Open Distributed Systems

- Addition of new components.
- Replacement of existing components.
- Changes in interconnections.

Actor Configurations

model open system components:
- set of individually named actors.
- messages "on-route".
- interface to environment:
  - receptionists
  - external actors
Synchronous vs Asynchronous Communication

- Tn-Calculus (and other process algebras such as CCS, CSP) take synchronous communication as a primitive.

- Actors assume asynchronous communication is more primitive.
Communication Medium

- In π-Calculus, channels are explicitly modelled. Multiple processes can share a channel, potentially causing interference.

- In the actor model, the communication medium is not explicit. Actors (active objects) are first-class, history-sensitive entities with an explicit identity used for communication.
FAIRNESS

The actor model theory assumes fair computations:

1. message delivery is guaranteed.
2. individual actor computations are guaranteed to progress.

Fairness is very useful for reasoning about equivalences of actor programs but can be hard/expensive to guarantee; in particular when distribution and failures are considered.
Programming Languages influenced by π-calculus and Actors.

- Scheme '75
- Actl '87
- Acorn '87
- Rosette '89
- Oblig '94
- Erlang '93
- ABCL '90
- SALSA '99
- Amber '86
- Facile '89
- CML '91
- Pict '94
- Nomadic Pict '99
- TOCAML '99
Agha, Mason, Smith & Talcott

1. Extend a functional language (\(\lambda\text{-calculus}\) + iss + pairs) with actor primitives

2. Define an operational semantics for actor configurations.

3. Study various notions of equivalence of actor expressions and configurations.

4. Assume fairness:
   - guaranteed message delivery.
   - individual actor progress.
\[\lambda\text{-calculus}\]

Syntax

\[e ::= v \mid \lambda v.e \mid (e e)\]

Example

\[(\varepsilon x.x)\ 5\]

\[5 \xrightarrow{\text{\(\pi\)}} x\{5/x\}\]

\([5/x]\ x\]

\[\xi\]
PAIRING PRIMITIVES

\[ \text{pr}(x,y) \] returns a pair containing \( x \) \& \( y \).

\[ \text{ispr}(x) \] returns \( \top \) if \( x \) is a pair; \( \bot \) otherwise.

\[ 1\text{st} \left( \text{pr}(x,y) \right) = x \] 1st returns The first value of a pair.

\[ 2\text{nd} \left( \text{pr}(x,y) \right) = y \] 2nd returns The second value.
**Actor Primitives**

\[
\text{send}(a, v) \quad \text{sends value } v \text{ to actor } a.
\]

\[
\text{letactor}(x := b) \quad \text{creates a new actor with behavior } b, \text{ and binds variable } x \text{ in expression } e \text{ to the address of the newly created actor.}
\]

\[
\text{become}(b) \quad \text{creates an anonymous actor to carry out the rest of the computation, and changes behavior to } b.
\]
Actor Language Examples

\[ b5 = \text{rec}(\lambda y. \lambda x. \text{seg (send (x, 5), become (y)))} \]

receives an actor name \( x \) and sends
the number 5 to that actor, then it
becomes the same behavior \( y \).

An actor

Sample Usage

\[ e = \text{letactor} \{ z := b5 \} \text{ send (z, a) } \]

A Sink

\[ \text{sink} = \text{rec}(\lambda b. \lambda m. \text{become (b)}) \]

an actor that disregards all messages.
Reference Cell in Actor Language

\[ B_{cell} = \text{rec}(\lambda b. \lambda c. \lambda m. \]
\[ \quad \text{if } \text{get}(m), \]
\[ \quad \quad \text{seq}(\text{become}(b(c)), \]
\[ \quad \quad \quad \text{send}(\text{content}(m), c)) \]
\[ \quad \text{if } \text{set}(m), \]
\[ \quad \quad \text{become}(b(\text{content}(m))) \]
\[ \quad \quad \text{become}(b(c))))))) \]

Using the cell:

\[ \text{letactor } \{ a := B_{cell}(0) \} e \]

\[ e = \text{seq}(\text{send}(a, \text{mkset}(3)), \]
\[ \text{send}(a, \text{mkset}(5)), \]
\[ \text{send}(a, \text{mkget}(e))) \]
Exercises

1. Write `get?`, `cust set?`, `contents`, `mkset`, `mkget` to complete the reference cell example in the AMST actor language.

2. Modify `Bcell` to notify a customer when the cell value is updated (such as in the TT-calculus cell example).
DINING PHILosophers in ANST ACtor LAnguage

\[ B_{ph} = \text{rec}(A_b, A_l, A_r, \lambda self. \text{acks} \lambda m. \]

\[ \text{if } (\text{eq?}(m, \text{self}), \]

\[ \text{if } (\text{eq?}(\text{acks}, 0), \]

\[ \text{become} (b(2, r, \text{self}, \text{acks} + 1))), \]

\[ \text{send}(l, \text{mkrelease}(\text{self})), \]

\[ \text{send}(r, \text{mkrelease}(\text{self})), \]

\[ \text{become} (b(2, l, \text{self}, \Theta)), \]

\[ \text{send}(l, \text{mkpickup}(\text{self})), \]

\[ \text{send}(r, \text{mkpickup}(\text{self}))), \]

\[ \text{become} (b(2, l, \text{self}, \text{acks}))), \]
DINING PHILOSOPHERS IN AMST (2)

\(B\text{fork} = \text{rec}\ (\lambda h. \lambda w. \lambda m.\)
\[\begin{align*}
&\text{if (pickup? (m),}\notag \\
&\quad \text{if (eq? (h, nil),}\notag \\
&\quad \quad \text{seg (send (phil(m), phil(m)),}\notag \\
&\quad \quad \quad \text{become (b(phil(m), nil)))},\notag \\
&\quad \quad \text{become (b(h, phil(m))))},\notag \\
&\text{if (release? (m),}\notag \\
&\quad \text{if (eq? (w, nil),}\notag \\
&\quad \quad \text{become (b(nil, nil))},\notag \\
&\quad \quad \text{seg (send (w,w),}\notag \\
&\quad \quad \quad \text{become(b(w,nil))))},\notag \\
&\quad \quad \text{become(b(h,w))))}\notag \\
\end{align*}\]
DINING PHILOSOPHERS in AMST (3)

Using the definitions to set up a 2-phil's dining table:

\[
\begin{array}{c}
\text{Defactor} \{ & f_1 := \text{Bpark}(\text{nil}, \text{nil}), \\
& f_2 := \text{Bpark}(\text{nil}, \text{nil}), \\
& P_1 := \text{Bpml}(f_1, f_2, P, 0), \\
& P_2 := \text{Bpml}(f_2, f_1, P, 0) \} \in \\
\end{array}
\]

where \( e \) is defined as:

\[
e = \text{seq}(\text{send}(f_1, \text{mkpickup}(P_1)), \\
\quad \text{send}(f_2, \text{mkpickup}(P_1)), \\
\quad \text{send}(f_1, \text{mkpickup}(P_2)), \\
\quad \text{send}(f_2, \text{mkpickup}(P_2)))
\]
Dining Philosophers in AMST (4)

Auxiliary definitions:

\[ mk_{\text{pickup}} = \lambda p. \text{pr}(\text{"pickup"}, p) \]
\[ mk_{\text{release}} = \lambda p. \text{pr}(\text{"release"}, p) \]
\[ \text{pickup?} = \lambda m. \text{if } (\text{ispr?}(m), \text{eq?}(\text{1st}(m), \text{"pickup"}), \text{nil}) \]
\[ \text{release?} = \lambda m. \text{if } (\text{ispr?}(m), \text{eq?}(\text{1st}(m), \text{"release"}), \text{nil}) \]
\[ phil = \lambda m. \text{if } (\text{pickup?}(m), \text{2nd}(m), \text{nil}) \]
Actor Garbage Collection

- 11
- 12

Blocked (Idle)

Unblocked (non-idle)

Root

Diagram with nodes connected by arrows.