Behavioural Types for Local-First Software

Emilio Tuosto @ GSSI

joint work with

Roland Kuhn @ Actyx and Hernán Melgratti @ UBA

It-Matters
Lucca 11-12 July, 2023

– Prelude –

Take-away message

An approach to

trade consistency for availability in systems of **asymmetric replicated peers**

Take-away message

An approach to

trade consistency for availability in systems of **asymmetric replicated peers**

using local-first's principles to establish eventual consensus

Take-away message

An approach to

trade consistency for availability in systems of **asymmetric replicated peers**

using local-first's principles to establish eventual consensus

formally supported by behavioural types

Take-away message

An approach to

trade consistency for availability in systems of **asymmetric replicated peers**

using local-first's principles to establish eventual consensus

formally supported by behavioural types

- swarm = (machines + local logs) $*$ imaginary global log
- swarm protocols: systems from an abstract global viewpoint
- enforce good behaviour via behavioural typing

Take-away message

An approach to

trade consistency for availability in systems of **asymmetric replicated peers**

using local-first's principles to establish eventual consensus

formally supported by behavioural types

- swarm $=$ (machines $+$ local logs) $*$ imaginary global log
- swarm protocols: systems from an abstract global viewpoint
- enforce good behaviour via behavioural typing

See our recent ECOOP 2023 paper
(to appear; extended version available at `https://arxiv.org/abs/2305.04848`)

Distributed coordination

An “old” problem

Distributed agreement
Distributed sharing
Security
Computer-assisted collaborative work
...

With some “solutions”

Centralisation points
Consensus protocols
Commutative replicated data types
...

Distributed coordination

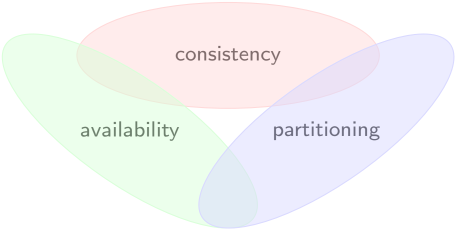
An "old" problem

Distributed agreement
Distributed sharing
Security
Computer-assisted collaborative work
...

Availability = Money

Kohavi et al. KDD'14

- Amazon sales down 1% if 100ms delay
- Google searches down 0.2% - 0.6% if 100-400ms delay
- Bing's revenue down ∼1.5% if 250ms delay



With some "solutions"

Centralisation points
Consensus protocols
Commutative replicated data types
...

A new (?) solution

What about using local-first principles?

Thou shall be autonomous

Thou shall collaborate

Thou shall recognise conflicts

Thou shall resolve conflicts

Thou shall be consistent

Plan of the talk

Some motivations

Our formalisation

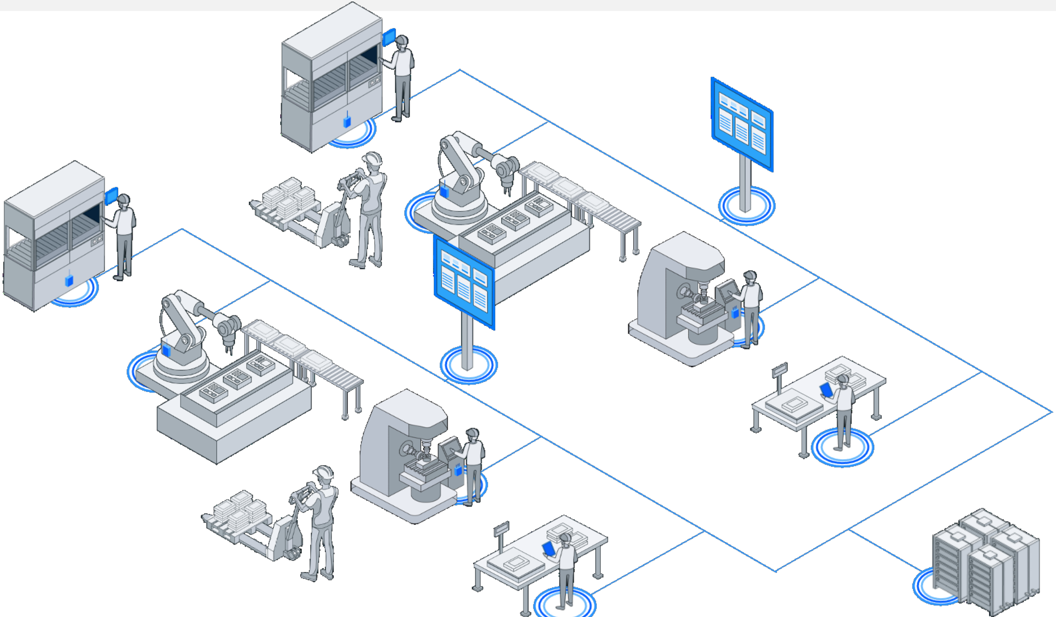
Our typing discipline

Tool support

Open issues

– Motivations –

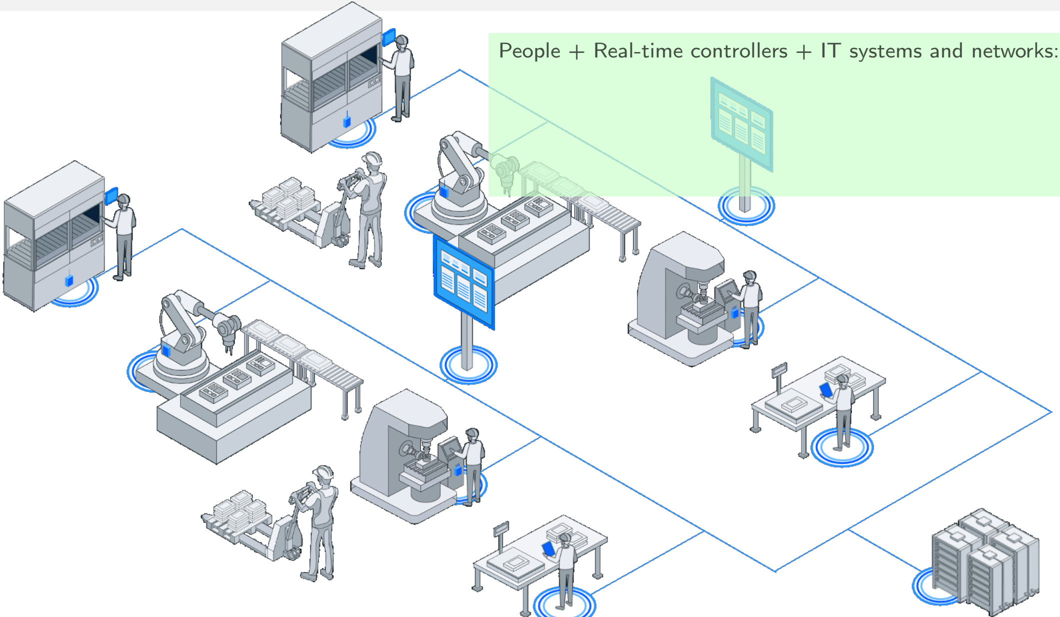
A collaborative environment and its execution model



(the pictures are courtesy of Actyx AG)

A collaborative environment and its execution model

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



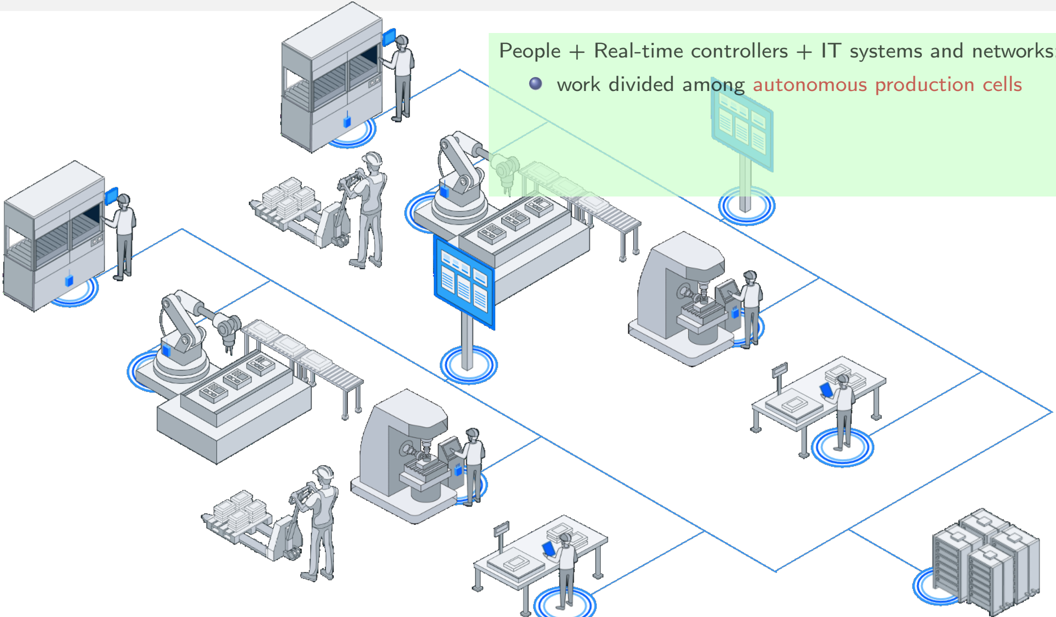
(the pictures are courtesy of Actyx AG)

A collaborative environment and its execution model

(the pictures are courtesy of Actyx AG)

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

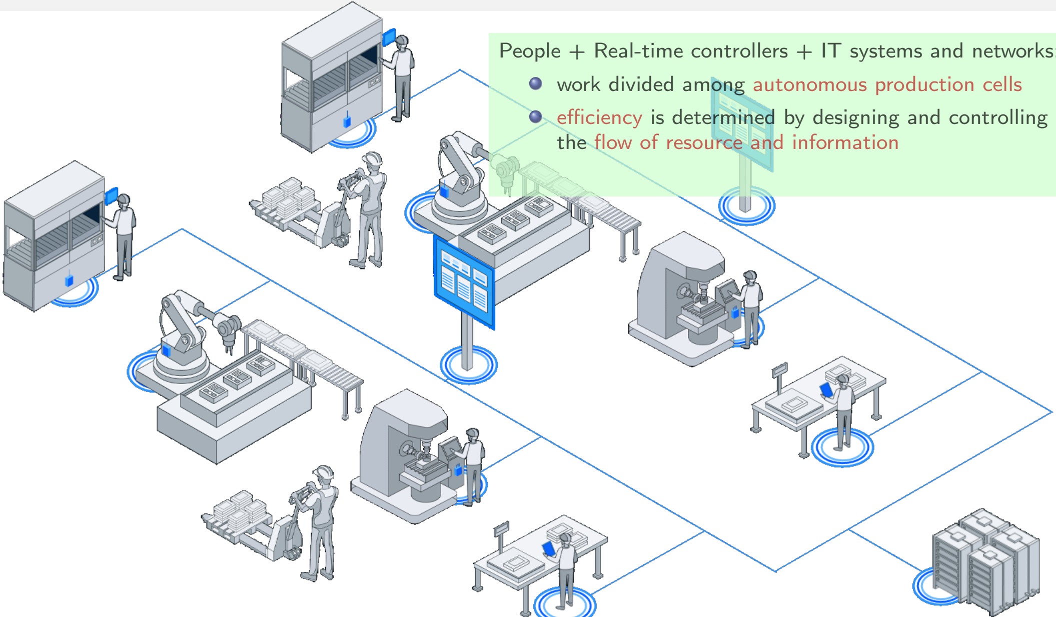
- work divided among autonomous production cells



A collaborative environment and its execution model

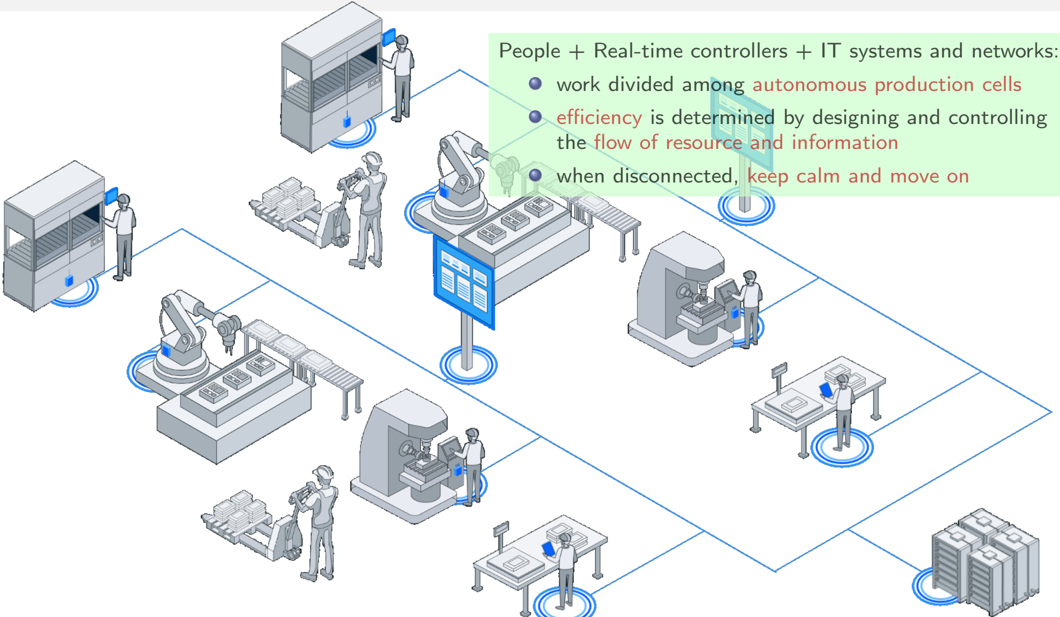
People + Real-time controllers + IT systems and networks:

- 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)

A collaborative environment and its execution model



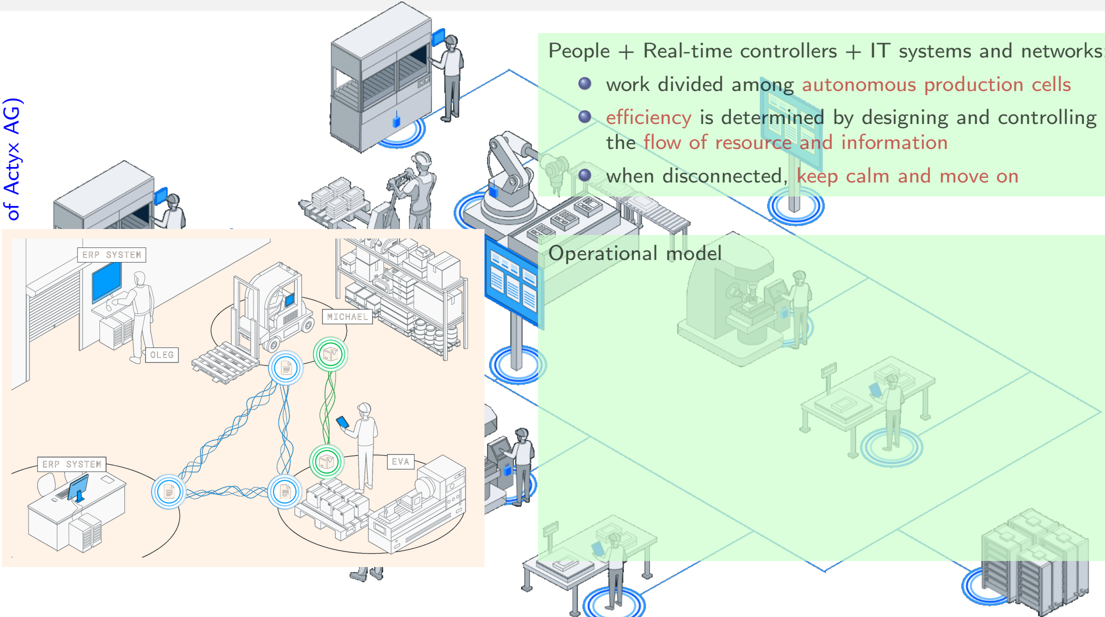
People + Real-time controllers + IT systems and networks:

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

(the pictures are courtesy of Actyx AG)

A collaborative environment and its execution model

of Actyx AG)



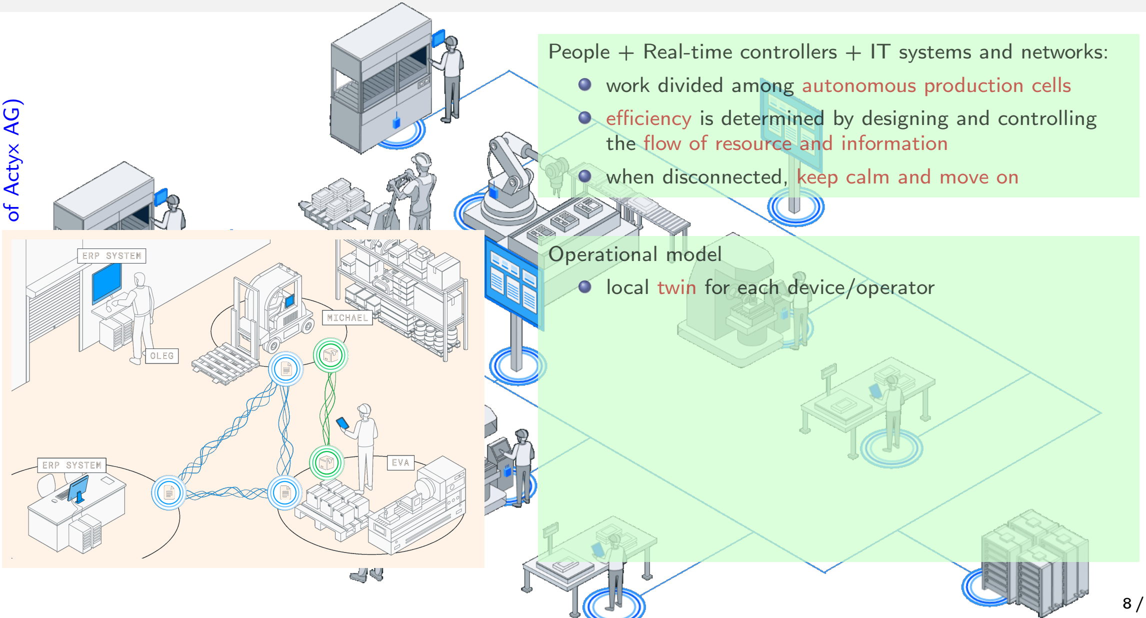
People + Real-time controllers + IT systems and networks:

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

Operational model

A collaborative environment and its execution model

of Actyx AG)



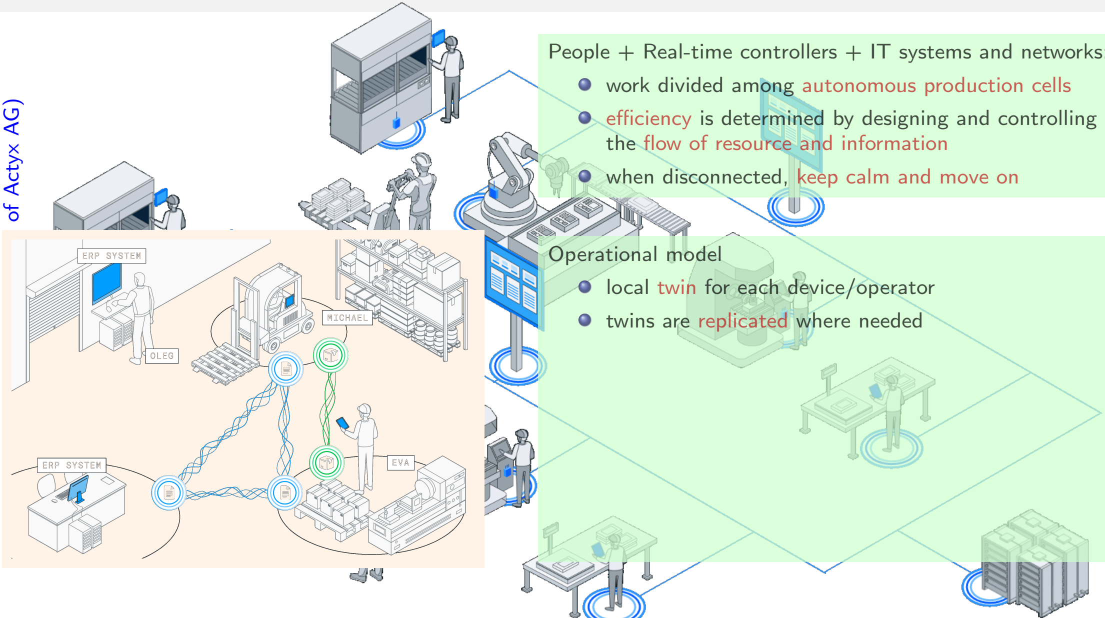
People + Real-time controllers + IT systems and networks:

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

Operational model

- local twin for each device/operator

of Actyx AG)



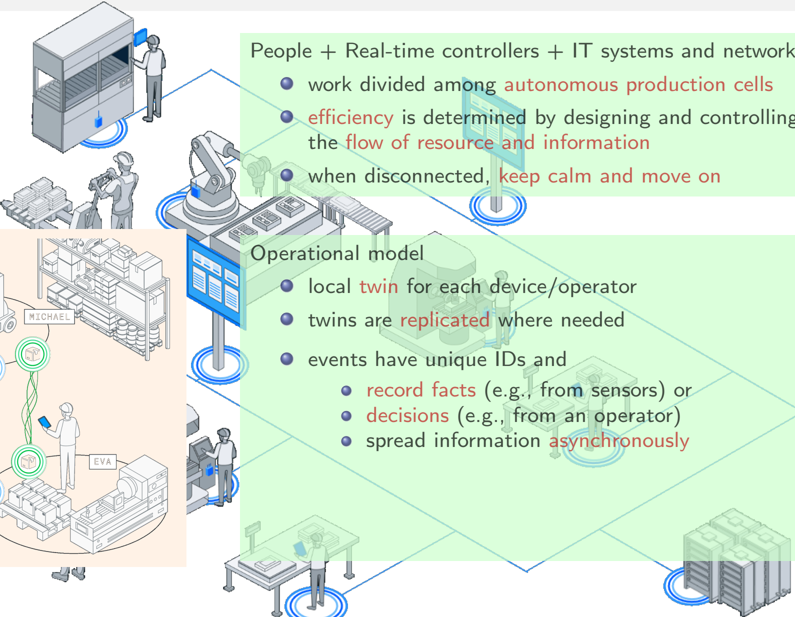
People + Real-time controllers + IT systems and networks:

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

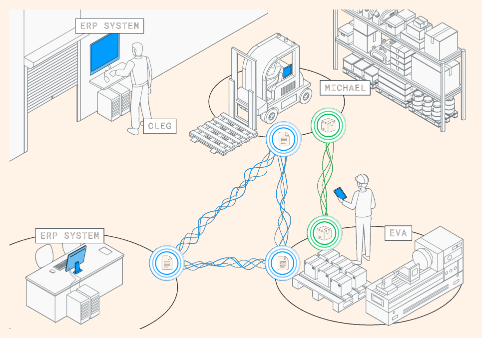
Operational model

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

A collaborative environment and its execution model



of Actyx AG)



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

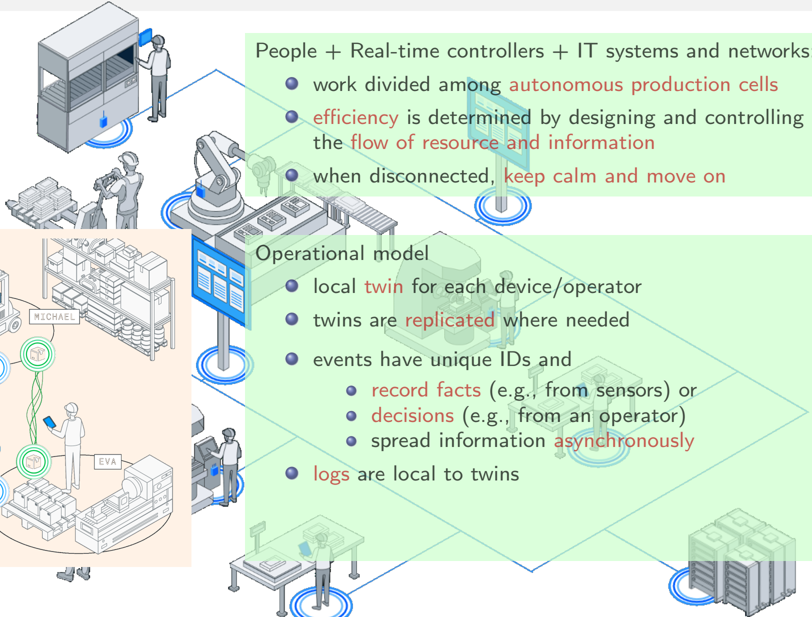
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- when disconnected, keep calm and move on

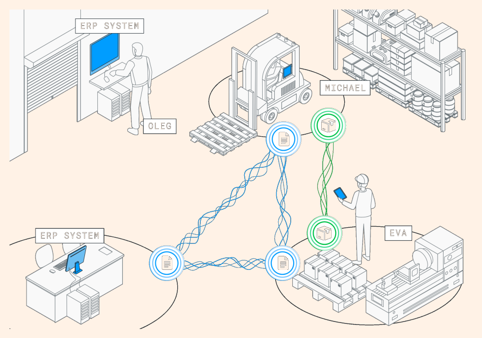
Operational model

- 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

A collaborative environment and its execution model

of Actyx AG)





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

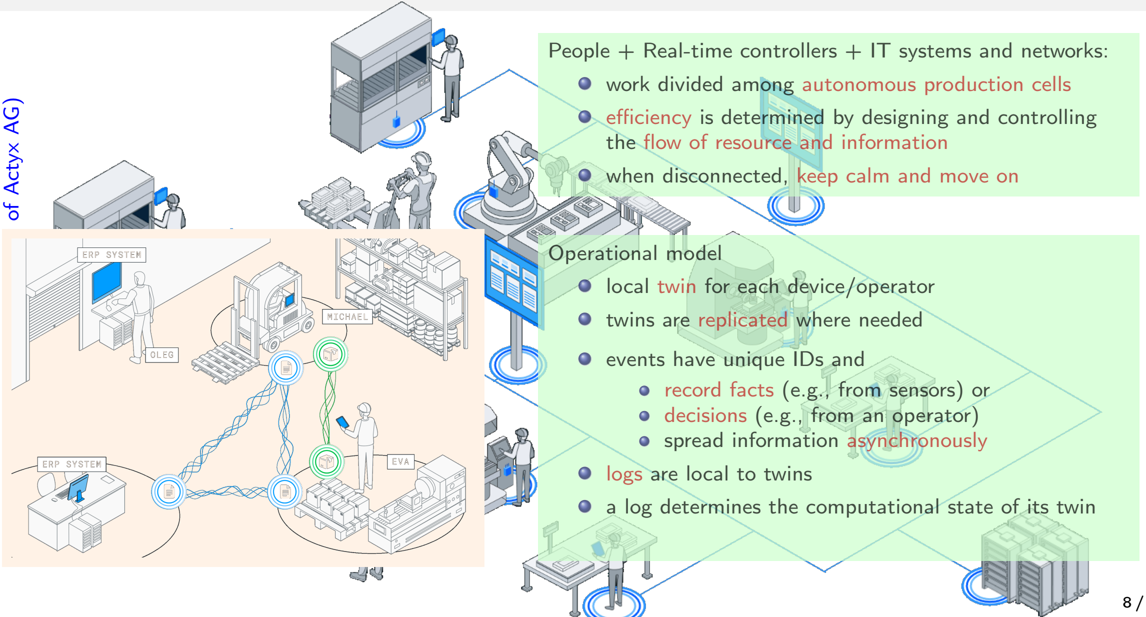
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- when disconnected, keep calm and move on

Operational 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 collaborative environment and its execution model

of Actyx AG)



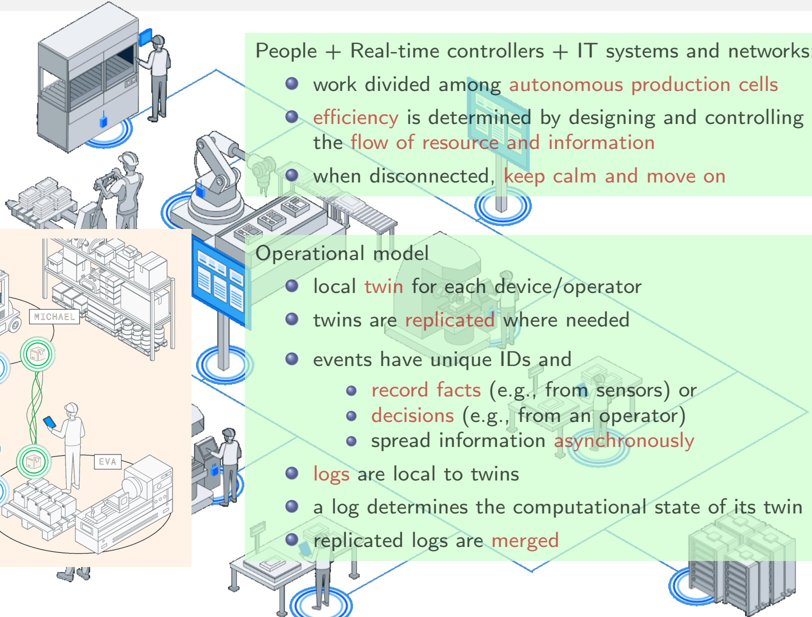
People + Real-time controllers + IT systems and networks:

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

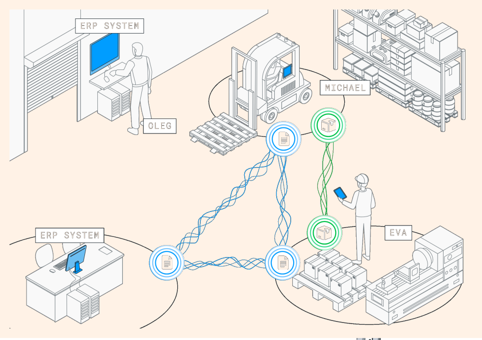
Operational model

- 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 determines the computational state of its twin

A collaborative environment and its execution model



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
- when disconnected, keep calm and move on

Operational 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 determines the computational state of its twin
- replicated logs are merged

The execution scheme

```
while true:
    execute ;
    propagate ;
    merge
```

Other application domains / motivations

More applications

Robots (e.g., rescue missions or space applications)

Collaborative applications (`https://automerge.org/`)

Home automation

Other application domains / motivations

IoT...really?

Why your fridge and mobile should go in the cloud to talk to each other?

Other application domains / motivations

"Anytime, anywhere..." really?

like the AWS's outage on 25/11/2020

or almost all Google services down on 14/12/2020

DSL typical availability of 97% (& some SLA have no lower bound) checkout
`https://www.internetsociety.org/blog/2022/03/what-is-the-digital-divide`

Other application domains / motivations

Also, taking decisions locally

can reduce downtime

shifts data ownership

gets rid of any centralization point...for real

Plan of the talk

A motivating case study

Our formalisation

Our typing discipline

Tool support

Future work

– A formal model –

Ingredients (I): events & logs

Events e

Logs $e_1 \cdot e_2 \dots$

Ingredients (I): events & logs

Events	$\vdash$	e	:	t
		$src(e)$		
Logs	$\vdash$	$e_1 \cdot e_2 \ldots$	:	$t_1 \cdot t_2 \ldots$

Ingredients (I): events & logs

Events $\vdash$ e : t

$$src(e)$$

Logs $\vdash$ $e_1 \cdot e_2 \ldots$: $\mathsf{t}_1 \cdot \mathsf{t}_2 \ldots$

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

Ingredients (II): log shipping

Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)

Ingredients (II): log shipping

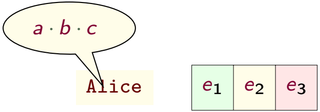
Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)

Such events are **appended** to the logs of machines in **two phases**:

Ingredients (II): log shipping

Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)
Such events are **appended** to the logs of machines in **two phases**:

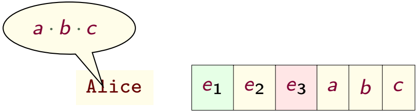
Phase I: emitted events are appended to the local log of the emitting machine



Ingredients (II): log shipping

Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)
Such events are **appended** to the logs of machines in **two phases**:

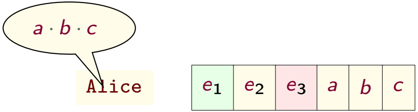
Phase I: emitted events are appended to the local log of the emitting machine



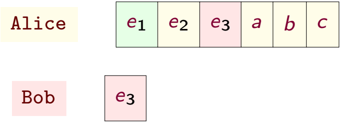
Ingredients (II): log shipping

Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)
Such events are **appended** to the logs of machines in **two phases**:

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



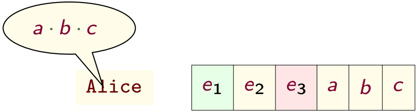
Phase II: newly emitted events are shipped to other machines



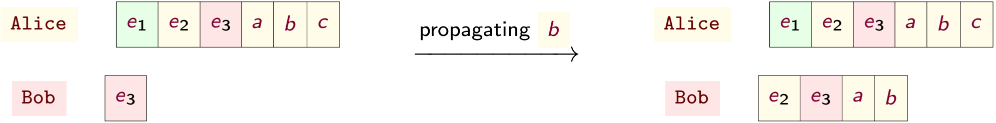
Ingredients (II): log shipping

Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)
Such events are **appended** to the logs of machines in **two phases**:

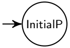
Phase I: emitted events are appended to the local log of the emitting machine



Phase II: newly emitted events are shipped to other machines

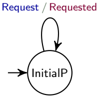


Machines by example



`InitialP` $=$

Machines by example



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

Machines by example

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

Machines by example

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

$$\texttt{AuctionP} =$$

Machines by example

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

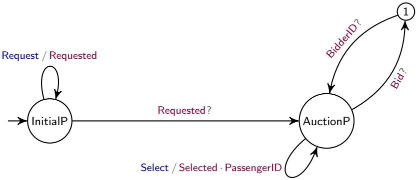
$$\texttt{AuctionP} \quad = \quad \Big[\ \ \text{Bid}\,?\ \underline{\text{BidderId}\,?\ \texttt{AuctionP}}$$

Machines by example

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

$$\texttt{AuctionP} \quad = \quad [\; \mathsf{Bid}?\, \underline{\mathsf{BidderId}?\, \texttt{AuctionP}}$$

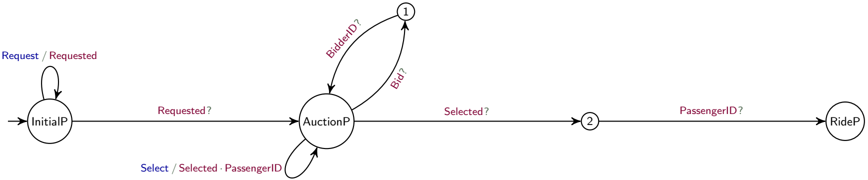
Machines by example



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

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

Machines by example



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

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

$$\texttt{RideP} = \cdots$$

Machines, formally

Fix a set of commands ranged over by c

Let κ range over finite maps from commands to non-empty log types

Machines, formally

Fix a set of commands ranged over by $\mathtt{c}$

Let κ range over finite maps from commands to non-empty log types

Machine: **deterministic regular term** of $\mathtt{M} \overset{\text{co}}{::=} \kappa \cdot [\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$

Machines, formally

Fix a set of commands ranged over by $\mathtt{c}$

Let κ range over finite maps from commands to non-empty log types

Machine: **deterministic regular term** of $\mathtt{M} \overset{\text{co}}{::=} \kappa \cdot [\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$

Think of machines as emitters/consumers of events with a semantics given in terms of 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, formally

Fix a set of commands ranged over by $\mathtt{c}$

Let κ range over finite maps from commands to non-empty log types

Machine: **deterministic regular term** of $\mathtt{M} \stackrel{\text{co}}{::=} \kappa \cdot [\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$

Think of machines as emitters/consumers of events with a semantics given in terms of 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

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

That is

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

Swarms

Swarms: $\mathtt{M}_1\boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n\boxed{\ell_n} \mid \ell$ s.t. $\ell = \bigcup_{1 \leq i \leq n} \ell_i$ and $\ell_i \sqsubseteq \ell$ for $1 \leq i \leq n$

Swarms

Swarms: $\mathtt{M}_1\boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n\boxed{\ell_n} \mid \ell$ s.t. $\ell = \bigcup_{1 \leq i \leq n} \ell_i$ and $\ell_i \sqsubseteq \ell$ for $1 \leq i \leq n$

where $\ell_1 \sqsubseteq \ell_2$ is the sublog relation defined as

- $\ell_1 \subseteq \ell_2$ and $<_{\ell_1} \subseteq <_{\ell_2}$ and

  That is
  all events of ℓ_1 appear in the same order in ℓ_2

- $e <_{\ell_2} e'$, $src(e) = src(e')$ and $e' \in \ell_1 \implies e \in \ell_1$

  That is
  the per-source partitions of ℓ_1 are prefixes of the corresponding partitions of ℓ_2

Swarms

Swarms: $\mathtt{M}_1\boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n\boxed{\ell_n} \mid \ell$ s.t. $\ell = \bigcup_{1 \le i \le n} \ell_i$ and $\ell_i \sqsubseteq \ell$ for $1 \le i \le n$

where $\ell_1 \sqsubseteq \ell_2$ is the sublog relation defined as

- $\ell_1 \subseteq \ell_2$ and $<_{\ell_1} \subseteq <_{\ell_2}$ and

That is
all events of ℓ_1 appear in the same order in ℓ_2

- $e <_{\ell_2} e'$, $src(e) = src(e')$ and $e' \in \ell_1 \implies e \in \ell_1$

That is
the per-source partitions of ℓ_1 are prefixes of the corresponding partitions of ℓ_2

The propagation of newly generated events happens by merging logs:
Log merging: $\ell_1 \bowtie \ell_2 = \{\ell \mid \ell \subseteq \ell_1 \cup \ell_2 \text{ and } \ell_1 \sqsubseteq \ell \text{ and } \ell_2 \sqsubseteq \ell\}$

Semantics of swarms

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

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

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

Plan of the talk

A motivating case study

Our formalisation

Our typing discipline

Tool support

Future work

– Behavioural types for swarms –

Inspired by choreographies

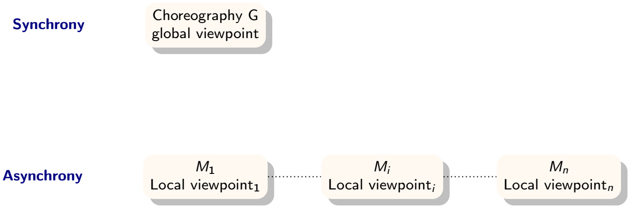
Quoting W3C:

"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local systems"

Inspired by choreographies

Quoting W3C:

"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local systems"



Inspired by choreographies

Quoting W3C:

"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local systems"

Synchrony

Choreography G
global viewpoint

Asynchrony

M_1
Local viewpoint$_1$

M_i
Local viewpoint$_i$

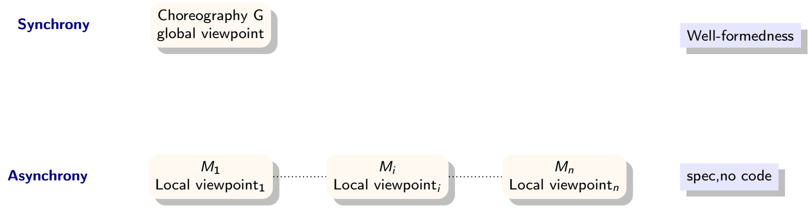
M_n
Local viewpoint$_n$

spec,no code

Inspired by choreographies

Quoting W3C:

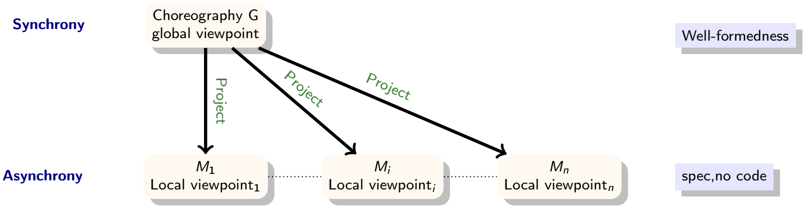
"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local systems"



Inspired by choreographies

Quoting W3C:

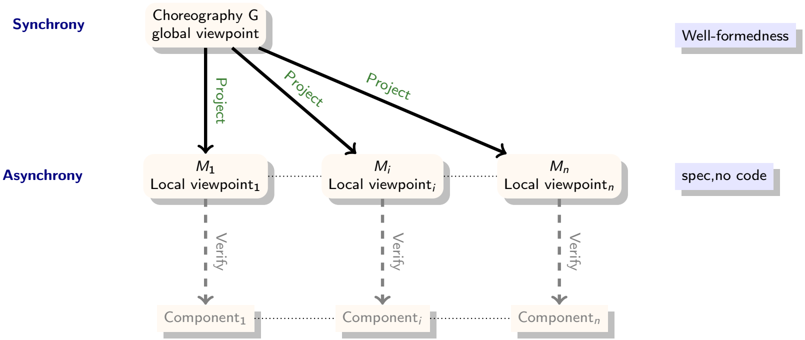
"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local systems"



Inspired by choreographies

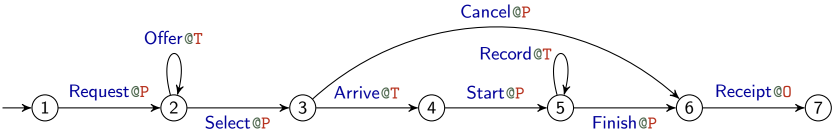
Quoting W3C:

"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local systems"



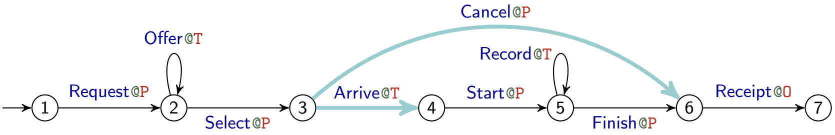
Swarm protocols by example

An intuitive auction protocol for a passenger `P` to get a taxi `T`:



Swarm protocols by example

An intuitive auction protocol for a passenger P to get a taxi T:



Swarm protocols: global type for local-first applications

An **idealised** specification relying on **synchronous communication**

Fix a set of roles ranged over by $\mathtt{R}$ (e.g., $\mathtt{P}$, $\mathtt{T}$, and $\mathtt{O}$ on slide 31)

The syntax of swarm protocols is again given co-inductively:

$$\mathtt{G} \overset{\text{co}}{::=} \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathtt{G}_i \quad \Big| \quad 0 \qquad \text{where } I \text{ is a finite set (of indexes)}$$

An example

A swarm protocol for the taxi scenario on slide 31:

$$G = \mathsf{Request}@\mathtt{P}\langle \mathsf{Requested}\rangle \,.\, G_{\mathsf{auction}}$$

$$\begin{aligned} G_{\mathsf{auction}} &= \mathsf{Offer}@\mathtt{T}\langle \mathsf{Bid}\cdot\mathsf{BidderID}\rangle \,.\, G_{\mathsf{auction}} \\ &+ \mathsf{Select}@\mathtt{P}\langle \mathsf{Selected}\cdot\mathsf{PassengerID}\rangle \,.\, G_{\mathsf{choose}} \end{aligned}$$

$$\begin{aligned} G_{\mathsf{choose}} &= \mathsf{Arrive}@\mathtt{T}\langle \mathsf{Arrived}\rangle \,.\, \mathsf{Start}@\mathtt{P}\langle \mathsf{Started}\rangle \,.\, G_{\mathsf{ride}} \\ &+ \mathsf{Cancel}@\mathtt{P}\langle \mathsf{Cancelled}\rangle \,.\, \mathsf{Receipt}@\mathtt{O}\langle \mathsf{Receipt}\rangle \,.\, 0 \end{aligned}$$

$$\begin{aligned} G_{\mathsf{ride}} &= \mathsf{Record}@\mathtt{T}\langle \mathsf{Path}\rangle \,.\, G_{\mathsf{ride}} \\ &+ \mathsf{Finish}@\mathtt{P}\langle \mathsf{Finished}\cdot\mathsf{Rating}\rangle \,.\, \mathsf{Receipt}@\mathtt{O}\langle \mathsf{Receipt}\rangle \,.\, 0 \end{aligned}$$

An example

A swarm protocol for the taxi scenario on slide 31:

$$\mathsf{G} = \mathsf{Request}@\mathtt{P}\langle\mathsf{Requested}\rangle . \mathsf{G}_{\mathsf{auction}}$$

$$\begin{aligned}\mathsf{G}_{\mathsf{auction}} = {} & \mathsf{Offer}@\mathtt{T}\langle\mathsf{Bid}\cdot\mathsf{BidderID}\rangle . \mathsf{G}_{\mathsf{auction}} \\ & + \mathsf{Select}@\mathtt{P}\langle\mathsf{Selected}\cdot\mathsf{PassengerID}\rangle . \mathsf{G}_{\mathsf{choose}}\end{aligned}$$

Note the log types in each prefixes

$$\begin{aligned}\mathsf{G}_{\mathsf{choose}} = {} & \mathsf{Arrive}@\mathtt{T}\langle\mathsf{Arrived}\rangle . \mathsf{Start}@\mathtt{P}\langle\mathsf{Started}\rangle . \mathsf{G}_{\mathsf{ride}} \\ & + \mathsf{Cancel}@\mathtt{P}\langle\mathsf{Cancelled}\rangle . \mathsf{Receipt}@\mathtt{O}\langle\mathsf{Receipt}\rangle . 0\end{aligned}$$

$$\begin{aligned}\mathsf{G}_{\mathsf{ride}} = {} & \mathsf{Record}@\mathtt{T}\langle\mathsf{Path}\rangle . \mathsf{G}_{\mathsf{ride}} \\ & + \mathsf{Finish}@\mathtt{P}\langle\mathsf{Finished}\cdot\mathsf{Rating}\rangle . \mathsf{Receipt}@\mathtt{O}\langle\mathsf{Receipt}\rangle . 0\end{aligned}$$

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 $i \in \{1 \ldots, n\}$
- 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 $\mathsf{G} \xrightarrow{\mathtt{c}_i \,/\, \mathtt{l}_i} \mathsf{G}_i$

Semantics of swarm protocols

One rule only!

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

One rule only!

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

where

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

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

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

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

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

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

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

- 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

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

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

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 $\kappa = \{(\mathtt{c}_i \,/\, \mathtt{l}_i) \mid \mathtt{R}_i = \mathtt{R} \text{ and } i \in I\}$

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 subscriptions : maps from roles to sets of event types

In pub-sub, processes subscribe to "topics"

Another attempt

Let's subscribe to subscriptions : 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

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

where

Another attempt

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

In pub-sub, processes subscribe to "topics"

Given $G = \sum_{i \in I} c_i @ R_i \langle l_i \rangle . G_i$, the projection of G on a role R with respect to subscription σ is

$$G \downarrow_R^\sigma = \kappa \cdot [\&_{j \in J} \mathsf{filter}(l_j, \sigma(R))?\, G_j \downarrow_R^\sigma]$$ where

$$\kappa = \{c_i / l_i \mid R_i = R \text{ and } i \in I\}$$

$$J = \{i \in I \mid \mathsf{filter}(l_i, \sigma(R)) \neq \epsilon\}$$

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

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 $\mathsf{G} = \sum_{i \in I} \mathsf{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i$

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

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

Command causality $\quad$ if $\mathtt{R}$ executes a command in G_i

then $\sigma(\mathtt{R}) \cap \mathtt{l}_i \neq \emptyset$ and $\sigma(\mathtt{R}) \cap \mathtt{l}_i \supseteq \bigcup_{\mathtt{R}' \in_\sigma \mathsf{G}_i} \sigma(\mathtt{R}') \cap \mathtt{l}_i$

Well-formedness = Causality + Determinacy

Trading consistency for availability has implications:

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

$\implies$ 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 $\quad \sigma(\mathtt{R}_i) \cap \mathtt{l}_i \neq \emptyset$

Command causality $\quad$ 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 $\quad \mathtt{R} \in_\sigma \mathtt{G}_i \implies \mathtt{l}_i[0] \in \sigma(\mathtt{R})$

Well-formedness = Causality + Determinacy - Confusion

Trading consistency for availability has implications:

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

$\implies$ 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 $\mathsf{G} = \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i$

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

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

Determinacy $\mathtt{R} \in_\sigma \mathsf{G}_i \implies \mathtt{l}_i[0] \in \sigma(\mathtt{R})$

Confusion freeness for each $\mathtt{t}$ starting a log emitted by a command in G
there is a unique state G' reachable from G which emits $\mathtt{t}$

Implementations

Write $\ell \equiv_{\mathsf{G},\sigma} \ell'$ when ℓ and ℓ' have the same effective type wrt G and σ

A swarm $(\mathtt{S}, \epsilon)$ is eventually faithful to G and σ if $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}, \ell)$ then there is $(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell')$ with $\ell \equiv_{\mathsf{G},\sigma} \ell'$

Implementations

Write $\ell \equiv_{\mathsf{G},\sigma} \ell'$ when ℓ and ℓ' have the same effective type wrt G and σ

A swarm $(\mathtt{S}, \epsilon)$ is eventually faithful to G and σ if $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}, \ell)$ then there is $(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell')$ with $\ell \equiv_{\mathsf{G},\sigma} \ell'$

A (σ, G)-realisation is a swarm $(\mathtt{S}, \epsilon)$ such that, for each $i \in \operatorname{dom} \mathtt{S}$, there exists a role $\mathtt{R} \in \mathsf{roles}(\mathsf{G}, \sigma)$ such that $\mathtt{S}(i) = \mathsf{G} \downarrow^{\sigma}_{\mathtt{R}}$ $\square$

Implementations & projections

Write $\ell \equiv_{\mathsf{G},\sigma} \ell'$ when ℓ and ℓ' have the same effective type wrt G and σ

A swarm $(\mathtt{S}, \epsilon)$ is eventually faithful to G and σ if $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}, \ell)$ then there is $(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell')$ with $\ell \equiv_{\mathsf{G},\sigma} \ell'$

A (σ, G)-realisation is a swarm $(\mathtt{S}, \epsilon)$ such that, for each $i \in \operatorname{dom} \mathtt{S}$, there exists a role $\mathtt{R} \in \mathrm{roles}(\mathsf{G}, \sigma)$ such that $\mathtt{S}(i) = \mathsf{G} \downarrow^{\sigma}_{\mathtt{R}}$ $\square$

Lemma (Projections of well-formed protocols are eventually faithful)

If G is a σ-WF protocol and $\big(\delta(\mathsf{G} \downarrow^{\sigma}_{\mathtt{R}}, \ell)\big) \downarrow_{\mathtt{c}\,/\,\mathtt{1}}$ then there exists $\ell' \equiv_{\mathsf{G},\sigma} \ell$ such that $(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell')$ and $\delta(\mathsf{G}, \ell') \xrightarrow{\mathtt{c}\,/\,\mathtt{1}} \mathsf{G}'$

On correct realisations

$$\begin{tikzcd}[row sep=small] & (\mathtt{s}, \ell_1) \\ (\mathtt{s}, \epsilon) \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\ & (\mathtt{s}, \ell_2) \end{tikzcd}$$

consistent if there is ℓ s.t.

$$\begin{tikzcd}[row sep=small] & & (\mathtt{s}, \ell'_1) \\ (\mathtt{s}, \epsilon) \arrow[r, Rightarrow] & (\mathtt{s}, \ell) \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\ & & (\mathtt{s}, \ell'_2) \end{tikzcd}$$

with $\ell_1 = \ell \cdot \ell'_1$ and $\ell_2 = \ell \cdot \ell'_2$ and $\ell'_1 \cap \ell'_2 = \emptyset$

A set of runs is consistent when its elements are pair-wise consistent

On correct realisations

$$(\mathtt{s}, \epsilon) \text{ with } \nearrow (\mathtt{s}, \ell_1) \text{ and } \searrow (\mathtt{s}, \ell_2)$$

consistent if there is ℓ s.t.

$$(\mathtt{s}, \epsilon) \Longrightarrow (\mathtt{s}, \ell) \text{ with } \nearrow (\mathtt{s}, \ell'_1) \text{ and } \searrow (\mathtt{s}, \ell'_2)$$

with $\ell_1 = \ell \cdot \ell'_1$ and $\ell_2 = \ell \cdot \ell'_2$ and $\ell'_1 \cap \ell'_2 = \emptyset$

A set of runs is consistent when its elements are pair-wise consistent

Notation

For $(\mathtt{G}, \epsilon) \xrightarrow{\mathtt{c}_1 \,/\, \mathtt{l}_1} (\mathtt{G}, \ell_1) \xrightarrow{\mathtt{c}_2 \,/\, \mathtt{l}_2} \cdots \xrightarrow{\mathtt{c}_n \,/\, \mathtt{l}_n} (\mathtt{G}, \overbrace{\ell_1 \cdot \ell_2 \cdot \cdots \cdot \ell_n}^{=\ell})$
let $\ell^{(j)} = \ell_1 \cdot \cdots \cdot \ell_j$

On correct realisations

$$\begin{array}{l} \\ (\mathsf{s}, \epsilon) \\ \\ \end{array}\!\!\begin{array}{l} \nearrow (\mathsf{s}, \ell_1) \\ \\ \searrow (\mathsf{s}, \ell_2) \end{array}$$

consistent if there is ℓ s.t. $(\mathsf{s}, \epsilon) \Longrightarrow (\mathsf{s}, \ell) \begin{array}{l} \nearrow (\mathsf{s}, \ell_1') \\ \\ \searrow (\mathsf{s}, \ell_2') \end{array}$ with $\ell_1 = \ell \cdot \ell_1'$ and $\ell_2 = \ell \cdot \ell_2'$ and $\ell_1' \cap \ell_2' = \emptyset$

A set of runs is consistent when its elements are pair-wise consistent

Notation

For $(\mathsf{G}, \epsilon) \xrightarrow{\mathtt{c}_1 \,/\, \mathtt{l}_1} (\mathsf{G}, \ell_1) \xrightarrow{\mathtt{c}_2 \,/\, \mathtt{l}_2} \cdots \xrightarrow{\mathtt{c}_n \,/\, \mathtt{l}_n} (\mathsf{G}, \overbrace{\ell_1 \cdot \ell_2 \cdots \cdot \ell_n}^{=\ell})$
let $\ell^{(j)} = \ell_1 \cdot \cdots \cdot \ell_j$

Admissibility

A log ℓ is admissible for a σ-WF protocol G if there are consistent runs $\{(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell_i)\}_{1 \le i \le k}$ and a log $\ell' \in (\bowtie_{1 \le i \le k} \ell_i)$ such that

$$\ell = \bigcup_{1 \le i \le k} \ell_i, \quad \ell' \equiv_{\mathsf{G},\sigma} \ell, \quad \text{and} \quad \ell_i^{(j)} \sqsubseteq \ell \text{ for all } 1 \le i \le k$$

Results

Let G be well-formed; a realisation is a swarm whose components are projections of G

Lemma (Well-formedness generates any admissible log)

If ℓ is admissible for G then there is a log ℓ' such that $(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell')$ and $\ell \equiv_{\mathsf{G},\sigma} \ell'$

Theorem (Well-formed protocols generate only admissible logs)

If $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}', \ell)$ for $(\mathtt{S}, \epsilon)$ realisation of G then ℓ is admissible for G

Corollary

Every realisation of G is eventually faithful wrt G and σ

Theorem (Full realisations are complete)

If $\mathtt{S}$ is a full realisation of G and $(\mathsf{G}, \epsilon) \Longrightarrow (\mathsf{G}, \ell')$ then there is $\mathtt{S}'$ s.t. $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}', \ell)$

Plan of the talk

A motivating case study

Our formalisation

Our typing discipline

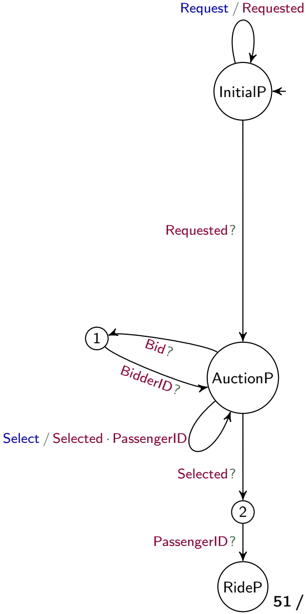
Tool support

Future work

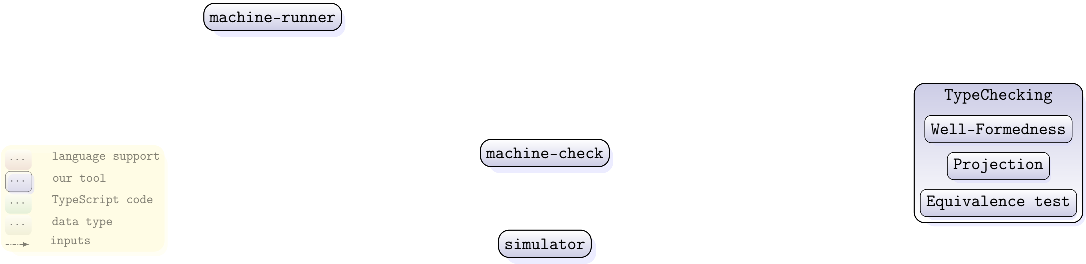
– Tooling –

```
// 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> { ... }
```

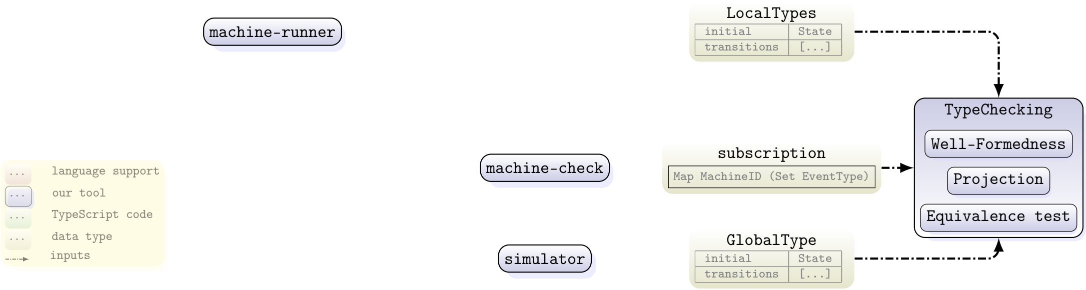


Architecture



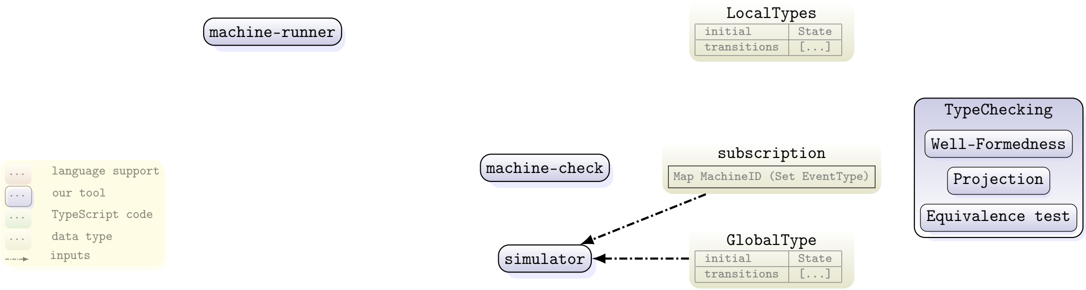
- TypeChecking implements the functionalities of our typing discipline
- simulator simulates the semantics of swarm realisations
- machine-check and machine-runner integrate our framework in the Actyx platform

Architecture



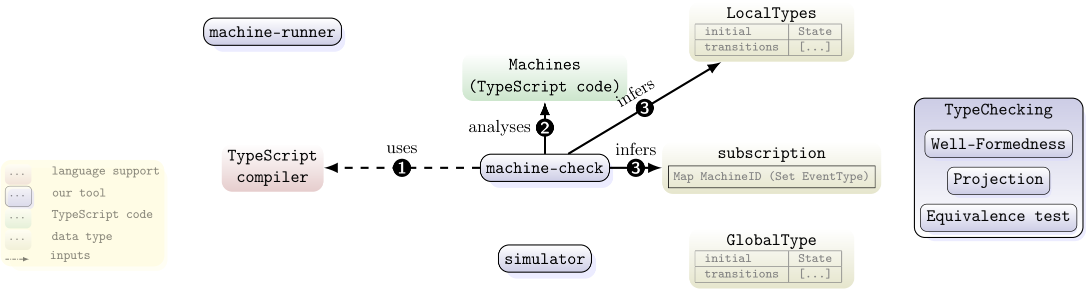
- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

Architecture



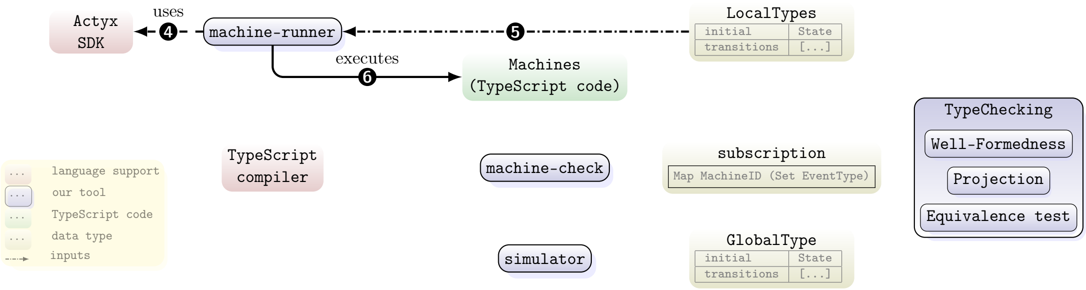
- TypeChecking implements the functionalities of our typing discipline
- simulator simulates the semantics of swarm realisations
- machine-check and machine-runner integrate our framework in the Actyx platform

Architecture



- 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



- 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

If you want to play with our prototype?

Have a look at

- our ECOOP artifact paper (not online yet; extended version at `https://arxiv.org/abs/2305.04848`)
- code at `https://doi.org/10.5281/zenodo.7737188`
- An ISSTA tool paper from Actyx (`https://arxiv.org/abs/2306.09068`)

Plan of the talk

A motivating case study

Our formalisation

Our typing discipline

Tool support

Future work

– Epilogue –

To be continued....

There are a number of future directions to explore:

To be continued....

There are a number of future directions to explore:

Identify weaker conditions for well-formedness

To be continued....

There are a number of future directions to explore:

- Identify weaker conditions for well-formedness
- “Efficiency”

To be continued....

There are a number of future directions to explore:

- Identify weaker conditions for well-formedness
- “Efficiency”
- Subscriptions are hard to determine

To be continued....

There are a number of future directions to explore:

- Identify weaker conditions for well-formedness
- “Efficiency”
- Subscriptions are hard to determine
- Relax some of our assumptions

To be continued....

There are a number of future directions to explore:

- Identify weaker conditions for well-formedness
- "Efficiency"
- Subscriptions are hard to determine
- Relax some of our assumptions
  - Compensations
  - Unreliable propagation

To be continued....

There are a number of future directions to explore:

- Identify weaker conditions for well-formedness
- "Efficiency"
- Subscriptions are hard to determine
- Relax some of our assumptions
  - Compensations
  - Unreliable propagation
  - Adversarial contexts

To be continued....

There are a number of future directions to explore:

- Identify weaker conditions for well-formedness
- "Efficiency"
- Subscriptions are hard to determine
- Relax some of our assumptions
  - Compensations
  - Unreliable propagation
  - Adversarial contexts
- ....

Summary

An interesting paradigm grounded on principles for local-first principles: temporary inconsistency are tolerated provided that they can be (and are) resolved at some point

Summary

An interesting paradigm grounded on principles for local-first principles: temporary inconsistency are tolerated provided that they can be (and are) resolved at some point

A formal semantics that faithfully captures Actyx's platform

Summary

An interesting paradigm grounded on principles for local-first principles: temporary inconsistency are tolerated provided that they can be (and are) resolved at some point

A formal semantics that faithfully captures Actyx's platform

and behavioural types to specify and verify eventual consensus

Thank you!