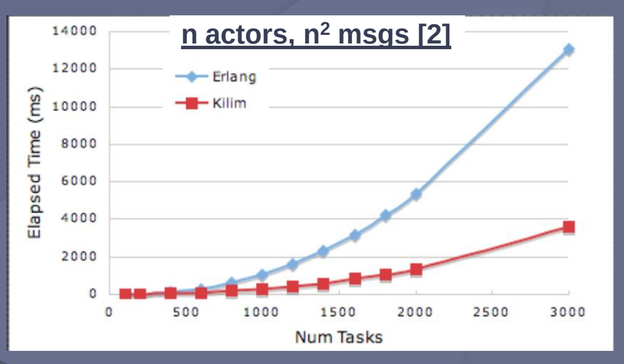
- *reply is free, req is cuttable*
- *reply is invalid, req is invalid*

Send message / transfer ownership

Receive message / get ownership

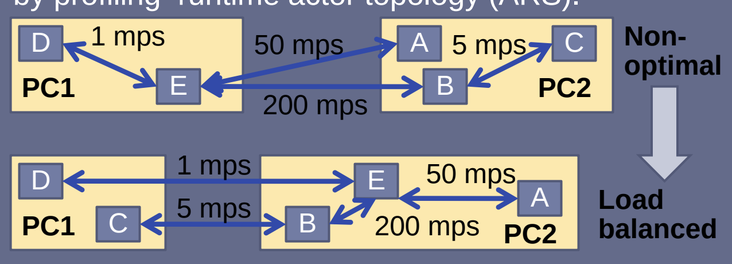
- *con is free*
- *con is invalid*

- **Example: renders 2 parts of webpage in parallel**
  - When blocking on DB another part of page/request can be handled via fast task switching.
  - Database connections shared by a queue and linear ownership passing (no locks).
- **Very fast: 4x faster than Erlang, 100x Java threads!**



Massively Distributed Computation

- **"Internet Operating System": Middleware for internet scale distributed computing with SALSA actors (and MPI) [5].**
- Implements: Adaptive decentralized load balancing by profiling runtime actor topology (ARS).

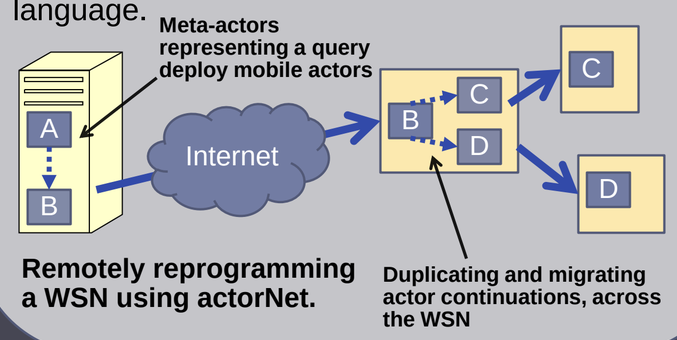


- ARS gave 10x performance increase over round robin on sparse topology benchmark.

(mps = messages per second)

Wireless Sensor Networks

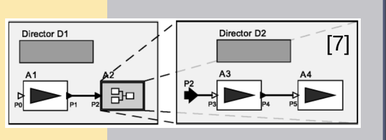
- **"ActorNet": Mobile agent platform for WSNs via a custom Scheme interpreter [3].**
- Provides: Actor migration, Virtual memory, Garbage collection, and Multitasking for Mica2.
- Enables: program portability, remote code deployment, & reconfiguration to conserve energy.
- Used as the basis of the uQueries domain specific language.



Signal Processing

- **"CAL": Domain specific actor language for signal processing algorithms [4].**
- Automatic code generation of C and VHDL via recent **CAL2C** and **CAL2HDL** generators [6].
- Very concise, more flexible, architecture independent implementation => portable.
- Visual & hierarchical design via Ptolemy II [7].

```
actor sum[T] (T init) T A ==> T B:
  T sum := init;
  action [a] ==> [sum] do
    sum := sum + a;
  endaction
endactor      [8]
```



- MPEG4 Decoder: 4000lines CAL vs 15000 VHDL, **1.6x faster** performance & 4x faster development than handwritten VHDL.
- Used by ISO for new MPEG "RCV" codec.

Conclusions

- **Kilim** demonstrates:
  - Actor continuation passing allows **fast task switching**
  - **Linear type systems** can enable resource sharing & **fast message passing**.
- **IOS & ActorNet** exploit:
  - Actor migration to provide adaptive **mobile agent based programming**.
- **CAL** shows actor-oriented programming can be
  - very concise, logical hierarchical structure, intuitive concurrency, and **allows efficient multiplatform code generation**.
- As architectures are becoming more distributed and more abstraction is required, the predicted benefits of the Actor model are beginning to be realized over 30 years after its conception [9].

[1] C. Hewitt, "Viewing control structures as patterns of passing messages," Massachusetts Institute of Technology, Tech. Rep., 1976.
[2] S. Srinivasan and A. Mycroft, "Kilim: Isolation-typed actors for java (a million actors, safe zero-copy communication)," 2008.
[3] K. M. YoungMin Kwon, Sameer Sundresh and G. Agha, "Actornet:
An actor platform for wireless sensor networks," 2006.
[4] J. Eker and J. W. Janneck, "Cal language report: Specification of the cal actor language," University of California at Berkeley, Tech. Rep. UCB/ERL M03/48, 2003
[5] K. E. Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela, "The internet operating system: Middleware for adaptive distributed computing," in International Journal of High Performance Computing Applications (IJHPCA), Special Issue on Scheduling Techniques for Large-Scale Distributed Platforms, 2006.
[6] C. L. et al., "Dataflow/actor-oriented language for the design of complex signal processing systems," in In Proceedings of Conference on Design and Architectures for Signal and Image Processing (DASIP 2008), Bruxelles: Belgique (2008), 2008.
[7] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neunendorffer, S. Sachs, Y. Xiong, "Taming Heterogeneity—The Ptolemy Approach," in proceedings of the IEEE, 91(1):127-144, Jan 2003.
[8] Y. Zhao, "An introduction to the CAL actor language," University of Salzburg, May 2002.
[9] T. Aubrey-Jones, "Resurrecting Actors: New Applications of an Old Paradigm in Engineering and Science", University of Southampton, UK, 2009.

UNIVERSITY OF Southampton
School of Electronics and Computer Science