Behavioural Types and Local-First Principles for Swarms
Swarming up with local-first principles

Emilio Tuosto @ GSSI

Joint work with

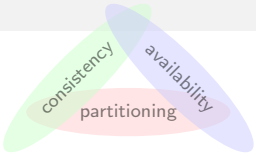
Roland Kuhn @ ??? and Hernán Melgratti @ UBA

APM @ Turin
17 Aprile, 2026

Take-away message

Realising fully distributed computation is challenging

Take-away message



Realising fully distributed computation is challenging

- Distributed systems are limited by the CAP theorem
  - The computational state is scattered
  - Components "retrieve" the state through asynchronous communications

Take-away message

Realising fully distributed computation is challenging

- Distributed systems are limited by the CAP theorem
  - The computational state is scattered
  - Components "retrieve" the state through asynchronous communications
- We 're scared by inconsistencies

Take-away message

Realising fully distributed computation is challenging

- Distributed systems are limited by the CAP theorem
  - The computational state is scattered
  - Components "retrieve" the state through asynchronous communications
- We 're scared by inconsistencies
- Beaten tracks
  - "standard" behavioural types "work" but, they're partial to C
  - conflict-free replicated data types work, but hard to be found

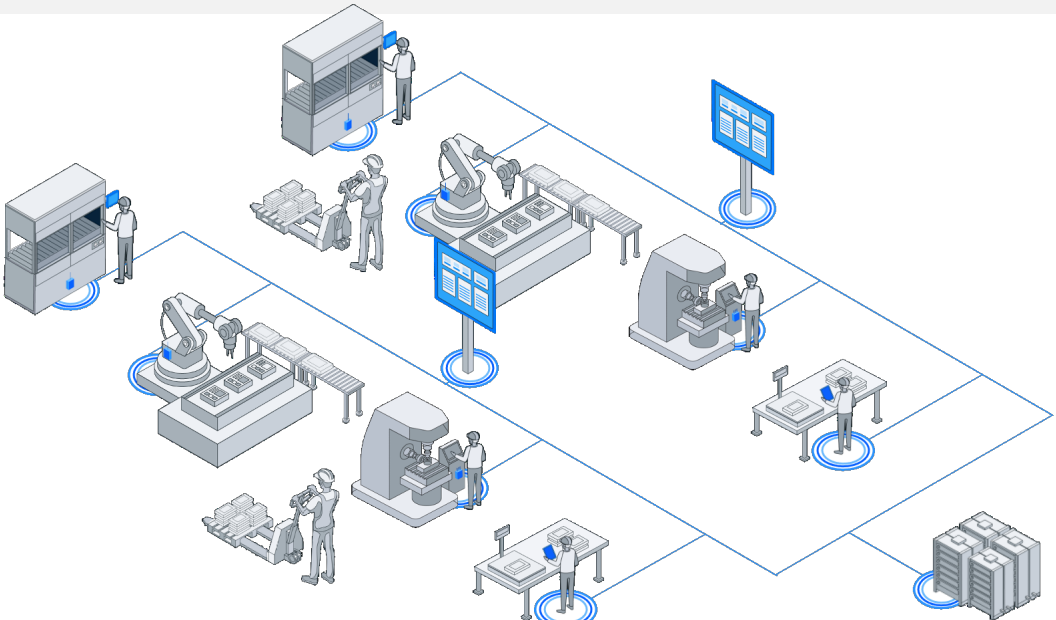
Take-away message

Realising fully distributed computation is challenging

- Distributed systems are limited by the CAP theorem
  - The computational state is scattered
  - Components "retrieve" the state through asynchronous communications
- We 're scared by inconsistencies
- Beaten tracks
  - "standard" behavioural types "work" but, they're partial to C
  - conflict-free replicated data types work, but hard to be found
- So, let's go coordination-free
  - swarm protocols are global specs of swarms
  - allow speculative progress and tolerate inconsistencies
  - to be taken care of eventually

– Context –

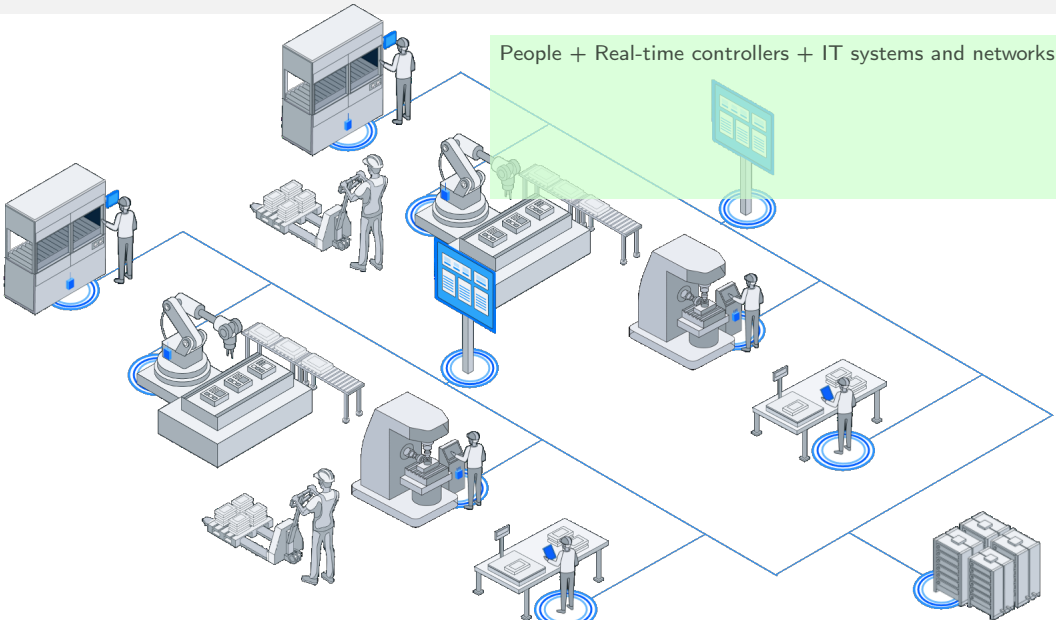
An execution model for a collaborative environment



(the pictures are courtesy of Actyx AG)

An execution model for a collaborative environment

People + Real-time controllers + IT systems and networks:

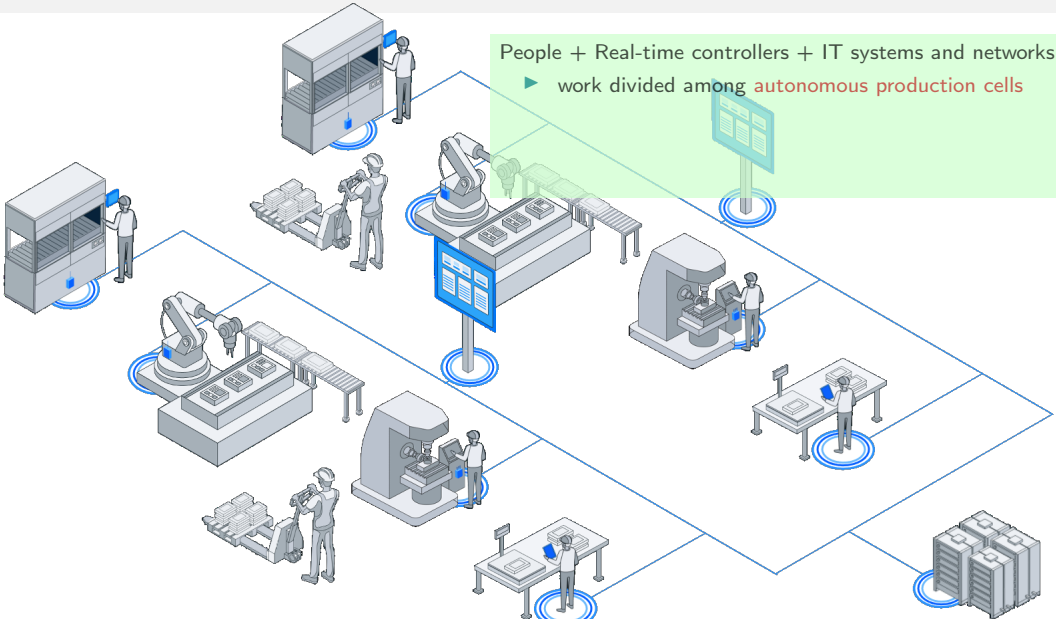


(the pictures are courtesy of Actyx AG)

An execution model for a collaborative environment

People + Real-time controllers + IT systems and networks:

- work divided among autonomous production cells

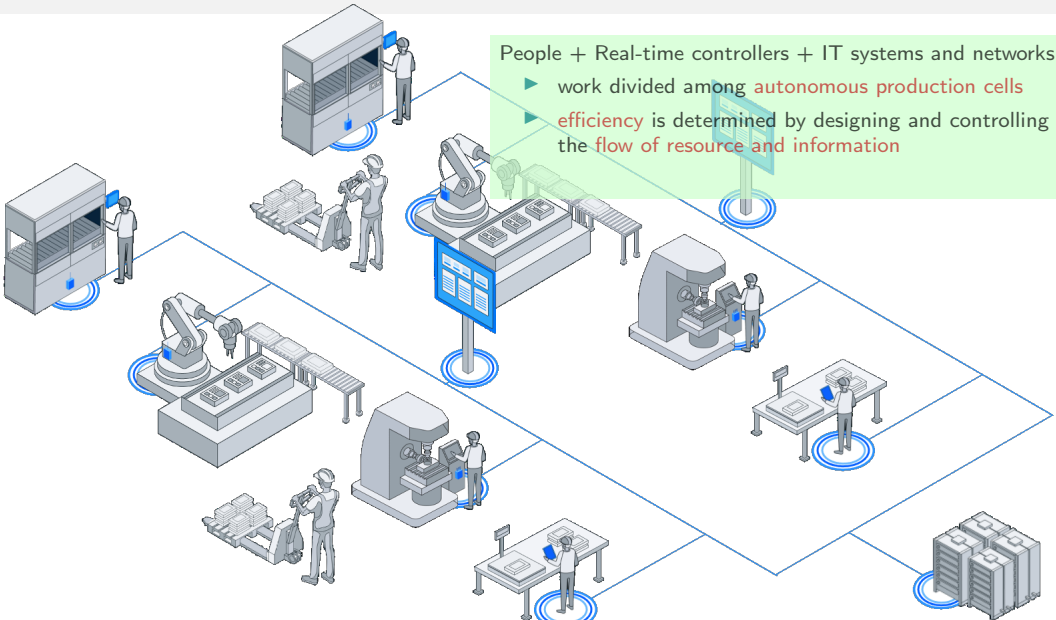


(the pictures are courtesy of Actyx AG)

An execution model for a collaborative environment

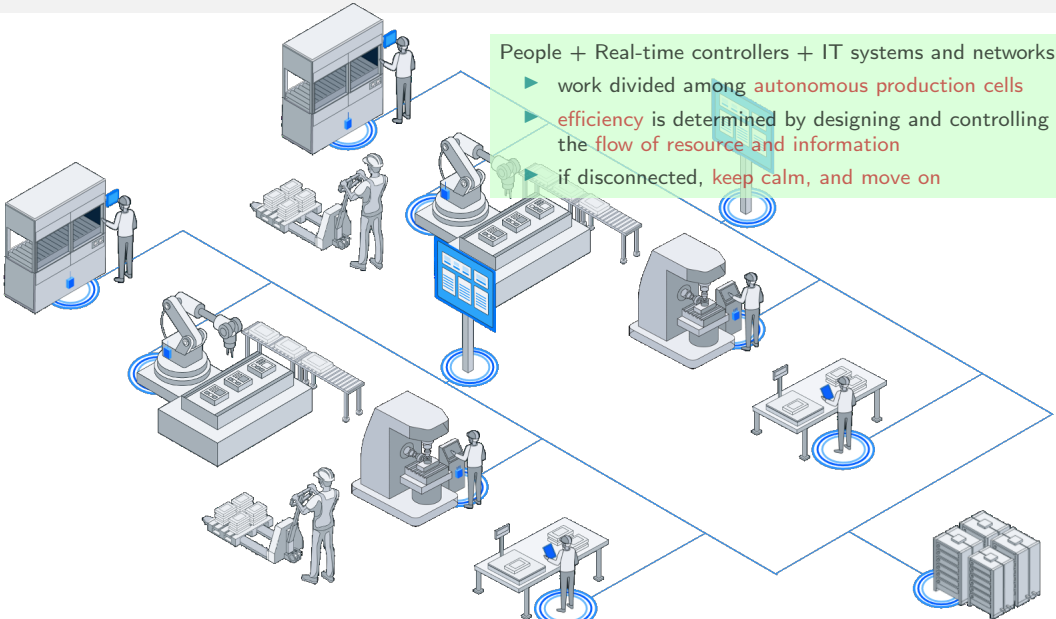
People + Real-time controllers + IT systems and networks:

- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information



(the pictures are courtesy of Actyx AG)

An execution model for a collaborative environment



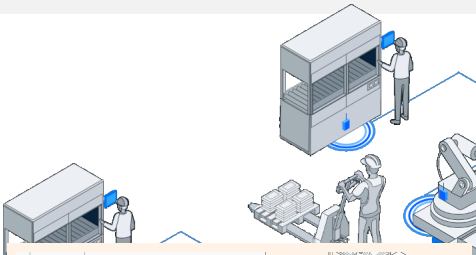
People + Real-time controllers + IT systems and networks:

- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

(the pictures are courtesy of Actyx AG)

An execution model for a collaborative environment

(the pictures are courtesy of Actyx AG)



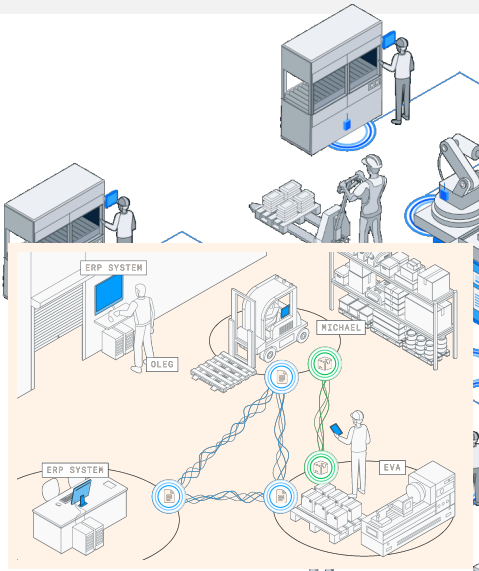
People + Real-time controllers + IT systems and networks:

- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

Execution model

An execution model for a collaborative environment

(the pictures are courtesy of Actyx AG)



People + Real-time controllers + IT systems and networks:

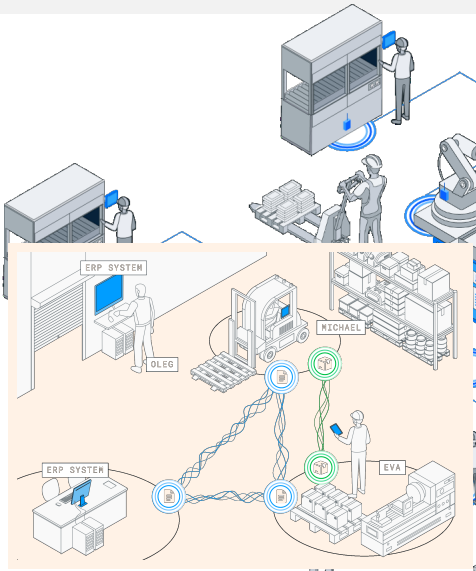
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

Execution model

- local twin for each device/operator

An execution model for a collaborative environment

(the pictures are courtesy of Actyx AG)



People + Real-time controllers + IT systems and networks:

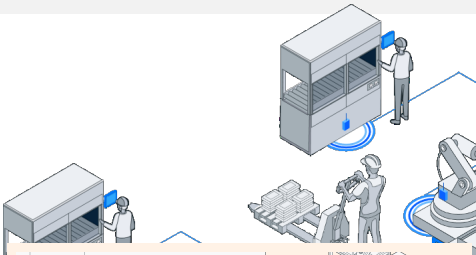
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

Execution model

- local twin for each device/operator
- twins are replicated where needed

An execution model for a collaborative environment

(the pictures are courtesy of Actyx AG)



People + Real-time controllers + IT systems and networks:

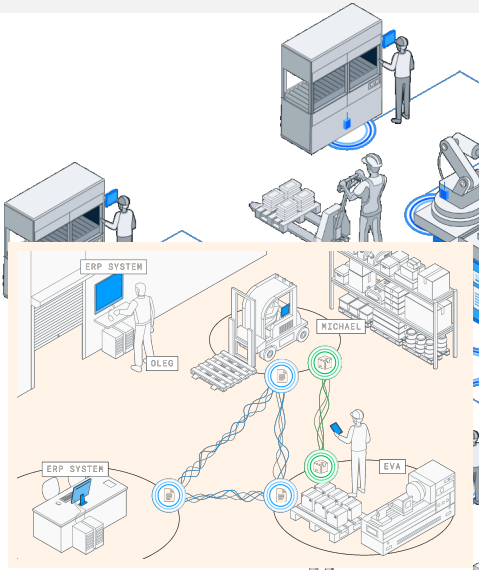
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

Execution model

- local twin for each device/operator
- twins are replicated where needed
- events have unique IDs and
  - record facts (e.g., from sensors) or
  - decisions (e.g., from an operator)
  - spread information asynchronously

An execution model for a collaborative environment

(the pictures are courtesy of Actyx AG)



People + Real-time controllers + IT systems and networks:

- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

Execution model

- local twin for each device/operator
- twins are replicated where needed
- events have unique IDs and
  - record facts (e.g., from sensors) or
  - decisions (e.g., from an operator)
  - spread information asynchronously
- logs are local to twins
  - a log $\equiv$ computational state of its twin
  - replicated logs are merged

Some motivations

We don't put it on paper...but honestly...it was mainly for fun

Some motivations

We don't put it on paper...but honestly...it was mainly for fun

- Are there any non-distributed applications/systems?

Some motivations

We don't put it on paper...but honestly...it was mainly for fun

- Are there any non-distributed applications/systems?
- Is it safe/desirable to let your fridge and mobile go in the cloud to interact?
  - Where would you like to keep your data?
  - What about your privacy?

Some motivations

We don't put it on paper...but honestly...it was mainly for fun

- Are there any non-distributed applications/systems?
- Is it safe/desirable to let your fridge and mobile go in the cloud to interact?
  - Where would you like to keep your data?
  - What about your privacy?
- "Anytime, anywhere..."

Some motivations

We don't put it on paper...but honestly...it was mainly for fun

- Are there any non-distributed applications/systems?
- Is it safe/desirable to let your fridge and mobile go in the cloud to interact?
  - Where would you like to keep your data?
  - What about your privacy?

really?

- "Anytime, anywhere..."
  - like during the AWS's outage on 25/11/2020
  - or almost all Google services down on 14/12/2020
  - DSL typical availability of 97%
    - acutally, some SLA have no lower bound [What Is the Digital Divide?]

Stop having your head in the clouds

Stop having your head in the clouds

Local-first principles

- Thou shalt be available
- Thou shalt be autonomous
- Thou shalt be collaborative
- Thou shalt recognise and embrace conflicts
- Thou shalt resolve conflicts and get consistency eventually

Stop having your head in the clouds

Local-first principles

- Thou shalt be available
- Thou shalt be autonomous
- Thou shalt be collaborative
- Thou shalt recognise and embrace conflicts
- Thou shalt resolve conflicts and get consistency eventually

Challenges in specifying protocols where decisions

- don't require consensus
- are based on stale local states
- yet, "work"

Our proposal

We define behavioural specs that

- feature
  - pub-subscribe (instead point-to-point)
  - (generalised) choices
  - arbitrary (and variable) number of instances
- trade "continuous" consistency for availability
  - price: "old" properties (eg session fidelity) are lost
  - gain: eventual-consistency & flexiblity (eg roles can have multiple instances)

design (well)

+

project

+

run

Classic behavioural specs

design (well)
+
project
+
run

LF approach

execute
+
propagate
+
merge

– Swarms and Swarm Protocols –

Ingredients (I): events & logs

Events e

$src(e)$

Logs $e_1 \cdot e_2 \ldots$

Ingredients (I): events & logs

Events $\quad \vdash \quad e \quad : \quad t$

$$src(e)$$

Logs $\quad \vdash \quad e_1 \cdot e_2 \ldots \quad : \quad t_1 \cdot t_2 \ldots$

Ingredients (I): events & logs

Events $\vdash e : t$

$src(e)$

Logs $\vdash e_1 \cdot e_2 \ldots : t_1 \cdot t_2 \ldots$

order induced by $\ell = e_1 \cdots e_n$ $e_i <_\ell e_j \iff i < j$

Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)

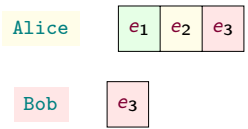
Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)
Events are appended to the logs of machines in **two phases**:

Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)
Events are appended to the logs of machines in **two phases**:

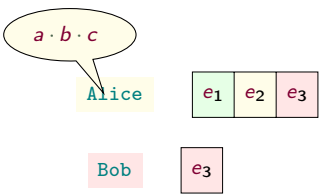
1st Phase: emitted events are appended to the local log of the emitting machine



Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)
Events are appended to the logs of machines in **two phases**:

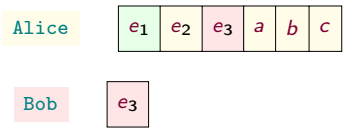
1st Phase: emitted events are appended to the local log of the emitting machine



Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)
Events are appended to the logs of machines in **two phases**:

1st Phase: emitted events are appended to the local log of the emitting machine

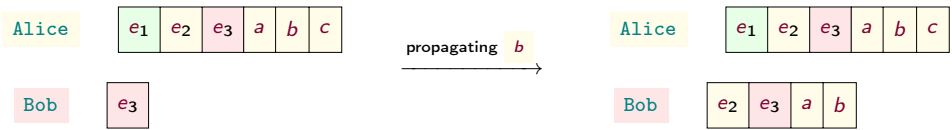


Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)
Events are appended to the logs of machines in **two phases**:

1st Phase: emitted events are appended to the local log of the emitting machine

2nd Phase: newly emitted events are shipped to other machines

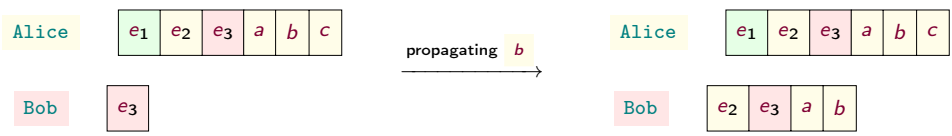


Ingredients (II): log shipping

Machines **emit** logs upon execution of commands (we'll see how in a moment)
Events are appended to the logs of machines in **two phases**:

1st Phase: emitted events are appended to the local log of the emitting machine

2nd Phase: newly emitted events are shipped to other machines



a is shipped too!

Machines by example

Let's build a machine `InitialP` for coordinating a taxi service.


```
InitialP  =
```

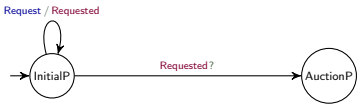
Machines by example

Let's build a machine `InitialP` for coordinating a taxi service.

$$\texttt{InitialP} \quad = \quad \text{Request} \mapsto \text{Requested} \cdot$$

Machines by example

Let's build a machine `InitialP` for coordinating a taxi service.

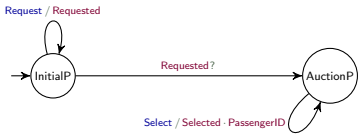


$$\texttt{InitialP} \quad = \quad \text{Request} \mapsto \text{Requested} \cdot [\text{Requested}?\,\underline{\texttt{AuctionP}}]$$

$$\texttt{AuctionP} \quad =$$

Machines by example

Let's build a machine `InitialP` for coordinating a taxi service.

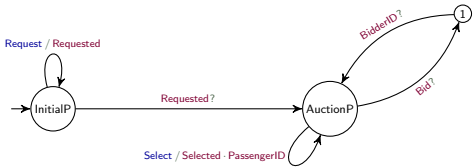


$$\texttt{InitialP} \quad = \quad \text{Request} \mapsto \text{Requested} \cdot \, [\text{Requested}?\, \underline{\texttt{AuctionP}}]$$

$$\texttt{AuctionP} \quad = \quad \text{Select} \mapsto \text{Selected} \cdot \text{PassengerId} \cdot$$

Machines by example

Let's build a machine `InitialP` for coordinating a taxi service.

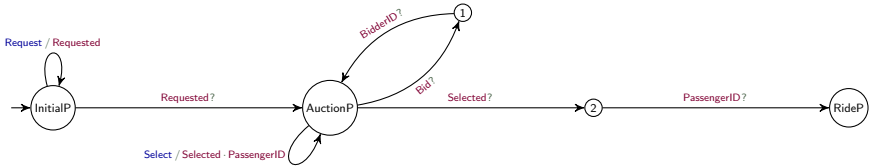


$$\texttt{InitialP} \quad = \quad \text{Request} \mapsto \text{Requested} \cdot \, [\text{Requested}? \, \underline{\texttt{AuctionP}}]$$

$$\texttt{AuctionP} \quad = \quad \text{Select} \mapsto \text{Selected} \cdot \text{PassengerId} \cdot \, [\\ \text{Bid}? \, \underline{\text{BidderId}? \, \texttt{AuctionP}}$$

Machines by example

Let's build a machine `InitialP` for coordinating a taxi service.



$$
\begin{aligned}
\texttt{InitialP} &= \mathsf{Request} \mapsto \mathsf{Requested} \cdot [\mathsf{Requested}?\,\underline{\texttt{AuctionP}}] \\
\texttt{AuctionP} &= \mathsf{Select} \mapsto \mathsf{Selected} \cdot \mathsf{PassengerId} \cdot [\\
&\qquad \mathsf{Bid}?\,\mathsf{BidderId}?\,\underline{\texttt{AuctionP}} \\
&\qquad \&\\
&\qquad \mathsf{Selected}?\,\mathsf{PassengerId}?\,\underline{\texttt{RideP}} \\
&\quad] \\
\texttt{RideP} &= \cdots
\end{aligned}
$$

Machines' semantics

So, think of $\mathtt{M} = \kappa \cdot [\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$ as an FSA emitting/consuming events according to its transitions:

- either self-loops (determined by the κ part)
- or event consumption (determined by the guards of the branches $\mathtt{t}_i$)

Machines' semantics

So, think of $\mathtt{M} = \kappa \cdot [\mathtt{t}_1?\,\mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n?\,\mathtt{M}_n]$ as an FSA emitting/consuming events according to its transitions:

- either self-loops (determined by the κ part)
- or event consumption (determined by the guards of the branches $\mathtt{t}_i$)

State transition function :

$$\delta(\mathtt{M}, \epsilon) = \mathtt{M}$$

$$\delta(\mathtt{M}, e \cdot \ell) = \begin{cases} \delta(\mathtt{M}', \ell) & \text{if } \vdash e : \mathtt{t},\ \mathtt{M} \xrightarrow{\mathtt{t}?} \mathtt{M}' \\ \delta(\mathtt{M}, \ell) & \text{otherwise} \end{cases}$$

Machines' semantics

So, think of $\mathtt{M} = \kappa \cdot [\mathtt{t}_1?\,\mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n?\,\mathtt{M}_n]$ as an FSA emitting/consuming events according to its transitions:

- either self-loops (determined by the κ part)
- or event consumption (determined by the guards of the branches $\mathtt{t}_i$)

State transition function :

$$\delta(\mathtt{M}, \epsilon) = \mathtt{M}$$

$$\delta(\mathtt{M}, e \cdot \ell) = \begin{cases} \delta(\mathtt{M}', \ell) & \text{if } \vdash e : \mathtt{t},\ \mathtt{M} \xrightarrow{\mathtt{t}?} \mathtt{M}' \\ \delta(\mathtt{M}, \ell) & \text{otherwise} \end{cases}$$

That is

$\mathtt{M}$ with local log ℓ is in the implicit state $\delta(\mathtt{M}, \ell)$ reached after processing each event in ℓ

Machines' semantics

So, think of $\mathtt{M} = \kappa \cdot [\mathtt{t}_1?\,\mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n?\,\mathtt{M}_n]$ as an FSA emitting/consuming events according to its transitions:

- either self-loops (determined by the κ part)
- or event consumption (determined by the guards of the branches $\mathtt{t}_i$)

State transition function :

$$\delta(\mathtt{M}, \epsilon) = \mathtt{M}$$

$$\delta(\mathtt{M}, e \cdot \ell) = \begin{cases} \delta(\mathtt{M}', \ell) & \text{if } \vdash e : \mathtt{t},\ \mathtt{M} \xrightarrow{\mathtt{t}?} \mathtt{M}' \\ \delta(\mathtt{M}, \ell) & \text{otherwise} \end{cases}$$

That is
M with local log ℓ is in the implicit state $\delta(\mathtt{M}, \ell)$ reached after processing each event in ℓ

$$\frac{\delta(\mathtt{M}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} \delta(\mathtt{M}, \ell) \qquad \ell' \text{ fresh} \qquad \vdash \ell' : \mathtt{l}}{(\mathtt{M}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} (\mathtt{M}, \ell \cdot \ell')}$$

Machines' semantics

So, think of $\mathtt{M} = \kappa \cdot [\mathtt{t}_1?\,\mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n?\,\mathtt{M}_n]$ as an FSA emitting/consuming events according to its transitions:

- either self-loops (determined by the κ part)
- or event consumption (determined by the guards of the branches $\mathtt{t}_i$)

State transition function :

$$\delta(\mathtt{M}, \epsilon) = \mathtt{M}$$

$$\delta(\mathtt{M}, e \cdot \ell) = \begin{cases} \delta(\mathtt{M}', \ell) & \text{if } \vdash e : \mathtt{t},\ \mathtt{M} \xrightarrow{\mathtt{t}?} \mathtt{M}' \\ \delta(\mathtt{M}, \ell) & \text{otherwise} \end{cases}$$

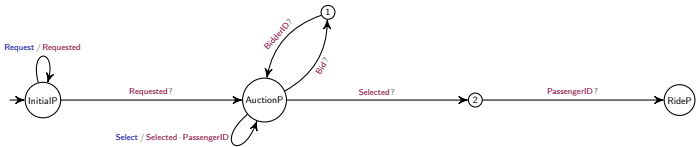
$$\frac{\delta(\mathtt{M}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \delta(\mathtt{M}, \ell) \qquad \ell' \text{ fresh} \qquad \vdash \ell' : \mathtt{1}}{(\mathtt{M}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} (\mathtt{M}, \ell \cdot \ell')}$$

That is
$\mathtt{M}$ with local log ℓ is in the implicit state $\delta(\mathtt{M}, \ell)$ reached after processing each event in ℓ

That is
after processing the events in ℓ, $\mathtt{M}$ reaches a state enabling $\mathtt{c}\,/\,\mathtt{1}$ then the command execution can emit ℓ' of type $\mathtt{1}$ and append it to the local log of $\mathtt{M}$

An example

Take the machine `InitialP` (slide 12) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\not\vdash \mathit{ignoreMe} : \mathsf{Requested}$ and $\not\vdash \mathit{ignoreMeToo} : \mathsf{Requested}$

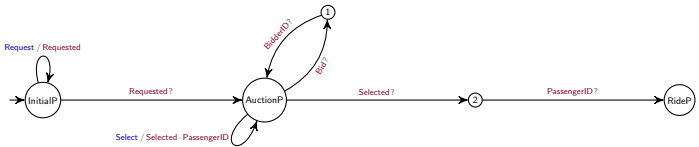


By definition of δ

- $\delta(\texttt{InitialP}, \ell) = \texttt{InitialP}$

An example

Take the machine `InitialP` (slide 12) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\nvdash \mathit{ignoreMe} : \mathrm{Requested}$ and $\nvdash \mathit{ignoreMeToo} : \mathrm{Requested}$

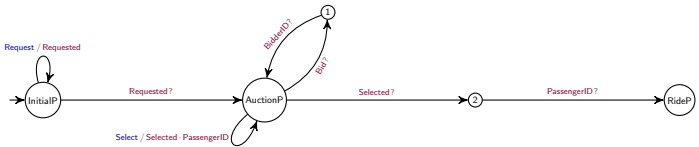


By definition of δ

- $\delta(\texttt{InitialP}, \ell) = \texttt{InitialP}$ hence
- $\delta(\texttt{InitialP}, \ell) \xrightarrow{\text{Request / Requested}} \delta(\texttt{InitialP}, \ell)$

An example

Take the machine `InitialP` (slide 12) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\nvdash \mathit{ignoreMe} : \mathrm{Requested}$ and $\nvdash \mathit{ignoreMeToo} : \mathrm{Requested}$



By definition of δ

- $\delta(\texttt{InitialP}, \ell) = \texttt{InitialP}$ hence
- $\delta(\texttt{InitialP}, \ell) \xrightarrow{\text{Request / Requested}} \delta(\texttt{InitialP}, \ell)$ hence
- $(\texttt{InitialP}, \ell) \xrightarrow{\text{Request / Requested}} (\texttt{InitialP}, \ell \cdot \mathit{Requested})$
  with $\vdash \mathit{Requested} : \mathrm{Request}$ and $\mathit{src}(\mathit{Requested}) = \mathtt{P}$

Some considerations

Commands are enabled only from the state reached after processing all the events in the local log of the machine featuring them

Some considerations

Commands are enabled only from the state reached after processing all the events in the local log of the machine featuring them

Logs grow monotonically

Some considerations

Commands are enabled only from the state reached after processing all the events in the local log of the machine featuring them

Logs grow monotonically

The environment triggers commands: swarms are inherently non-deterministic!

Some considerations

Commands are enabled only from the state reached after processing all the events in the local log of the machine featuring them

Logs grow monotonically

The environment triggers commands: swarms are inherently non-deterministic!

We have formalised the emission of events and their consumption

Some considerations

Commands are enabled only from the state reached after processing all the events in the local log of the machine featuring them

Logs grow monotonically

The environment triggers commands: swarms are inherently non-deterministic!

We have formalised the emission of events and their consumption

We now focus on the formalisation of log shipping

Coherence

A swarm $\mathtt{M}_1\boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n\boxed{\ell_n} \mid \ell$ is coherent if $\ell = \bigcup_{1 \leq i \leq n} \ell_i$ and $\ell_i \sqsubseteq \ell$ for $1 \leq i \leq n$

Coherence

A swarm $\mathtt{M}_1\boxed{\ell_1}\,|\,\ldots\,|\,\mathtt{M}_n\boxed{\ell_n}\,|\,\ell$ is coherent if $\ell = \bigcup_{1\le i\le n} \ell_i$ and $\ell_i \sqsubseteq \ell$ for $1 \le i \le n$

where $\ell_i \sqsubseteq \ell$ is the sublog relation defined as

- $\ell_i \subseteq \ell$ and $<_{\ell_i} \subseteq <_\ell$ and

That is
all events of ℓ_i appear in the same order in ℓ

- $e <_\ell e'$, $src(e) = src(e')$ and $e' \in \ell_1 \implies e \in \ell_i$

That is
the per-source partitions of ℓ_i are prefixes of the corresponding partitions of ℓ

Coherence

A swarm $\mathtt{M}_1\boxed{\ell_1}\,|\,\ldots\,|\,\mathtt{M}_n\boxed{\ell_n}\,|\,\ell$ is coherent if $\ell = \bigcup_{1\le i\le n} \ell_i$ and $\ell_i \sqsubseteq \ell$ for $1 \le i \le n$

where $\ell_i \sqsubseteq \ell$ is the sublog relation defined as

- $\ell_i \subseteq \ell$ and $<_{\ell_i} \subseteq <_\ell$ and

That is
all events of ℓ_i appear in the same order in ℓ

- $e <_\ell e'$, $src(e) = src(e')$ and $e' \in \ell_1 \implies e \in \ell_i$

That is
the per-source partitions of ℓ_i are prefixes of the corresponding partitions of ℓ

Hereafter, we assume coherence

Semantics by example

If $\mathtt{B}\boxed{b} \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{B}\boxed{b \cdot d \cdot e}$ with $\vdash d \cdot e : \mathtt{1}$

Semantics by example

If $\mathtt{B}\boxed{b} \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{B}\boxed{b \cdot d \cdot e}$ with $\vdash d \cdot e : \mathtt{1}$

then, by [Local] $\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \ell$

Semantics by example

If $\mathtt{B}\boxed{b} \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{B}\boxed{b \cdot d \cdot e}$ with $\vdash d \cdot e : \mathtt{1}$

then, by [Local] $\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \ell$ *Many!*

for all $\ell \in (b \cdot a \cdot c) \bowtie (b \cdot d \cdot e)$

$$\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \overbrace{b \cdot a \cdot d \cdot e \cdot c}^{=\ell}$$

Semantics by example

If $\mathtt{B}\boxed{b} \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{B}\boxed{b \cdot d \cdot e}$ with $\vdash d \cdot e : \mathtt{1}$

then, by [Local] $\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \ell$ *Many!*

for all $\ell \in (b \cdot a \cdot c) \bowtie (b \cdot d \cdot e)$

$$\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \overbrace{b \cdot a \cdot d \cdot e \cdot c}^{=\ell}$$

By rule [Prop] we can propagate a non-deterministically a chosen sublog of $b \cdot d \cdot e$

Semantics by example

If $\mathtt{B}\boxed{b} \xrightarrow{\mathtt{c}\,/\,1} \mathtt{B}\boxed{b \cdot d \cdot e}$ with $\vdash d \cdot e : 1$

then, by [Local] $\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,1} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \ell$ *Many!*

for all $\ell \in (b \cdot a \cdot c) \bowtie (b \cdot d \cdot e)$

$$\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b} \mid \mathtt{C}\boxed{c} \mid b \cdot a \cdot c \xrightarrow{\mathtt{c}\,/\,1} \mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \overbrace{b \cdot a \cdot d \cdot e \cdot c}^{=\ell}$$

By rule [Prop] we can propagate a non-deterministically a chosen sublog of $b \cdot d \cdot e$

Let's propagate $d \cdot e$ $\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \ell \xrightarrow{\tau} \mathtt{A}\boxed{b \cdot a \cdot d \cdot e} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{c} \mid \ell$

$\mathtt{A}\boxed{a} \mid \mathtt{B}\boxed{b \cdot d \cdot e} \mid \mathtt{C}\boxed{b \cdot d \cdot e \cdot c} \mid \ell \xleftarrow{\tau}$

Semantics of swarms

By rule [Local] below, a command's execution updates both local and global logs

$$\frac{\mathtt{S}(i) = \mathtt{M}\boxed{\ell_i} \qquad \mathtt{M}\boxed{\ell_i} \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} \mathtt{M}\boxed{\ell_i'} \qquad \mathit{src}(\ell_i' \setminus \ell_i) = \{i\} \qquad \ell' \in \ell \bowtie \ell_i'}{(\mathtt{S}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} (\mathtt{S}[i \mapsto \mathtt{M}\boxed{\ell_i'}], \ell')}\,[\text{Local}]$$

Semantics of swarms

By rule [Local] below, a command's execution updates both local and global logs

$$\frac{\mathtt{S}(i) = \mathtt{M}\boxed{\ell_i} \qquad \mathtt{M}\boxed{\ell_i} \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} \mathtt{M}\boxed{\ell_i'} \qquad \mathit{src}(\ell_i' \setminus \ell_i) = \{i\} \qquad \ell' \in \ell \bowtie \ell_i'}{(\mathtt{S}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} (\mathtt{S}[i \mapsto \mathtt{M}\boxed{\ell_i'}], \ell')}\,[\text{Local}]$$

$$\frac{\mathtt{S}(i) = \mathtt{M}\boxed{\ell_i} \qquad \ell_i \sqsubseteq \ell' \sqsubseteq \ell \qquad \ell_i \subset \ell'}{(\mathtt{S}, \ell) \xrightarrow{\tau} (\mathtt{S}[i \mapsto \mathtt{M}\boxed{\ell'}], \ell)}\,[\text{Prop}]$$

By rule [Prop] above, the propagation of events happens

- by shipping a **non-deterministically chosen** subset of events in the global log
- to a **non-deterministically chosen** machine

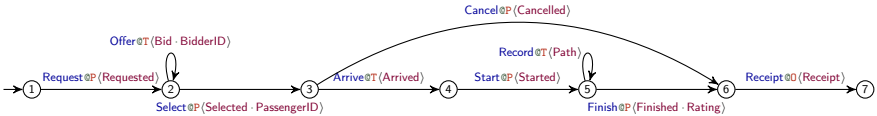
– Behavioural Types for Swarms –

Swarm protocols as FSA

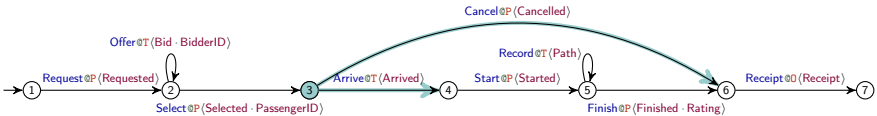
Like for machines, a swarm protocols $\mathsf{G} = \sum_{i \in I} \mathtt{c}_i@\mathtt{R}_i\langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i$ has an associated FSA:

- the set of states consists of G plus the states in G_i for each $i \in \underline{n}$
- G is the initial state
- for each $i \in I$, G has a transition to state G_i labelled with $\mathtt{c}_i@\mathtt{R}_i\langle \mathtt{l}_i \rangle$, written $\mathsf{G} \xrightarrow{\mathtt{c}_i \,/\, \mathtt{l}_i} \mathsf{G}_i$

An example


Cancel@P⟨Cancelled⟩
Offer@T⟨Bid · BidderID⟩
Record@T⟨Path⟩
Request@P⟨Requested⟩
Arrive@T⟨Arrived⟩
Start@P⟨Started⟩
Receipt@O⟨Receipt⟩
Select@P⟨Selected · PassengerID⟩
Finish@P⟨Finished · Rating⟩
1
2
3
4
5
6
7

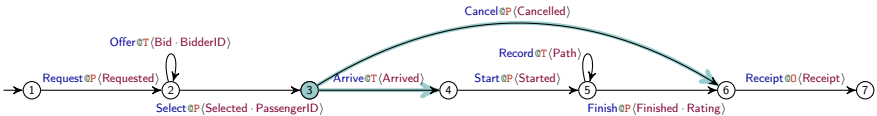
An example



There is a race in state 3!

- the driver of the selected taxi may invoke Arrive
- **while** P loses patience and invokes Cancel

An example



There is a race in state 3!

- the driver of the selected taxi may invoke Arrive
- **while** P loses patience and invokes Cancel

This protocol violates well-formedness conditions typically imposed on behavioural types due to the race in state 3 (because it has two selectors, which is also true of states 2 and 5)

Semantics of swarm protocols

One rule only!

$$\frac{}{(\mathsf{G}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} (\mathsf{G}, \ell \quad)}\ \text{[G-Cmd]}$$

Semantics of swarm protocols

One rule only!

$$\frac{\delta(\mathsf{G}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathsf{G}'}{(\mathsf{G}, \ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} (\mathsf{G}, \ell \quad)} \text{[G-Cmd]}$$

where

$$\delta(\mathsf{G}, \ell) = \begin{cases} \mathsf{G} & \text{if } \ell = \epsilon \\ \delta(\mathsf{G}', \ell'') & \text{if } \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathsf{G}' \text{ and } \vdash \ell' : \mathtt{1} \text{ and } \ell = \ell' \cdot \ell'' \\ \bot & \text{otherwise} \end{cases}$$

Logs to be consumed "atomically", hence $\delta(\mathsf{G}, \ell)$ may be undefined

Semantics of swarm protocols

One rule only!

$$\frac{\delta(\mathsf{G},\ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathsf{G}' \qquad \vdash \ell' : \mathtt{1} \qquad \ell' \text{ log of fresh events}}{(\mathsf{G},\ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} (\mathsf{G},\ell\cdot\ell')}\text{[G-Cmd]}$$

where

$$\delta(\mathsf{G},\ell) = \begin{cases} \mathsf{G} & \text{if } \ell = \epsilon \\ \delta(\mathsf{G}',\ell'') & \text{if } \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathsf{G}' \text{ and } \vdash \ell' : \mathtt{1} \text{ and } \ell = \ell'\cdot\ell'' \\ \bot & \text{otherwise} \end{cases}$$

Logs to be consumed "atomically", hence $\delta(\mathsf{G},\ell)$ may be undefined

Semantics of swarm protocols

One rule only!

$$\frac{\delta(\mathsf{G},\ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} \mathsf{G}' \qquad \vdash \ell' : \mathtt{l} \qquad \ell' \text{ log of fresh events}}{(\mathsf{G},\ell) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} (\mathsf{G},\ell \cdot \ell')}\text{[G-Cmd]}$$

where

Logs to be consumed "atomically", hence $\delta(\mathsf{G},\ell)$ may be undefined

$$\delta(\mathsf{G},\ell) = \begin{cases} \mathsf{G} & \text{if } \ell = \epsilon \\ \delta(\mathsf{G}',\ell'') & \text{if } \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} \mathsf{G}' \text{ and } \vdash \ell' : \mathtt{l} \text{ and } \ell = \ell' \cdot \ell'' \\ \bot & \text{otherwise} \end{cases}$$

We restrict ourselves to deterministic swarm protocols that is, on different transitions from a same state, we require that

- log types start differently — log determinism
- pairs (command,role) differ — command determinism

From swarm protocols to machines

Transitions of a swarm protocol G are labelled with a role that may invoke the command

From swarm protocols to machines

Transitions of a swarm protocol G are labelled with a role that may invoke the command

Each machine plays one role

From swarm protocols to machines

Transitions of a swarm protocol G are labelled with a role that may invoke the command

Each machine plays one role

Obtain machines by projecting G on each role

From swarm protocols to machines

Transitions of a swarm protocol G are labelled with a role that may invoke the command

Each machine plays one role

Obtain machines by projecting G on each role

First attempt

$$\Big(\sum_{i\in I} \mathtt{c}_i@\mathtt{R}_i\langle \mathtt{l}_i\rangle \,.\, \mathsf{G}_i\Big)\downarrow_{\mathtt{R}} = \kappa \cdot [\&_{i\in I}\, \mathtt{l}_i?\, \mathsf{G}_i\downarrow_{\mathtt{R}}]$$

where $\kappa = \{(\mathtt{c}_i \,/\, \mathtt{l}_i) \;\big|\; \mathtt{R}_i = \mathtt{R} \text{ and } i \in I\}$ y

From swarm protocols to machines

Transitions of a swarm protocol G are labelled with a role that may invoke the command

Each machine plays one role

Obtain machines by projecting G on each role

First attempt

$$\Big(\sum_{i\in I} \mathtt{c}_i@\mathtt{R}_i\langle \mathtt{l}_i\rangle \,.\, \mathsf{G}_i\Big)\downarrow_{\mathtt{R}} = \kappa \cdot [\&_{i\in I}\, \mathtt{l}_i?\, \mathsf{G}_i\downarrow_{\mathtt{R}}]$$

where $\kappa = \{(\mathtt{c}_i \,/\, \mathtt{l}_i) \;\big|\; \mathtt{R}_i = \mathtt{R} \text{ and } i \in I\}$ y

simple, but

- projected machines are large in all but the most trivial cases
- processing **all** events is undesirable: security and efficiency

Another attempt

Let's subscribe to event types: maps from roles to sets of event types

In pub-sub, processes subscribe to "topics"

Another attempt

Let's subscribe to event types: maps from roles to sets of event types

In pub-sub, processes subscribe to "topics"

Given $\mathsf{G} = \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i$, the
projection of G on a role $\mathtt{R}$ with respect to subscription σ is

$$\mathsf{G} \downarrow_{\mathtt{R}}^{\sigma} = \kappa \cdot [\&_{j \in J} \, \mathsf{filter}(\mathtt{l}_j, \sigma(\mathtt{R}))? \, \mathsf{G}_j \downarrow_{\mathtt{R}}^{\sigma}] \qquad \text{where}$$

Another attempt

Let's subscribe to event types: maps from roles to sets of event types

In pub-sub, processes subscribe to "topics"

Given $\mathsf{G} = \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i$, the
projection of G on a role R with respect to subscription σ is

$$\mathsf{G} \downarrow^{\sigma}_{\mathtt{R}} = \kappa \cdot [\&_{j \in J} \, \mathsf{filter}(\mathtt{l}_j, \sigma(\mathtt{R}))\,?\,\mathsf{G}_j \downarrow^{\sigma}_{\mathtt{R}}] \qquad \text{where}$$

$$\kappa = \bigcup_{i \in I} \{\mathtt{c}_i \,/\, \mathtt{l}_i \mid \mathtt{R}_i = \mathtt{R}\}$$

$$J = \bigcup_{i \in I} \{\mathsf{filter}(\mathtt{l}_i, \sigma(\mathtt{R})) \neq \epsilon\}$$

$$\mathsf{filter}(\mathtt{l}, E) = \begin{cases} \epsilon, & \text{if } \mathtt{t} = \epsilon \\ \mathtt{t} \cdot \mathsf{filter}(\mathtt{l}', E) & \text{if } \mathtt{t} \in E \text{ and } \mathtt{l} = \mathtt{t} \cdot \mathtt{l}' \\ \mathsf{filter}(\mathtt{l}, E) & \text{otherwise} \end{cases}$$

Well-formedness: sufficient conditions for well-behaviour

Transitory deviations are tolerated provided that consistency is eventually recovered

Well-formedness: sufficient conditions for well-behaviour

Transitory deviations are tolerated provided that consistency is eventually recovered

Example

`T` may bid after `P` has made their decision if the selection event `T` has not yet been received.

This inconsistency is temporary: when the selection event reaches `T` this inconsistency is recognised and resolved

Well-formedness

Trading consistency for availability has implications:

Well-formedness = Causality

Trading consistency for availability has implications:

Propagation of events is non-atomic (cf. rule [Prop])

$\Longrightarrow$ differences in how machines perceive the (state of the) computation

Causality

Fix a subscription σ. For each branch $i \in I$ of $\mathtt{G} = \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathtt{G}_i$

If $\mathtt{R}$ should have a command enabled after $\mathtt{c}_i$ then $\sigma(\mathtt{R})$ contains some event type emitted by $\mathtt{c}_i$

Explicit re-enabling $\quad \sigma(\mathtt{R}_i) \cap \mathtt{l}_i \neq \emptyset$

Command causality if $\mathtt{R}$ executes a command in $\mathtt{G}_i$
then $\sigma(\mathtt{R}) \cap \mathtt{l}_i \neq \emptyset$ and $\sigma(\mathtt{R}) \cap \mathtt{l}_i \supseteq \bigcup_{\mathtt{R}' \in_\sigma \mathtt{G}_i} \sigma(\mathtt{R}') \cap \mathtt{l}_i$

Well-formedness = Causality + Determinacy

Trading consistency for availability has implications:

Propagation of events is non-atomic (cf. rule [Prop])

$\Longrightarrow$ different roles may take inconsistent decisions

Causality & Determinacy

Fix a subscription σ. For each branch $i \in I$ of $\mathtt{G} = \sum_{i \in I} \mathtt{c}_i@\mathtt{R}_i\langle \mathtt{l}_i \rangle \,.\, \mathtt{G}_i$

Explicit re-enabling	$\sigma(\mathtt{R}_i) \cap \mathtt{l}_i \neq \emptyset$
Command causality	if $\mathtt{R}$ executes a command in $\mathtt{G}_i$ then $\sigma(\mathtt{R}) \cap \mathtt{l}_i \neq \emptyset$ and $\sigma(\mathtt{R}) \cap \mathtt{l}_i \supseteq \bigcup_{\mathtt{R}' \in_\sigma \mathtt{G}_i} \sigma(\mathtt{R}') \cap \mathtt{l}_i$
Determinacy	$\mathtt{R} \in_\sigma \mathtt{G}_i \implies \mathtt{l}_i[0] \in \sigma(\mathtt{R})$

Well-formedness = Causality + Determinacy - Confusion

Trading consistency for availability has implications:

Propagation of events is non-atomic (cf. rule [Prop])

$\Longrightarrow$ branches unambiguously identified and events emitted on eventually discharged branches ignored

Causality & Determinacy & Confusion freeness

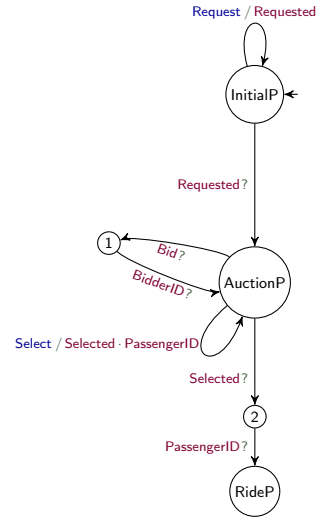
Fix a subscription σ. For each branch $i \in I$ of $\mathtt{G} = \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathtt{G}_i$

Explicit re-enabling	$\sigma(\mathtt{R}_i) \cap \mathtt{l}_i \neq \emptyset$
Command causality	if $\mathtt{R}$ executes a command in $\mathtt{G}_i$ then $\sigma(\mathtt{R}) \cap \mathtt{l}_i \neq \emptyset$ and $\sigma(\mathtt{R}) \cap \mathtt{l}_i \supseteq \bigcup_{\mathtt{R}' \in_\sigma \mathtt{G}_i} \sigma(\mathtt{R}') \cap \mathtt{l}_i$
Determinacy	$\mathtt{R} \in_\sigma \mathtt{G}_i \implies \mathtt{l}_i[0] \in \sigma(\mathtt{R})$
Confusion freeness	there is a unique subtree $\mathtt{G}'$ of $\mathtt{G}$ emitting $\mathtt{t}$ for each $\mathtt{t}$ starting a log emitted by a command in $\mathtt{G}$

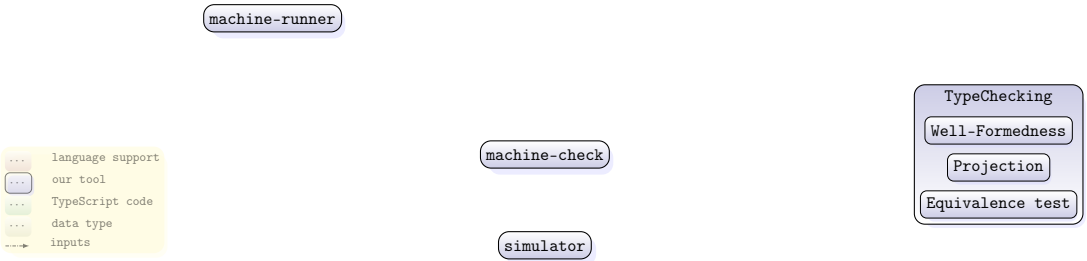
– Tooling –

```
// analogous for other events; "type" property matches type name (checked by tool)
type Requested = { type: 'Requested'; pickup: string; dest: string }
type Events = Requested | Bid | BidderID | Selected | ...

/** Initial state for role P */
@proto('taxiRide') // decorator injects inferred protocol into runtime
export class InitialP extends State<Events> {
  constructor(public id: string) { super() }
  execRequest(pickup: string, dest: string) {
    return this.events({ type: 'Requested', pickup, dest })
  }
  onRequested(ev: Requested) {
    return new AuctionP(this.id, ev.pickup, ev.dest, [])
  }
}
@proto('taxiRide')
export class AuctionP extends State<Events> {
  constructor(public id: string, public pickup: string, public dest: string,
    public bids: BidData[]) { super() }
  onBid(ev1: Bid, ev2: BidderID) {
    const [ price, time ] = ev1
    this.bids.push({ price, time, bidderID: ev2.id })
    return this
  }
  execSelect(taxiId: string) {
    return this.events({ type: 'Selected', taxiID },
                       { type: 'PassengerID', id: this.id })
  }
  onSelected(ev: Selected, id: PassengerID) {
    return new RideP(this.id, ev.taxiID)
  }
}
@proto('taxiRide')
export class RideP extends State<Events> { ... }
```

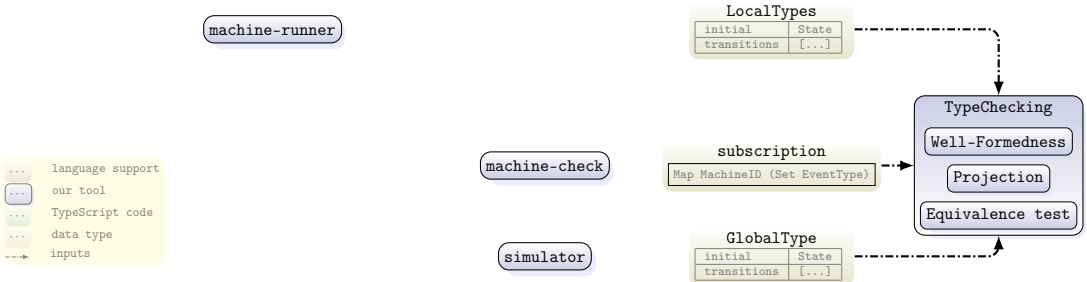


Architecture [2]



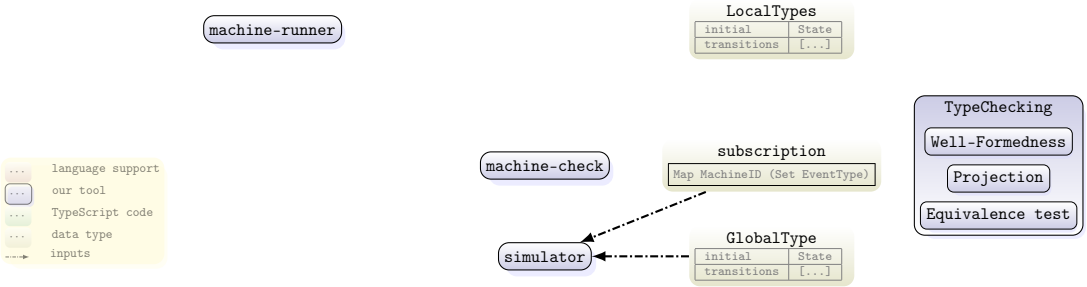
- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

Architecture [2]



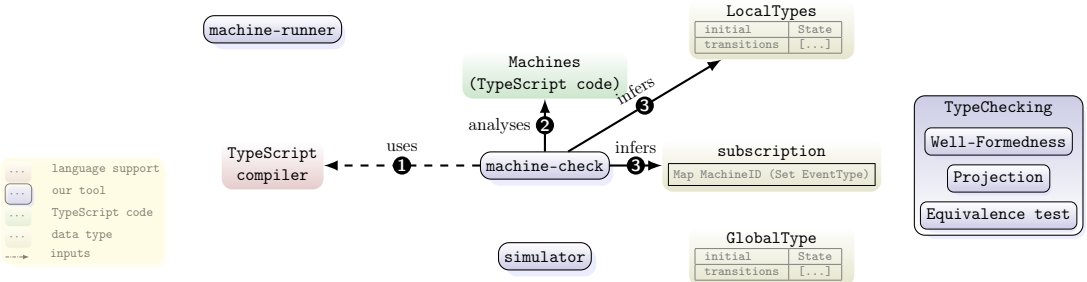
- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

Architecture [2]



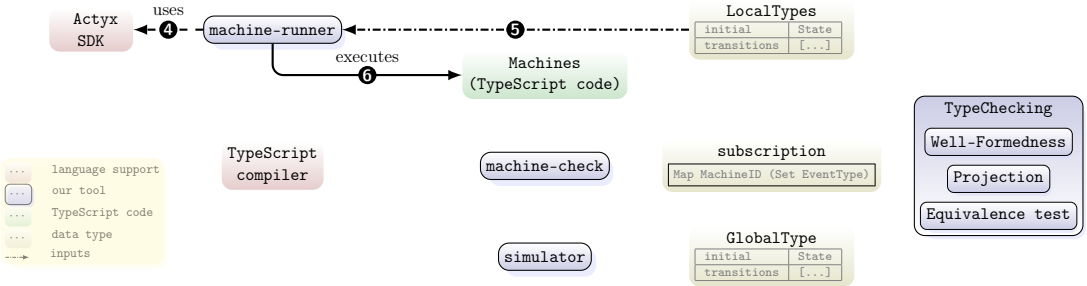
- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

Architecture [2]



- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

Architecture [2]



- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

– Wrapping up –

Summary & ongoing/future work

Summary & ongoing/future work

What did we glance at

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way
- Supported by a tool

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way
- Supported by a tool
- published at ECOOP 2023

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way
- Supported by a tool
- published at ECOOP 2023

What's next

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way
- Supported by a tool
- published at ECOOP 2023

What's next

- we've a generalisation of WF and compositionality (cf. ECOOP 2026)

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way
- Supported by a tool
- published at ECOOP 2023

What's next

- we've a generalisation of WF and compositionality (cf. ECOOP 2026)
- move from "closed" to "adversary" settings

Summary & ongoing/future work

What did we glance at

- A choreographic approach based on behavioural types
  - Supporting local-first principles
  - Inspired (and usable) in an industrial platform
  - Addressing non-standard properties
  - in a non-standard way
- Supported by a tool
- published at ECOOP 2023

What's next

- we've a generalisation of WF and compositionality (cf. ECOOP 2026)
- move from "closed" to "adversary" settings
- eventual consistency...fine...but how long should we wait for?

Thank you!

References I

[1] Roland Kuhn, Hernán C. Melgratti, and Emilio Tuosto. **Behavioural types for local-first software.** In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 15:1–15:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[2] Roland Kuhn, Hernán C. Melgratti, and Emilio Tuosto. **Behavioural types for local-first software (artifact).** *Dagstuhl Artifacts Ser.*, 9(2):14:1–14:5, 2023.