Smooth Orchestrators

Cosimo Laneve, Luca Padovani

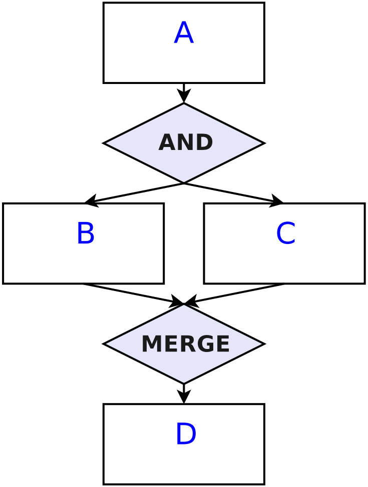
University of Bologna, University of Urbino

29 march 2006

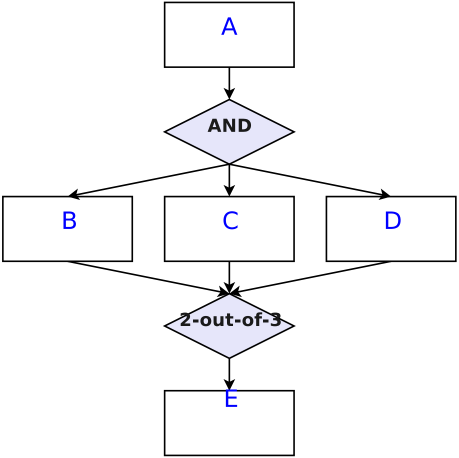
Summary

- Background
- Formal development
- Implementation
- Extensions and concluding remarks

Well-known workflow patterns: synchronization



Well-known workflow patterns: *n*-out-of-*m*



Synchronization patterns

Define a primitive *construct* that models synchronization patterns
Join-patterns in JoCaml:

```
let create_ref(y0) =
  let state(y) | get() = state(y) | reply y to get
  and state(y) | put(new_y) = state(new_y) | reply to put in
  state(y0) | reply get,put
;;
```

Remark: atomic reduction

Similar construct, different context

A synchronization pattern implemented in the join calculus is

- *permanent*
- *closed*

Orchestration of services is not necessarily permanent:

- ephemeral synchronization
- extending existing services

Similar construct, different context

A synchronization pattern implemented in the join calculus is

- *permanent*
- *closed*

Orchestration of services is not necessarily permanent:

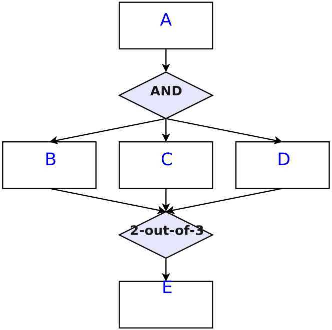
- ephemeral synchronization
- extending existing services

π with orchestrators

Asynchronous π-calculus with *orchestrators*:

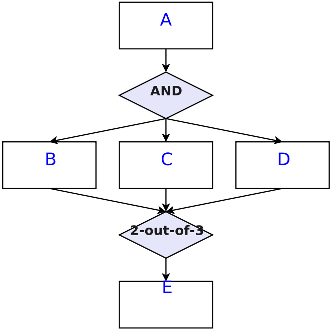
P	$::=$		**processes**
		0	(nil)
	$\mid$	$\overline{x}[\widetilde{u}]$	(output)
	$\mid$	$\sum_{i \in I} J_i \triangleright P_i$	(orchestrator)
	$\mid$	$(x)P$	(new)
	$\mid$	$P \mid P$	(parallel)
	$\mid$	$!P$	(replication)
J	$::=$		**join patterns**
	$\mid$	$x(\widetilde{u})$	(input)
	$\mid$	$J \,\&\, J$	(join)

Example: *n*-out-of-*m*



$$
\begin{aligned}
& a\_end(v) \triangleright \\
& \qquad \overline{b\_start}[v] \mid \overline{c\_start}[v] \mid \overline{d\_start}[v] \\
\mid & \qquad b\_end(x) \,\&\, c\_end(y) \triangleright \overline{e\_start}[x, y] \\
+ & \qquad b\_end(x) \,\&\, d\_end(z) \triangleright \overline{e\_start}[x, z] \\
+ & \qquad c\_end(y) \,\&\, d\_end(z) \triangleright \overline{e\_start}[y, z]
\end{aligned}
$$

Example: *n*-out-of-*m*



$$
\begin{array}{ll}
a\_end(v) \triangleright & \\
 & \overline{b\_start}[v] \mid \overline{c\_start}[v] \mid \overline{d\_start}[v] \\
\mid & b\_end(x) \,\&\, c\_end(y) \triangleright \overline{e\_start}[x,y] \\
+ & b\_end(x) \,\&\, d\_end(z) \triangleright \overline{e\_start}[x,z] \\
+ & c\_end(y) \,\&\, d\_end(z) \triangleright \overline{e\_start}[y,z]
\end{array}
$$

Issues

Linearity is hard to enforce statically:

$$\overline{x}[a, a] \mid x(u, v) \triangleright u(y) \,\&\, v(z) \triangleright P \rightarrow a(y) \,\&\, a(z) \triangleright P\{a/u, a/v\}$$

Global consensus:

Take $x(u) \,\&\, y(v) \triangleright P$ located at ℓ

- If x and y are *not* located at ℓ this reduction requires *non-local* – global – information
- Migrating the process does not help either (and who likes mobile agents anyway?)

These are non-issues in the join-calculus:

- joined channels are *fresh*. . .
- . . . hence they are co-located

Issues

Linearity is hard to enforce statically:

$$\overline{x}[a, a] \mid x(u, v) \triangleright u(y) \,\&\, v(z) \triangleright P \rightarrow a(y) \,\&\, a(z) \triangleright P\{a/u, a/v\}$$

Global consensus:

Take $x(u) \,\&\, y(v) \triangleright P$ located at ℓ

- If x and y are *not* located at ℓ this reduction requires *non-local* – global – information
- Migrating the process does not help either (and who likes mobile agents anyway?)

These are non-issues in the join-calculus:

- joined channels are *fresh*. . .
- . . . hence they are co-located

Issues

Linearity is hard to enforce statically:

$$\overline{x}[a, a] \mid x(u, v) \triangleright u(y) \,\&\, v(z) \triangleright P \rightarrow a(y) \,\&\, a(z) \triangleright P\{a/u, a/v\}$$

Global consensus:

Take $x(u) \,\&\, y(v) \triangleright P$ located at ℓ

- If x and y are *not* located at ℓ this reduction requires *non-local* – global – information
- Migrating the process does not help either (and who likes mobile agents anyway?)

These are non-issues in the join-calculus:

- joined channels are *fresh*. . .
- . . . hence they are co-located

Our solution: co-location constraints

P	$::=$		**processes**
		$\ldots$	
	$\mid$	$(x \mathbin{@} y)P$	(new)
J	$::=$		**join patterns**
	$\mid$	$x(\widetilde{u} \mathbin{@} \widetilde{v})$	(input)
	$\mid$	$J \,\&\, J$	(join)

- $(x@y)P$ means "create x at the same location as y"
- $(x@x)P$ means "create x at whatever location"
- $x(u@u, v@v) \triangleright P$ means "receive u and v no matter what their location is"
- $x(u@u, v@u) \triangleright P$ means "receive u and v if they are co-located"

Our solution: co-location constraints

P	$::=$		**processes**
		$\dots$	
	$\mid$	$(x \mathbin{@} y)P$	(new)
J	$::=$		**join patterns**
	$\mid$	$x(\widetilde{u} \mathbin{@} \widetilde{v})$	(input)
	$\mid$	$J \mathbin{\&} J$	(join)

- $(x@y)P$ means "create x at the same location as y"
- $(x@x)P$ means "create x at whatever location"
- $x(u@u, v@v) \triangleright P$ means "receive u and v no matter what their location is"
- $x(u@u, v@u) \triangleright P$ means "receive u and v if they are co-located"

Our solution: co-location constraints

P	$::=$		**processes**
		$\dots$	
	$\mid$	$(x \,@\, y)P$	(new)
J	$::=$		**join patterns**
	$\mid$	$x(\widetilde{u} \,@\, \widetilde{v})$	(input)
	$\mid$	$J \,\&\, J$	(join)

- $(x@y)P$ means "create x at the same location as y"
- $(x@x)P$ means "create x at whatever location"
- $x(u@u, v@v) \triangleright P$ means "receive u and v no matter what their location is"
- $x(u@u, v@u) \triangleright P$ means "receive u and v if they are co-located"

Our solution: co-location constraints

P	$::=$		**processes**
		$\dots$	
	$\mid$	$(x \mathbin{@} y)P$	(new)
J	$::=$		**join patterns**
	$\mid$	$x(\widetilde{u} \mathbin{@} \widetilde{v})$	(input)
	$\mid$	$J \mathbin{\&} J$	(join)

- $(x@y)P$ means "create x at the same location as y"
- $(x@x)P$ means "create x at whatever location"
- $x(u@u, v@v) \triangleright P$ means "receive u and v no matter what their location is"
- $x(u@u, v@u) \triangleright P$ means "receive u and v if they are co-located"

Co-location relation:

(BASE)
$$(\widetilde{x} \mathbin{@} \widetilde{y})(u \mathbin{@} v) \vdash u \frown v$$

(LIFT)
$$\frac{\widetilde{x} \mathbin{@} \widetilde{y} \vdash u \frown v \quad u, v \neq z}{(\widetilde{x} \mathbin{@} \widetilde{y})(z \mathbin{@} z') \vdash u \frown v}$$

Reduction:

$$\frac{\widetilde{z} \mathbin{@} \widetilde{y} \vdash a \frown b}{(\widetilde{z} \mathbin{@} \widetilde{y})\left(\overline{x}[a] \mid \overline{y}[b] \mid x(u \mathbin{@} u) \,\&\, y(v \mathbin{@} u) \triangleright P\right) \to (\widetilde{z} \mathbin{@} \widetilde{y})P\{^{a,\,b}/_{u,\,v}\}}$$

Co-location relation:

(BASE)

$$(\widetilde{x} \mathbin{@} \widetilde{y})(u \mathbin{@} v) \vdash u \frown v$$

(LIFT)

$$\frac{\widetilde{x} \mathbin{@} \widetilde{y} \vdash u \frown v \quad u, v \neq z}{(\widetilde{x} \mathbin{@} \widetilde{y})(z \mathbin{@} z') \vdash u \frown v}$$

Reduction:

$$\frac{\widetilde{z} \mathbin{@} \widetilde{y} \vdash a \frown b}{(\widetilde{z} \mathbin{@} \widetilde{y})\Big(\overline{x}[a] \mid \overline{y}[b] \mid x(u \mathbin{@} u) \mathbin{\&} y(v \mathbin{@} u) \triangleright P\Big) \to (\widetilde{z} \mathbin{@} \widetilde{y})P\{^{a,\,b}/_{u,\,v}\}}$$

Checking co-location

The process

$$w(x@x, y@y) \triangleright x(u@u) \,\&\, y(v@u) \triangleright P$$

raises a runtime error if provided with a message

$$\overline{w}[c, d]$$

where c and d are not co-located

- the message on w is lost forever
- we want to check co-location *statically* through a type system

Checking co-location

(NIL)
$$\Lambda \vdash 0$$

(OUTPUT)
$$\Lambda \vdash \overline{x}[\widetilde{u}]$$

(PAR)
$$\frac{\Lambda \vdash P \quad \Lambda \vdash Q}{\Lambda \vdash P \mid Q}$$

(BANG)
$$\frac{\Lambda \vdash P}{\Lambda \vdash !P}$$

(ORCH)
$$\frac{\Lambda(u@u)(v@u) \vdash P \qquad \Lambda \vdash x \frown y}{\Lambda \vdash x(u@u) \,\&\, y(v@u) \triangleright P}$$

(NEW)
$$\frac{\Lambda(x @ y) \vdash P}{\Lambda \vdash (x @ y)P}$$

A process P is *distributable* if $\varepsilon \vdash P$

Checking co-location

$$\frac{}{\Lambda \vdash 0}\ (\text{NIL}) \qquad \frac{}{\Lambda \vdash \overline{x}[\widetilde{u}]}\ (\text{OUTPUT}) \qquad \frac{\Lambda \vdash P \quad \Lambda \vdash Q}{\Lambda \vdash P \mid Q}\ (\text{PAR}) \qquad \frac{\Lambda \vdash P}{\Lambda \vdash !P}\ (\text{BANG})$$

$$\frac{\Lambda(u@u)(v@u) \vdash P \qquad \Lambda \vdash x \frown y}{\Lambda \vdash x(u@u)\,\&\,y(v@u) \triangleright P}\ (\text{ORCH}) \qquad \frac{\Lambda(x @ y) \vdash P}{\Lambda \vdash (x @ y)P}\ (\text{NEW})$$

A process P is *distributable* if $\varepsilon \vdash P$

Subject reduction

Distributable processes reduce to distributable processes

Theorem (subject reduction): If

- $(\widetilde{x} \,@\, \widetilde{y}) \vdash P$, and
- $(\widetilde{x} \,@\, \widetilde{y})P \rightarrow (\widetilde{x} \,@\, \widetilde{y})Q$

then

- $(\widetilde{x} \,@\, \widetilde{y}) \vdash Q$

Lemma (substitution): Substitution cannot be defined for single names only

Consider

$$(a \,@\, a)(u \,@\, u)(v \,@\, u) \vdash u \,\&\, v \triangleright 0 \qquad \{^a/_v\}$$

leads to

$$(a \,@\, a)(u \,@\, u) \nvdash u \,\&\, a \triangleright 0$$

Subtitutions must preserve co-location

Subject reduction

Distributable processes reduce to distributable processes

Theorem (subject reduction): If

- $(\widetilde{x} @ \widetilde{y}) \vdash P$, and
- $(\widetilde{x} @ \widetilde{y})P \rightarrow (\widetilde{x} @ \widetilde{y})Q$

then

- $(\widetilde{x} @ \widetilde{y}) \vdash Q$

Lemma (substitution): Substitution cannot be defined for single names only

Consider

$$(a @ a)(u @ u)(v @ u) \vdash u \,\&\, v \triangleright 0 \qquad \{^{a}/_{v}\}$$

leads to

$$(a @ a)(u @ u) \not\vdash u \,\&\, a \triangleright 0$$

Subtitutions must preserve co-location

The smoothness restriction

We decouple the orchestrator from the continuation through the $[\![\cdot]\!]$ encoding

$$[\![\sum_{i \in I} J_i \triangleright P_i]\!] = (z_i{}^{i \in I})\Big(\overbrace{\sum_{i \in I} J_i \triangleright \overline{z_i}[\widetilde{u}_i]}^{\text{smooth orchestrator}} \mid \underbrace{\prod_{i \in I} z_i(\widetilde{u}_i) \triangleright [\![P_i]\!]}_{\text{continuation}} \Big)$$

The smooth orchestrator is now free to migrate

Proposition (encoding correctness): P is barbed congruent to $[\![P]\!]$

The smoothness restriction

We decouple the orchestrator from the continuation through the $[\![\cdot]\!]$ encoding

$$[\![\sum_{i \in I} J_i \triangleright P_i]\!] = (z_i{}^{i \in I}) \Big(\overbrace{\sum_{i \in I} J_i \triangleright \overline{z_i}[\widetilde{u_i}]}^{\text{smooth orchestrator}} \quad | \quad \underbrace{\prod_{i \in I} z_i(\widetilde{u_i}) \triangleright [\![P_i]\!]}_{\text{continuation}} \Big)$$

The smooth orchestrator is now free to migrate

Proposition (encoding correctness): P is barbed congruent to $[\![P]\!]$

Implementation issues

Size of an orchestrator:

$$x_1(\widetilde{u}_1 \,@\, \widetilde{v}_1) \,\&\, \cdots \,\&\, x_n(\widetilde{u}_n \,@\, \widetilde{v}_n) \triangleright \overline{z}[\widetilde{u}_1 \cdots \widetilde{u}_n]$$

It may be encoded as

- a vector with $n+1$ names $x_1, \cdots, x_n, z$
- a vector of $k_1 + \cdots + k_n$ values, where $k_i = |\widetilde{u}_i|$:
  - the integer value j at position h indicates that the j-th and h-th bound names must be co-located
  - the constant c at position h indicates that the h-th bound name must be co-located with c

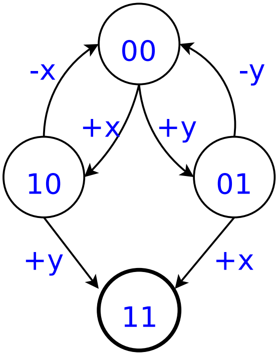
Co-location check: it basically amounts to comparing IP addresses

Compiling simple orchestrators

Turn an archestrator into a finite-state automaton (*c.f. Le Fessant, Maranget*):

$$x(u) \,\&\, y(v) \triangleright \overline{z}[uv]$$

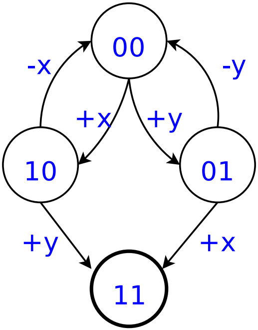
is compiled into



Compiling orchestrators with co-location constraints I

$$x({\color{red}u@u}, {\color{red}v@u}) \,\&\, y({\color{red}w@w}) \triangleright \overline{z}[uvw]$$

is compiled into



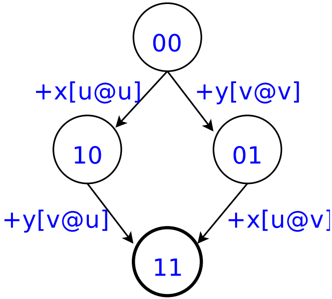
- ${\color{blue}+x}$ = "there is a message $\overline{x}[a, b]$ such that a and b are co-located"
- ${\color{blue}-x}$ = "there is **no** message $\overline{x}[a, b]$ such that a and b are co-located"

Compiling orchestrators with co-location constraints II

Consider

$$x(u@u) \,\&\, y(v@u) \triangleright P$$

Reception of a message on y *depends* on the message received on x.



What if the message on y arrives first?
We rewrite the orchestrator thus:

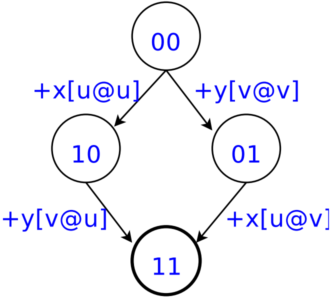
$$y(v@v) \,\&\, x(u@v) \triangleright P$$

Compiling orchestrators with co-location constraints II

Consider

$$x(u@u) \,\&\, y(v@u) \triangleright P$$

Reception of a message on y *depends* on the message received on x.



What if the message on y arrives first?

We rewrite the orchestrator thus:

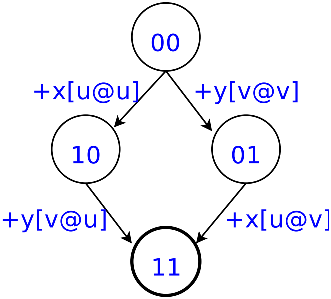
$$y(v@v) \,\&\, x(u@v) \triangleright P$$

Compiling orchestrators with co-location constraints II

Consider

$$x(u@u) \,\&\, y(v@u) \triangleright P$$

Reception of a message on y *depends* on the message received on x.



What if the message on y arrives first?
We rewrite the orchestrator thus:

$$y(v@v) \,\&\, x(u@v) \triangleright P$$

Compiling orchestrators with co-location constraints III

Non-linearity increases significantly the complexity of matching

Consider

$$x(u@u, v@u, w@w) \,\&\, x(a@a, b@b, c@b) \triangleright P$$

- $\overline{x}[z, z, z]$ can be interpreted as either

$$\overline{x}[z, z, z] \qquad \text{or} \qquad \overline{x}[z, z, z]$$

- Upon arrival of $\overline{x}[z, z, z]$ whichever transition is chosen might be the *wrong* one
- Neither of the two patterns is "more specific" than the other, they cannot be sorted

Compiling orchestrators with co-location constraints III

Non-linearity increases significantly the complexity of matching

Consider

$$x(u@u, v@u, w@w) \,\&\, x(a@a, b@b, c@b) \triangleright P$$

- $\overline{x}[z, z, z]$ can be interpreted as either

$$\overline{x}[z, z, z] \qquad \text{or} \qquad \overline{x}[z, z, z]$$

- Upon arrival of $\overline{x}[z, z, z]$ whichever transition is chosen might be the *wrong* one
- Neither of the two patterns is "more specific" than the other, they cannot be sorted

Compiling orchestrators with co-location constraints III

Non-linearity increases significantly the complexity of matching

Consider

$$x(u@u, v@u, w@w) \,\&\, x(a@a, b@b, c@b) \triangleright P$$

- $\overline{x}[z, z, z]$ can be interpreted as either

$$\overline{x}[z, z, z] \qquad \text{or} \qquad \overline{x}[z, z, z]$$

- Upon arrival of $\overline{x}[z, z, z]$ whichever transition is chosen might be the *wrong* one
- Neither of the two patterns is "more specific" than the other, they cannot be sorted

Compiling orchestrators: when and where?

- The smooth orchestrator is usually small in size
- The automaton is bigger, and it depends on the cardinality of name occurrences which are not known until runtime

Two strategies:

- eager compilation: bigger messages, needs patching
- delayed compilation: smaller messages, burden on the receiver

Compiling orchestrators: when and where?

- The smooth orchestrator is usually small in size
- The automaton is bigger, and it depends on the cardinality of name occurrences which are not known until runtime

Two strategies:

- eager compilation: bigger messages, needs patching
- delayed compilation: smaller messages, burden on the receiver

Extensions and conclusion

Further investigations:

- Stronger type system
  - eliminate runtime co-location checks
  - technically challenging (dangerous variables)

$$x(u@\alpha) \mathbin{\&} y(v@\alpha) \triangleright P$$

  x and y cannot be treated polymorphically w.r.t. α because of their dependency

- Expressivity (workflow patterns)

PiDuce prototype available at

`http://www.cs.unibo.it/PiDuce/`

(C# implementation)