Local-First Principles: a Behavioural Types Approach

Emilio Tuosto @ GSSI

joint work with

Roland Kuhn @ Actyx and Hernán Melgratti @ UBA

Tutorial at Discotec 2023
Lisbon 23 June, 2023

– Prelude –

Take-away message

Trade consistency for availability in system made of asymmetric replicated peers

Use local-first's principles and (re-)gain consistency ... eventually

A behavioural typing discipline supporting local-first principles for pub-sub P2P systems!

Take-away message

Trade consistency for availability in system made of asymmetric replicated peers

Use local-first's principles and (re-)gain consistency ... eventually

A behavioural typing discipline supporting local-first principles for pub-sub P2P systems!

- swarm protocols: systems from a global viewpoint
- machines: peers
- enforce good behaviour via behavioural typing

Take-away message

Trade consistency for availability in system made of asymmetric replicated peers

Use local-first's principles and (re-)gain consistency ... eventually

A behavioural typing discipline supporting local-first principles for pub-sub P2P systems!

- swarm protocols: systems from a global viewpoint
- machines: peers
- 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
Distributed consensus
Commutative replicated data types
...

Local-first...first

Autonomy

Thou shall collaborate and be autonomous
Thou shall recognise and embrace conflicts
Thou shall be consistent

Some implications

- peers are collaborative
- peers can locally make progress at all times...even under partial knowledge
- peers embrace inconsistency
- peers resolve conflicts, eventually

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	eggs
sugar	

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	eggs
sugar	pecorino romano cheese

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	eggs
sugar	pecorino romano cheese
	spaghetti

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	eggs
sugar	pecorino romano cheese
ground moka coffee	spaghetti

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	eggs
sugar	pecorino romano cheese
ground moka coffee	spaghetti
savoiardi biscuits	

Local-First at work

Alice and Bob decided to have spaghetti carbonara and tiramisù. They use a mobile app to share a grocery list.

Alice's mobile	Bob's mobile
mascarpone cheese	smoked guanciale
eggs	eggs
sugar	pecorino romano cheese
ground moka coffee	spaghetti
savoiardi biscuits	

Who should buy the eggs?

Plan of the talk

A motivating case study

Plan of the talk

A motivating case study

Our formalisation

Plan of the talk

A motivating case study

Our formalisation

Our typing discipline

Plan of the talk

A motivating case study

Our formalisation

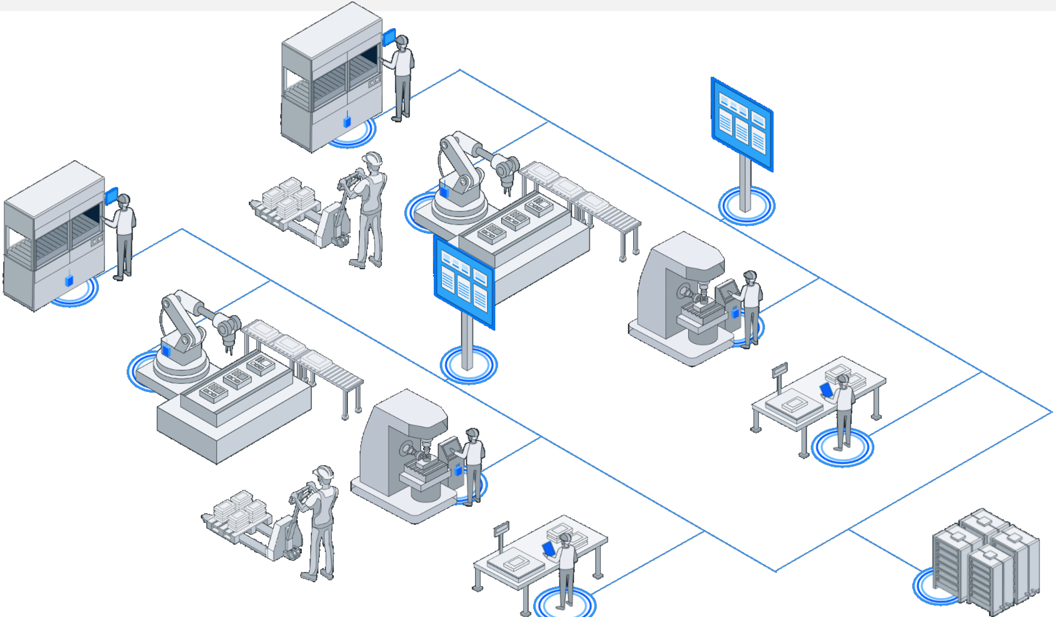
Our typing discipline

Tool support

– Motivations –

A collaborative environment and its execution model

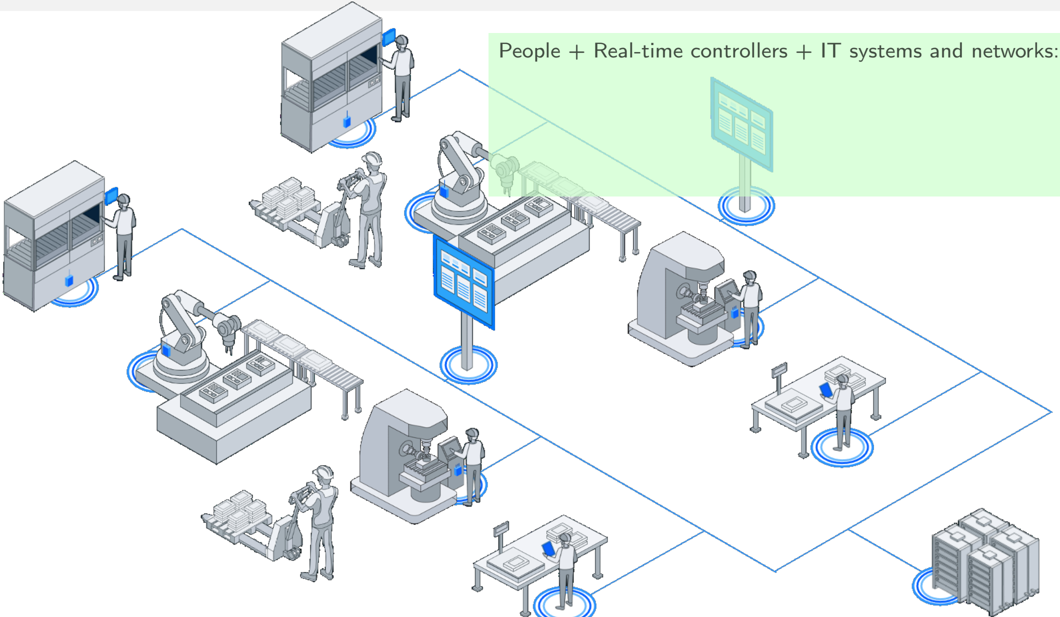
(the pictures are courtesy of Actyx AG)



A collaborative environment and its execution model

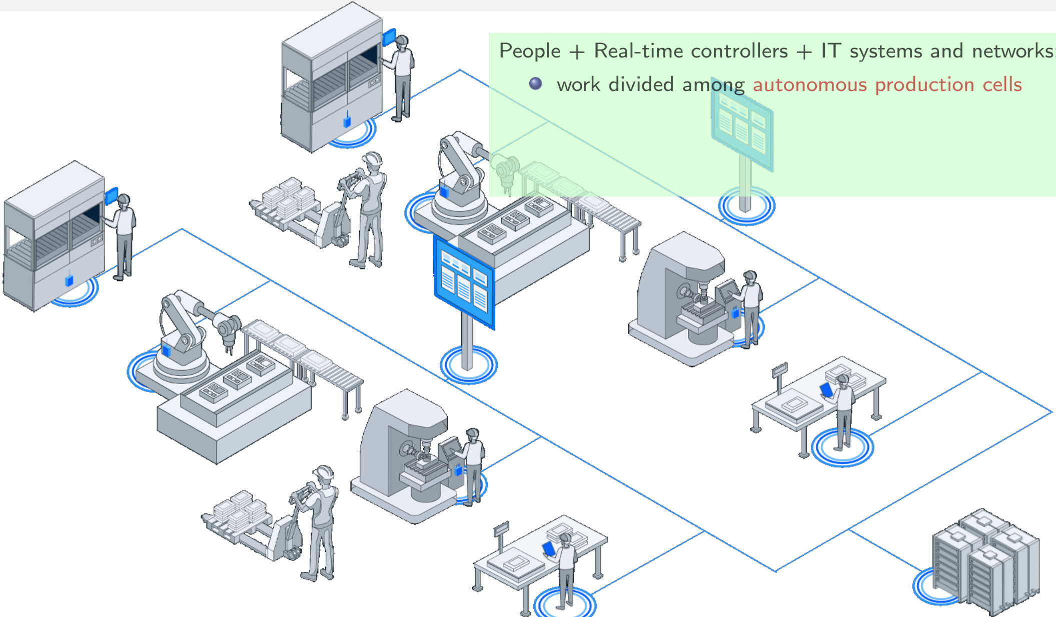
(the pictures are courtesy of Actyx AG)

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



A collaborative environment and its execution model

(the pictures are courtesy of Actyx AG)

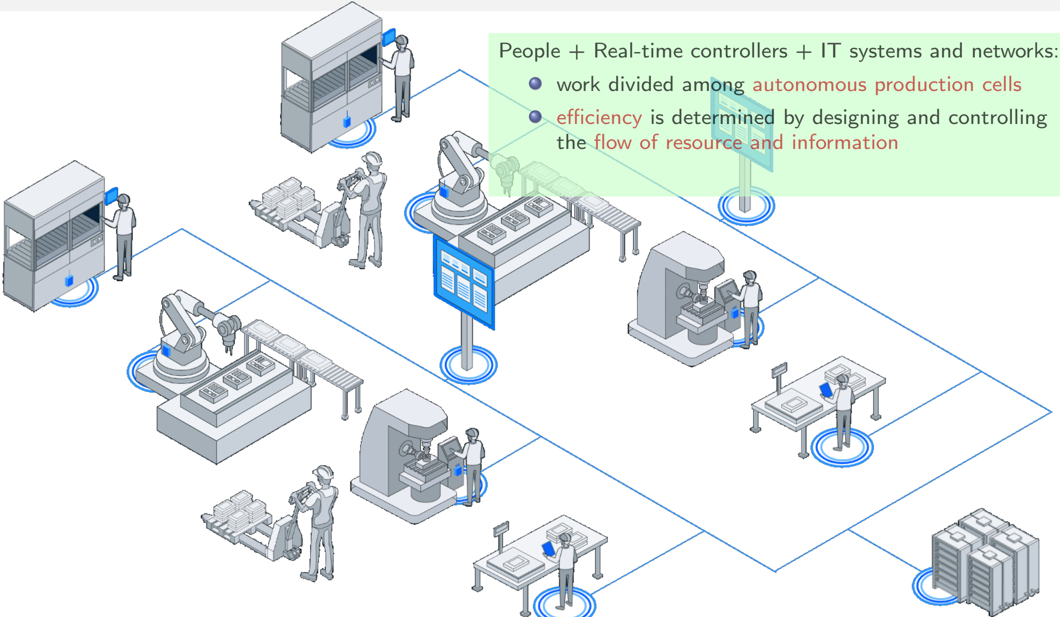


People + Real-time controllers + IT systems and networks:

- work divided among autonomous production cells

A collaborative environment and its execution model

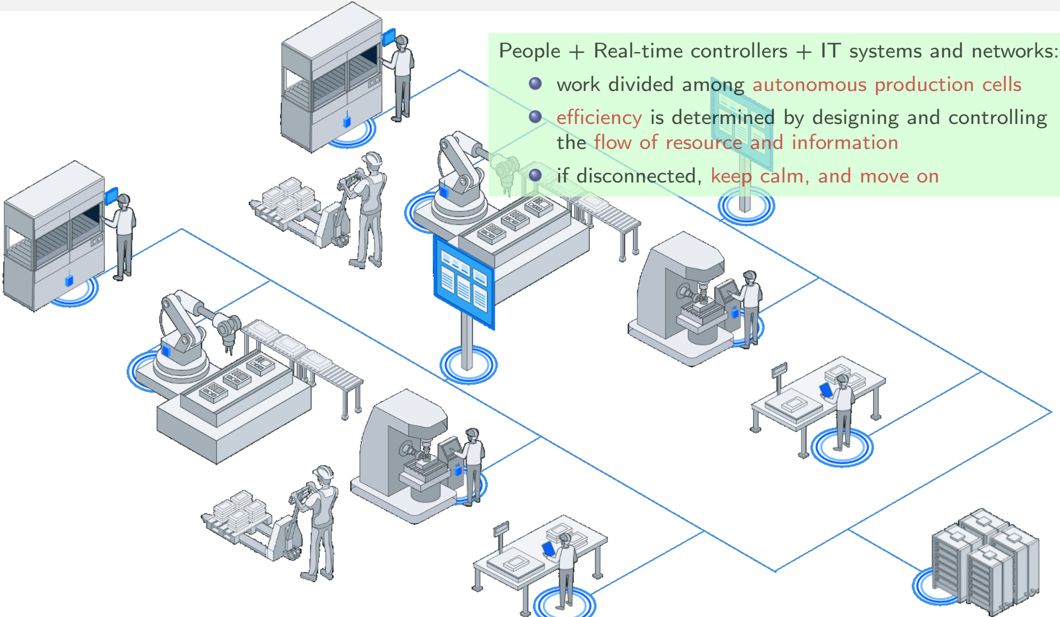
(the pictures are courtesy of Actyx AG)



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

A collaborative environment and its execution model



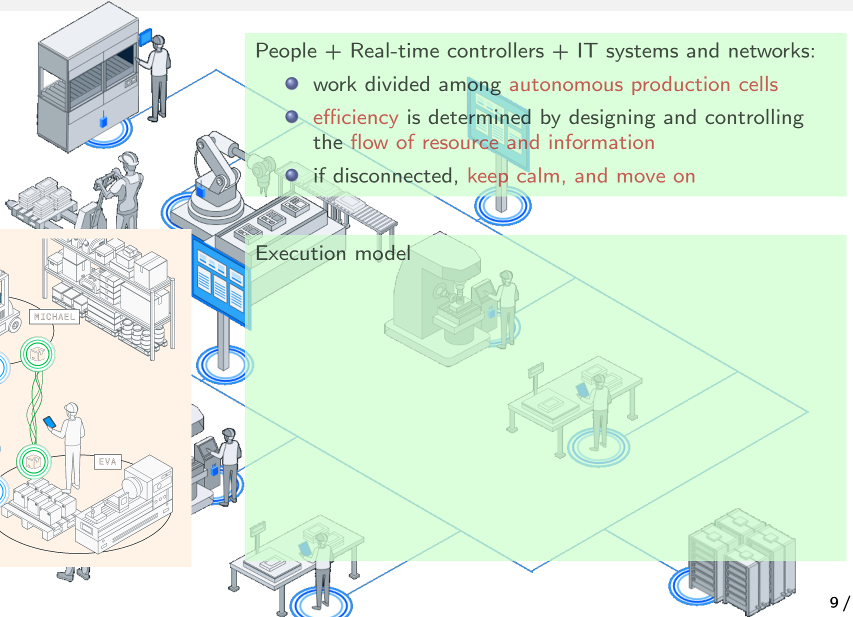
(the pictures are courtesy of Actyx AG)

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

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

A collaborative environment and its execution model

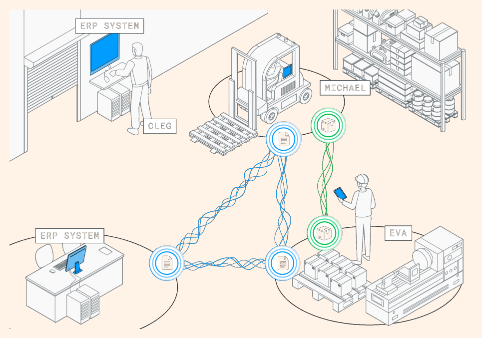
of Actyx AG)



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

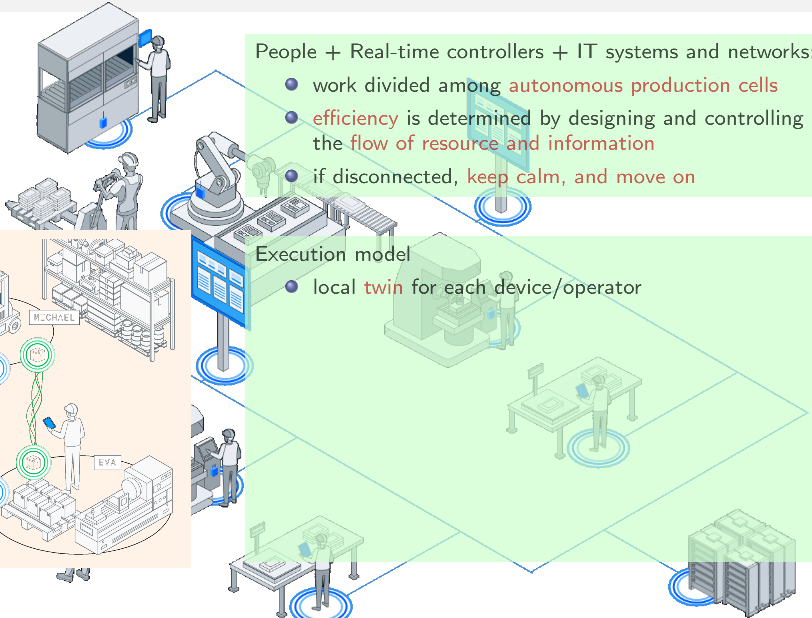
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

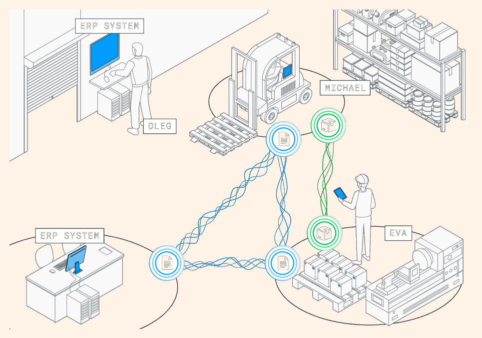
Execution model



A collaborative environment and its execution model

of Actyx AG)





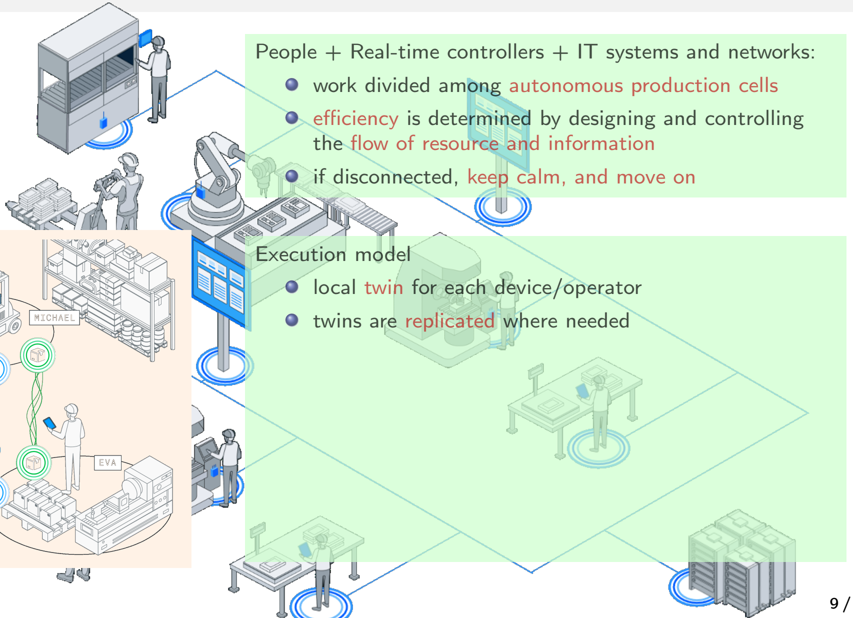
People + Real-time controllers + IT systems and networks:

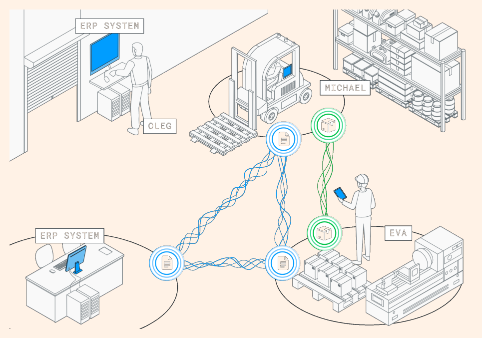
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

Execution model

- local twin for each device/operator

A collaborative environment and its execution model





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

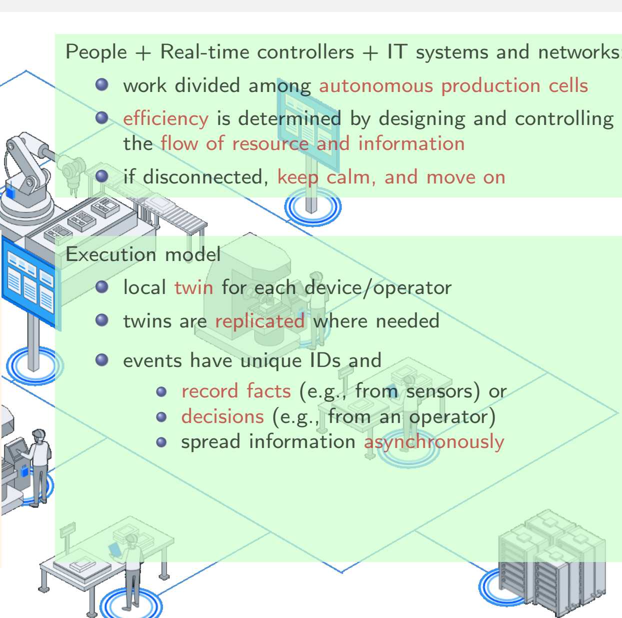
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

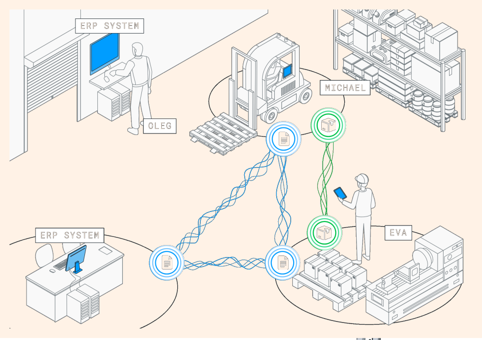
Execution model

- 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:

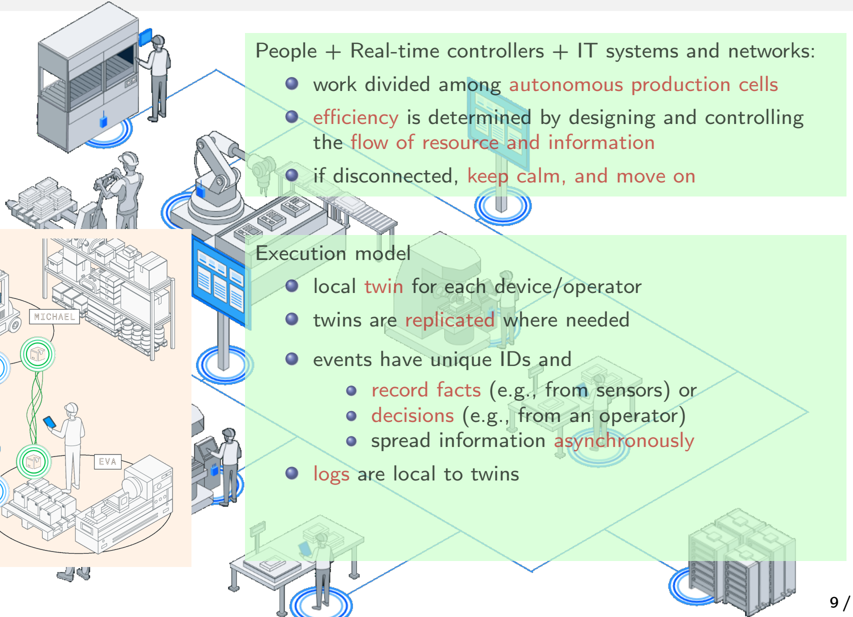
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

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

A collaborative environment and its execution model

of Actyx AG)

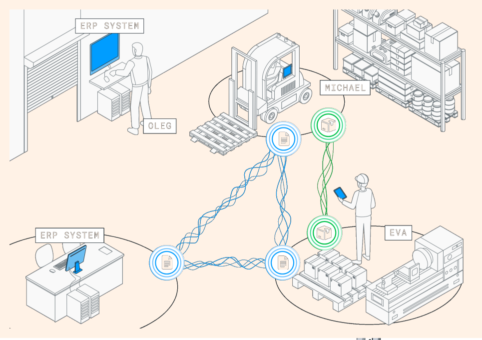


People + Real-time controllers + IT systems and networks:

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

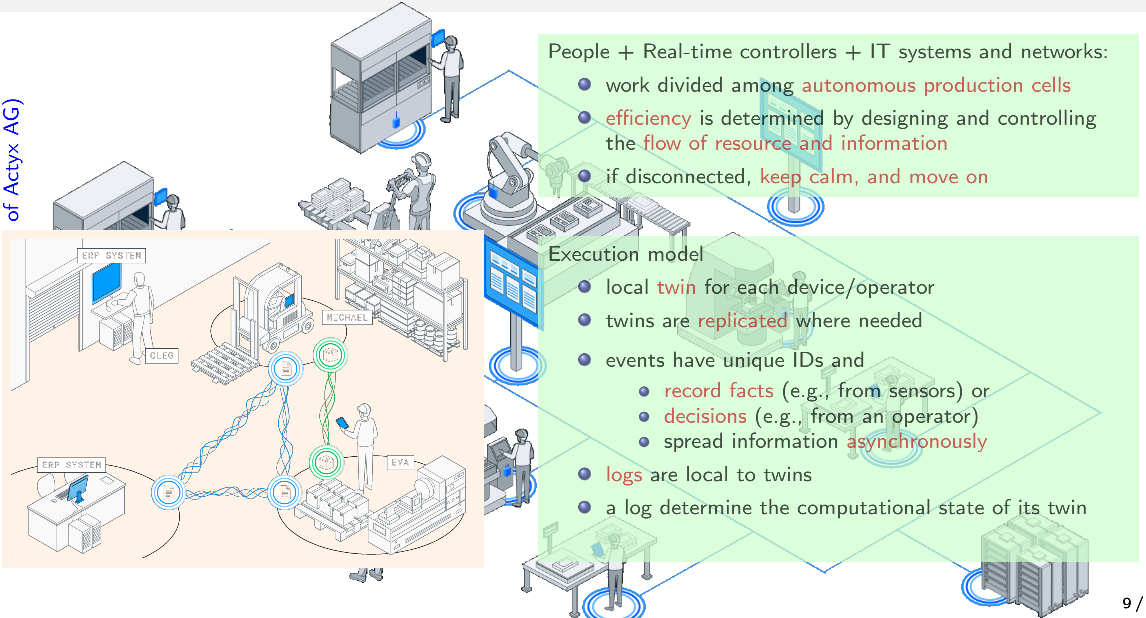
Execution model

- 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 collaborative environment and its execution model

of Actyx AG)



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

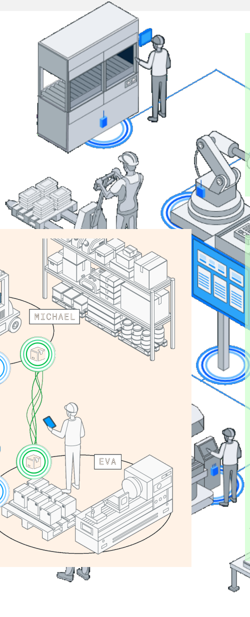
- work divided among autonomous production cells
- efficiency is determined by designing and controlling the flow of resource and information
- if disconnected, keep calm, and move on

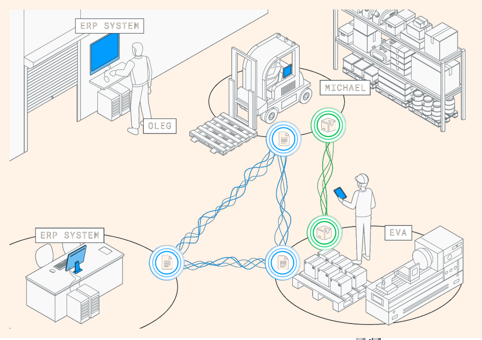
Execution model

- 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 determine 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
- 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 determine the computational state of its twin
- replicated logs are merged

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

Challenges

Specify application-level protocols where decisions

- don't require consensus

Challenges

Specify application-level protocols where decisions

- don't require consensus
- are based on stale local states

Challenges

Specify application-level protocols where decisions

- don't require consensus
- are based on stale local states
- yet, collaboration has to be successful

– A formal model –

Ingredients (I): events & logs

Events

$$e$$

$$src(e)$$

Logs

$$e_1 \cdot e_2 \ldots$$

Ingredients (I): events & logs

$$\text{Events} \quad \vdash \quad e \quad : \quad t$$

$$src(e)$$

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

Ingredients (I): events & logs

Events $\vdash$ e : $\mathtt{t}$

$$src(e)$$

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

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)

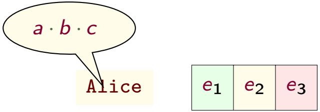
Machine `Alice` **emits** logs upon **execution** of commands (we'll see how in a moment)

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**:

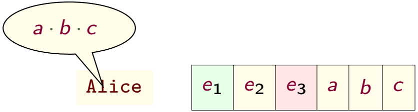
Phase I: emitted events are appended to the local log of the emitting machine



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**:

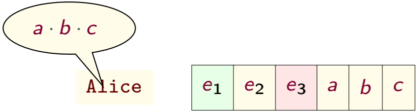
Phase I: emitted events are appended to the local log of the emitting machine



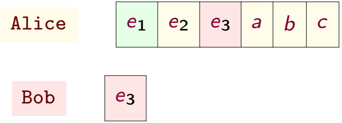
Ingredients (II): log shipping

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



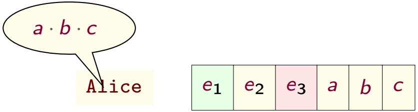
Phase II: newly emitted events are shipped to other machines



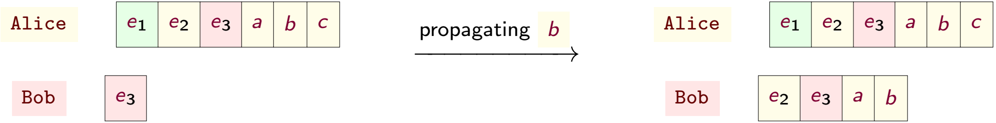
Ingredients (II): log shipping

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

Fix a set of commands ranged over by c

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

Machines

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

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

A machine is a **regular term** of this co-inductive grammar

$$\mathtt{M} \overset{\text{co}}{::=} \kappa\cdot[\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$$

for $i \in \{1 \ldots, n\}$, the guard of the i-th branch is $\mathtt{t}_i$

An infinite tree is regular when it has finitely-many subtrees The subtrees of $\mathtt{M} = \kappa\cdot[\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$ are $\mathtt{M}$ plus the subtrees of each $\mathtt{M}_i$

An example

Passenger $\texttt{P}$ launches an auction for a taxi $\texttt{T}$

$$
\begin{array}{lcl}
\texttt{InitialP} & = & \mathsf{Request} \mapsto \mathsf{Requested} \cdot [\mathsf{Requested}\,?\,\texttt{AuctionP}] \\[2ex]
\texttt{AuctionP} & = & \mathsf{Select} \mapsto \mathsf{Selected} \cdot \mathsf{PassengerId} \cdot [\\
 & & \qquad \mathsf{Bid}\,?\,\mathsf{BidderId}\,?\,\texttt{AuctionP} \\
 & & \qquad \& \\
 & & \qquad \mathsf{Selected}\,?\,\mathsf{PassengerId}\,?\,\texttt{RideP} \\
 & &] \\[2ex]
\texttt{RideP} & = & \cdots
\end{array}
$$

An example

Passenger $\mathtt{P}$ launches an auction for a taxi $\mathtt{T}$

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

$$\begin{aligned}\mathtt{AuctionP} = \; & \mathsf{Select} \mapsto \mathsf{Selected} \cdot \mathsf{PassengerId} \cdot [\\ & \qquad \mathsf{Bid}\,?\,\mathsf{BidderId}\,?\,\mathtt{AuctionP}\\ & \qquad \&\\ & \qquad \mathsf{Selected}\,?\,\mathsf{PassengerId}\,?\,\mathtt{RideP}\\ &]\end{aligned}$$

$$\mathtt{RideP} = \cdots$$

Notation

- abbreviate $\kappa \cdot [\mathtt{t}_1\,?\,\mathtt{M}_1 \;\&\; \cdots \;\&\; \mathtt{t}_n\,?\,\mathtt{M}_n]$ as $\kappa \cdot \mathtt{0}$ when $n = 0$
- write $\&_{1 \le i \le n}\, \mathtt{l}_i\,?\,\mathtt{M}_i$ in place of $\mathtt{t}_1\,?\,\mathtt{M}_1 \;\&\; \cdots \;\&\; \mathtt{t}_n\,?\,\mathtt{M}_n$

Treat κ *as its graph and e.g. write* $\mathtt{c} \,/\, \mathtt{l} \in \kappa$ *for* $\kappa(\mathtt{c}) = \mathtt{l}$ *or write* κ *as* $\{\mathtt{c}_1 \,/\, \mathtt{l}_1, \ldots, \mathtt{c}_h \,/\, \mathtt{l}_h\}$ *when* $\kappa : \mathtt{c}_i \mapsto \mathtt{l}_i$ *for* $i \in \{1, \ldots h\}$

Machines as automata

A machine $\mathtt{M} = \kappa \cdot [\mathtt{t}_1 ? \mathtt{M}_1 \mathbin{\&} \cdots \mathbin{\&} \mathtt{t}_n ? \mathtt{M}_n]$ is an FSA where:

- κ yields command-enabling transitions
- a branch $\mathtt{t}_i ? \mathtt{M}_i$ yields a transition $\mathtt{M} \xrightarrow{\mathtt{t}_i ?} \mathtt{M}_i$ when an event of type $\mathtt{t}_i$ is consumed

Machines as automata

A machine $\mathtt{M} = \kappa\cdot[\mathtt{t}_1 ? \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ? \mathtt{M}_n]$ is an FSA where:

- κ yields command-enabling transitions
- a branch $\mathtt{t}_i ? \mathtt{M}_i$ yields a transition $\mathtt{M} \xrightarrow{\mathtt{t}_i ?} \mathtt{M}_i$ when an event of type $\mathtt{t}_i$ is consumed

From machines to FSAs

- the states of the automaton are the subtrees of $\mathtt{M}$
- the initial state is $\mathtt{M}$ and
  - there is a self-loop transition to $\mathtt{M}$ labelled $\mathtt{c} \,/\, \mathtt{l}$ for each $\mathtt{c} \,/\, \mathtt{l} \in \kappa$
  - there is a transition labelled $\mathtt{t}_i ?$ to state $\mathtt{M}_i$ for each $i \in \{1 \ldots, n\}$
  - and likewise for $\mathtt{M}_i$

Machines as automata

A machine $\mathtt{M} = \kappa \cdot [\mathtt{t}_1 ?\, \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ?\, \mathtt{M}_n]$ is an FSA where:

- κ yields command-enabling transitions
- a branch $\mathtt{t}_i ?\, \mathtt{M}_i$ yields a transition $\mathtt{M} \xrightarrow{\mathtt{t}_i ?} \mathtt{M}_i$ when an event of type $\mathtt{t}_i$ is consumed

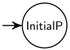
From machines to FSAs

- the states of the automaton are the subtrees of $\mathtt{M}$
- the initial state is $\mathtt{M}$ and
  - there is a self-loop transition to $\mathtt{M}$ labelled $\mathtt{c} \,/\, \mathtt{l}$ for each $\mathtt{c} \,/\, \mathtt{l} \in \kappa$
  - there is a transition labelled $\mathtt{t}_i ?$ to state $\mathtt{M}_i$ for each $i \in \{1 \ldots, n\}$
  - and likewise for $\mathtt{M}_i$

This construction yields a finite-state automaton by the regularity of $\mathtt{M}$

An example

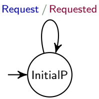
Let's build the FSA of the machine `InitialP` on slide 16.



`InitialP` $=$

An example

Let's build the FSA of the machine `InitialP` on slide 16.



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

An example

Let's build the FSA of the machine `InitialP` on slide 16.

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

An example

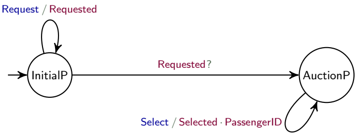
Let's build the FSA of the machine `InitialP` on slide 16.

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

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

An example

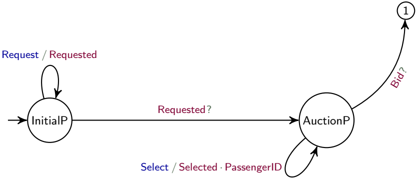
Let's build the FSA of the machine `InitialP` on slide 16.



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

$$\mathtt{AuctionP} = \mathsf{Select} \mapsto \mathsf{Selected} \cdot \mathsf{PassengerId} \cdot$$

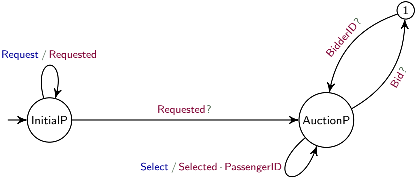
Let's build the FSA of the machine `InitialP` on slide 16.



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

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

Let's build the FSA of the machine `InitialP` on slide 16.

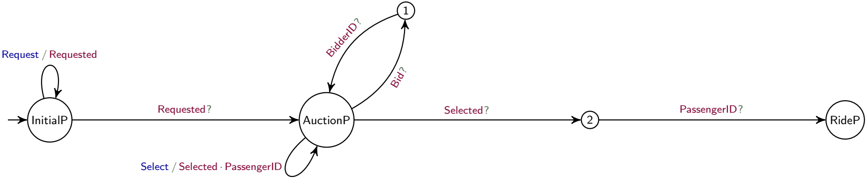


$$\texttt{InitialP} \quad = \quad \text{Request} \mapsto \text{Requested} \cdot \left[\text{Requested}?\, \underline{\texttt{AuctionP}}\right]$$

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

An example

Let's build the FSA of the machine `InitialP` on slide 16.



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

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

$$\mathtt{RideP} \quad = \quad \cdots$$

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 where transitions are

- either self-loops (determined by the κ part)
- or event consuptions (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 where transitions are

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

We restrict to **deterministic** machines and treat them as emitters/consumers of events with a semantics given in terms of state transition function :

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

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

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 where transitions are

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

We restrict to **deterministic** machines and treat them 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' semantics

So, think of $\mathtt{M} = \kappa\cdot[\mathtt{t}_1 ?\, \mathtt{M}_1 \,\&\, \cdots \,\&\, \mathtt{t}_n ?\, \mathtt{M}_n]$ as an FSA where transitions are

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

We restrict to **deterministic** machines and treat them 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

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

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 where transitions are

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

We restrict to **deterministic** machines and treat them as emitters/consumers of events with a semantics given in terms of state transition function :

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

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

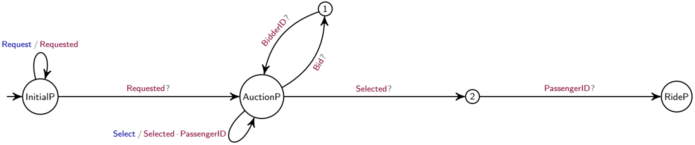
That is
M with local log ℓ is in the implicit state $\delta(\mathtt{M}, \ell)$ reached after processing each event in ℓ

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

That is
after processing the events in ℓ, 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 M

An example

Take the machine $\texttt{InitialP}$ (slide 18) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\nvdash \mathit{ignoreMe} : \mathsf{Requested}$ and $\nvdash \mathit{ignoreMeToo} : \mathsf{Requested}$

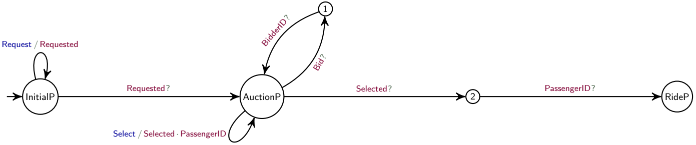


By definition of δ

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

An example

Take the machine $\texttt{InitialP}$ (slide 18) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\nvdash \mathit{ignoreMe} : \mathsf{Requested}$ and $\nvdash \mathit{ignoreMeToo} : \mathsf{Requested}$

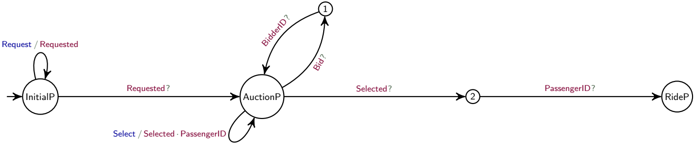


By definition of δ

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

An example

Take the machine $\texttt{InitialP}$ (slide 18) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\nvdash \mathit{ignoreMe} : \mathsf{Requested}$ and $\nvdash \mathit{ignoreMeToo} : \mathsf{Requested}$

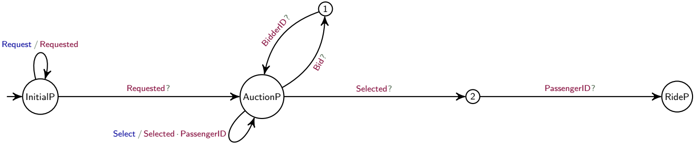


By definition of δ

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

An example

Take the machine $\texttt{InitialP}$ (slide 18) with a local log $\ell = \mathit{ignoreMe} \cdot \mathit{ignoreMeToo}$ where $\nvdash \mathit{ignoreMe} : \mathsf{Requested}$ and $\nvdash \mathit{ignoreMeToo} : \mathsf{Requested}$



By definition of δ

- $\delta(\texttt{InitialP}, \ell) = \texttt{InitialP}$ hence
- $\delta(\texttt{InitialP}, \ell) \xrightarrow{\mathsf{Request} \;/\; \mathsf{Requested}} \delta(\texttt{InitialP}, \ell)$
- $(\texttt{InitialP}, \ell) \xrightarrow{\mathsf{Request} \;/\; \mathsf{Requested}} (\texttt{InitialP}, \ell \cdot \mathit{Requested})$ hence with $\vdash \mathit{Requested} : \mathsf{Request}$ and $\mathit{src}(\mathit{Requested}) = \texttt{P}$ is possible

Exercise

Calculate $\delta(\texttt{InitialP}, \ell \cdot \mathit{Requested})$.

Some considerations

The commands are enabled only from the state reached after processing all the events in the local log of the machine

Some considerations

The commands are enabled only from the state reached after processing all the events in the local log of the machine

Deterministic machines may have non-deterministic behaviour!
Recall: commands are triggered by the environment

Some considerations

The commands are enabled only from the state reached after processing all the events in the local log of the machine

Deterministic machines may have non-deterministic behaviour!
Recall: commands are triggered by the environment

We have formalised the emission of events and their consumption
We now focus on the formalisation of log shipping

Swarms

A swarm (of size n) is a pair $(\mathtt{S}, \ell)$ where

- $\mathtt{S}$ maps each index $1 \leq i \leq n$ to a pair $(\mathtt{M}_i, \ell_i)$
- ℓ is the (global) log

Swarms

A swarm (of size n) is a pair $(\mathtt{S}, \ell)$ where

- $\mathtt{S}$ maps each index $1 \leq i \leq n$ to a pair $(\mathtt{M}_i, \ell_i)$
- ℓ is the (global) log

Notation

$\mathtt{M}_1 \boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n \boxed{\ell_n} \mid \ell$

Swarms

A swarm (of size n) is a pair $(\mathtt{S}, \ell)$ where

- $\mathtt{S}$ maps each index $1 \le i \le n$ to a pair $(\mathtt{M}_i, \ell_i)$
- ℓ is the (global) log

Notation

$\mathtt{M}_1 \boxed{\ell_1} \mid \dots \mid \mathtt{M}_n \boxed{\ell_n} \mid \ell$

Disclaimer

Seemingly, we've a contradiction: isn't the global log a centralisation point?

Well...no, it isn't: the global log is just a theoretical ploy!

- it abstracts away from low-level technical details for events' dispatching

Log shipping middlewares rely on timestamp mechanisms (Actyx uses Lamport's timestamps) and guarantee that events are in the same order in all the local logs

Swarms

A swarm (of size n) is a pair $(\mathtt{S}, \ell)$ where

- $\mathtt{S}$ maps each index $1 \leq i \leq n$ to a pair $(\mathtt{M}_i, \ell_i)$
- ℓ is the (global) log

Notation

$\mathtt{M}_1 \boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n \boxed{\ell_n} \mid \ell$

Disclaimer

Seemingly, we've a contradiction: isn't the global log a centralisation point?

Well...no, it isn't: the global log is just a theoretical ploy!

- it abstracts away from low-level technical details for events' dispatching
- it elegantly (IOHO) models asynchrony

Swarms

A swarm (of size n) is a pair $(\mathtt{S}, \ell)$ where

- $\mathtt{S}$ maps each index $1 \leq i \leq n$ to a pair $(\mathtt{M}_i, \ell_i)$
- ℓ is the (global) log

Notation

$\mathtt{M}_1\boxed{\ell_1} \mid \dots \mid \mathtt{M}_n\boxed{\ell_n} \mid \ell$

Disclaimer

Seemingly, we've a contradiction: isn't the global log a centralisation point?

Well...no, it isn't: the global log is just a theoretical ploy!

- it abstracts away from low-level technical details for events' dispatching
- it elegantly (IOHO) models asynchrony
- it is not used in our algorithms and tools

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

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

- $\ell_1 \subseteq \ell_2$ and $<_{\ell_1} \subseteq <_{\ell_2}$, that is and all events of ℓ_1 appear in the same order in ℓ_2
- $e \ <_{\ell_2} \ e'$ with $src(e) = src(e')$ 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

Coherence

A swarm $\mathtt{M}_1\boxed{\ell_1} \mid \ldots \mid \mathtt{M}_n\boxed{\ell_n} \mid \ell$ is coherent if $\ell = \bigcup_{1 \le i \le n} \ell_i$ 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 $<_{\ell_1} \subseteq <_{\ell_2}$, that is and all events of ℓ_1 appear in the same order in ℓ_2
- $e \;<_{\ell_2}\; e'$ with $src(e) = src(e')$ 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

Hereafter, we assume coherence

Merging logs

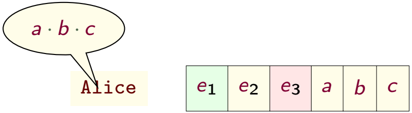
Log merging: $\ell_1 \bowtie \ell_2 = \{\ell \mid \ell \subseteq \ell_1 \cup \ell_2 \text{ and } \ell_1 \sqsubseteq \ell \text{ and } \ell_2 \sqsubseteq \ell\}$

Merging logs

Log merging: $\ell_1 \bowtie \ell_2 = \{\ell \mid \ell \subseteq \ell_1 \cup \ell_2 \text{ and } \ell_1 \sqsubseteq \ell \text{ and } \ell_2 \sqsubseteq \ell\}$

Exercise

Recall (slide 14) that



Suppose that `Alice` emits the events when the global log is $\ell = e_1 \cdot e_2 \cdot e_3 \cdot e$ and tell under which condition on e a system with global log ℓ and `Alice` as one of the machines is coherent

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

Semantics at work (I)

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

Semantics at work (I)

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

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

Semantics at work (I)

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

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

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

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

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

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

Exercise

Compute $(a \cdot b \cdot c) \bowtie (b \cdot d \cdot e)$

Semantics at work (II)

In the hypothesis of slide 26, we have that

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

is a possible transition in our semantics by [Local] with $\ell = b \cdot a \cdot d \cdot e \cdot c$.

Semantics at work (II)

In the hypothesis of slide 26, we have that

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

is a possible transition in our semantics by [Local] with $\ell = b \cdot a \cdot d \cdot e \cdot c$.

Exercise

Can we propagate just event e?

Semantics at work (II)

In the hypothesis of slide 26, we have that

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

is a possible transition in our semantics by [Local] with $\ell = b \cdot a \cdot d \cdot e \cdot c$.

Exercise

Can we propagate just event e?

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

Semantics at work (II)

In the hypothesis of slide 26, we have that

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

is a possible transition in our semantics by [Local] with $\ell = b \cdot a \cdot d \cdot e \cdot c$.

Exercise

Can we propagate just event e?

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

Let's propagate $d \cdot e$

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

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

Semantics at work (II)

In the hypothesis of slide 26, we have that

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

is a possible transition in our semantics by [Local] with $\ell = b \cdot a \cdot d \cdot e \cdot c$.

Exercise

Can we propagate just event e?

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

Let's propagate $d \cdot e$

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

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

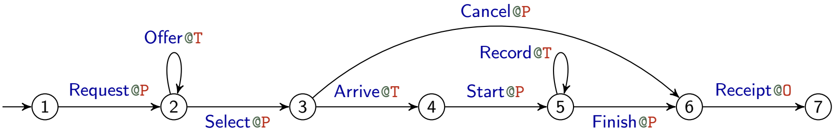
Excercise

In both cases b must be shipped too. Why?
And why is event a not shipped to $\mathtt{C}$ together with the events from $\mathtt{B}$?

– Behavioural types for swarms –

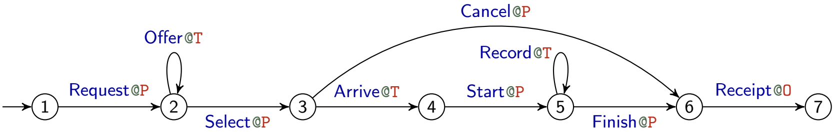
A taxi service

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



A taxi service

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

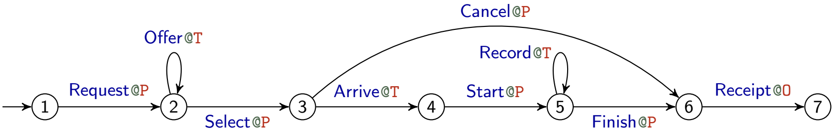


We assume

- one passenger and one office (for simplicity)

A taxi service

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

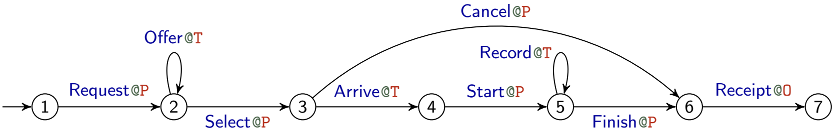


We assume

- one passenger and one office (for simplicity)
- but an arbitrary number of taxis

A taxi service

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



We assume

- one passenger and one office (for simplicity)
- but an arbitrary number of taxis
- a receipt is issued by the office O at the end of the ride (if any)

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"

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$_n$

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

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

Well-formedness

Asynchrony

M_1
Local viewpoint$_1$

M_i
Local viewpoint$_i$

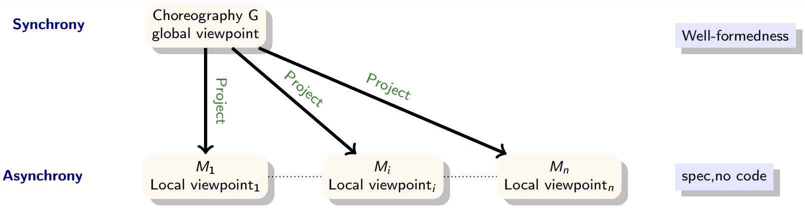
M_n
Local viewpoint$_n$

spec,no code

Choreographies

Quoting W3C:

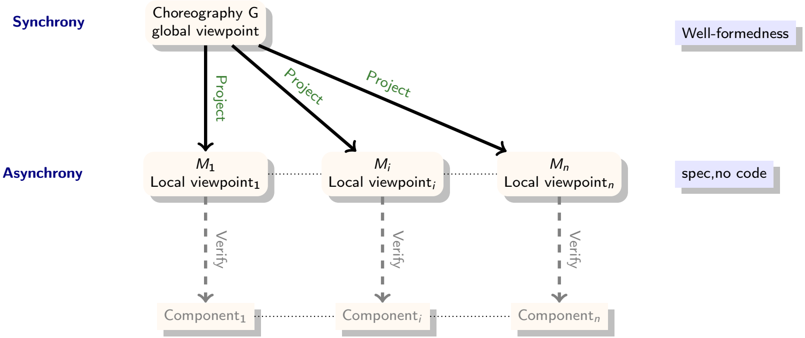
"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...]
Each party can then use the global definition to build and test solutions [...]
global specification is in turn realised by combination of the resulting local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



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: 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 29)

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

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

An example

A swarm protocol for the taxi scenario on slide 29:

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

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

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

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

An example

A swarm protocol for the taxi scenario on slide 29:

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

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

Note the log types in each prefixes

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

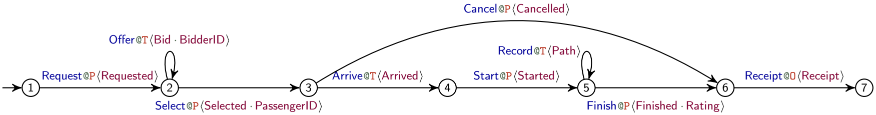
$$\begin{aligned} G_{\mathsf{ride}} &= \mathsf{Record}@\mathtt{T}\langle \mathsf{Path}\rangle \,.\, G_{\mathsf{ride}} \\ &\quad + \mathsf{Finish}@\mathtt{P}\langle \mathsf{Finished}\cdot\mathsf{Rating}\rangle \,.\, \mathsf{Receipt}@\mathtt{O}\langle \mathsf{Receipt}\rangle \,.\, 0 \end{aligned}$$

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:

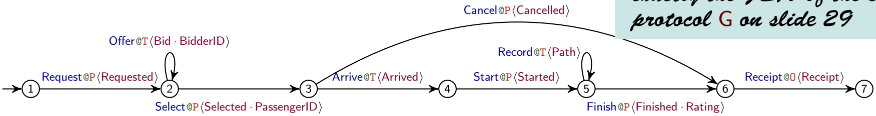
- the set of states consists G plus the states in G_i for each $i \in \{1 \ldots, n\}$
- G is the initial state
- for each $i \in I$, G has a transition to state G_i labelled with $\mathtt{c}_i@\mathtt{R}_i\langle \mathtt{l}_i \rangle$, written $\mathsf{G} \xrightarrow{\mathtt{c}_i \,/\, \mathtt{l}_i} \mathsf{G}_i$

An example



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

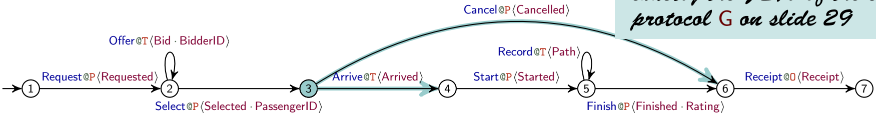
An example



Removing log types yields exactly the FSA of the swarm protocol G on slide 29

An example

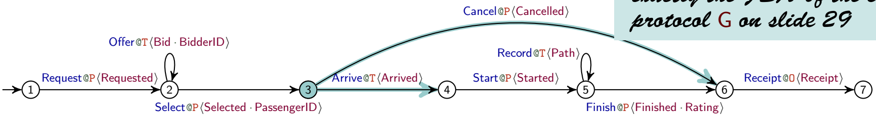
Removing log types yields exactly the FSA of the swarm protocol G on slide 29



There is a race in state 3!

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

An example



Removing log types yields exactly the FSA of the swarm protocol G on slide 29

There is a race in state 3!

- 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!

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

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

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

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

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!

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

where

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

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

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

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 use subscriptions : maps from roles to sets of event types

In pub-sub, processes subscribe to "topics"

Another attempt

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

In pub-sub, processes subscribe to "topics"

Given $\mathsf{G} = \left(\sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i\right)$ and a role $\mathtt{R}$ the projection of G on $\mathtt{R}$ 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 use subscriptions : maps from roles to sets of event types

In pub-sub, processes subscribe to "topics"

Given $\mathsf{G} = \left(\sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle \,.\, \mathsf{G}_i\right)$ and a role $\mathtt{R}$ the projection of G on $\mathtt{R}$ 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

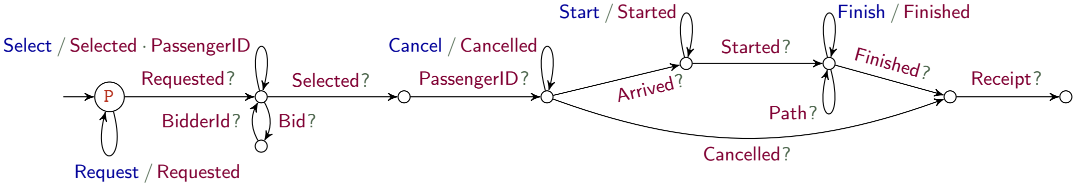
$$\kappa = \{\mathtt{c}_i \,/\, \mathtt{l}_i \mid \mathtt{R}_i = \mathtt{R} \text{ and } i \in I\}$$

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

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

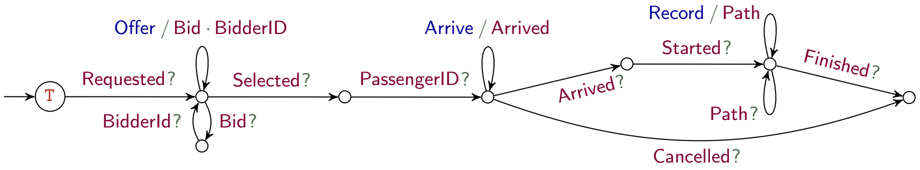
An example

A reasonable subscription for P is the total one since the passenger should be aware of all events: $\sigma(\mathtt{P})$ contains all event types



An example

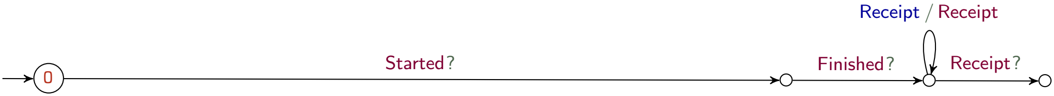
The taxi driver does not need to bother with the receipt: the subscription for $\sigma(\mathtt{T})$ consists of all messages but Receipt



An example

If we want the office to know only the details about the ride we set
$\sigma(0) = \{\text{Started}, \text{Finished}, \text{Receipt}\}$

If we want the office to know only the details about the ride we set
$\sigma(0) = \{\text{Started}, \text{Finished}, \text{Receipt}\}$



Exercise (hard)

Is this a good idea?

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` have made their selection 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: sufficient conditions for well-behaviour

Transitory deviations are tolerated provided that consistency is eventually recovered

Example

$\mathtt{T}$ may bid after $\mathtt{P}$ have made their selection if the selection event $\mathtt{T}$ has not yet been received. This inconsistency is temporary: when the selection event reaches $\mathtt{T}$ this inconsistency is recognised and resolved

Convention

Let's write $\mathtt{R} \in_\sigma \mathsf{G} = \sum_{i \in I} \mathtt{c}_i @ \mathtt{R}_i \langle \mathtt{l}_i \rangle . \mathsf{G}_i$ when there is $i \in I$ such that

$$\mathtt{R} = \mathtt{R}_i \quad \text{or} \quad \sigma(\mathtt{R}) \cap \mathtt{l}_i \neq \emptyset \quad \text{or} \quad \mathtt{R} \in_\sigma \mathsf{G}_i$$

and set $\mathsf{roles}(\mathsf{G}, \sigma) = \{\mathtt{R} \mid \mathtt{R} \in_\sigma \mathsf{G}\}$ and

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

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

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

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

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

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

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

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	if there is a unique subtree G' of G emitting $\mathtt{t}$ for each $\mathtt{t}$ starting a log emitted by a command in G

Some considerations

Further consequences:

- **Unspecified receptions** are just ignored according to the δ transition function of machines
- It is fine to violate **session fidelity**, provided that consistency is eventually attained
- Care is therefore necessary
  - for the definition of **correctness**
  - and for the **correct realisation** of swarm protocols

Some considerations

Further consequences:

- **Unspecified receptions** are just ignored according to the δ transition function of machines
- It is fine to violate **session fidelity**, provided that consistency is eventually attained
- Care is therefore necessary
  - for the definition of **correctness**
  - and for the **correct realisation** of swarm protocols

Of course we appeal to projections

On correctness

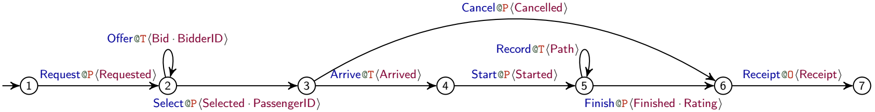
$(\mathtt{S}, \ell)$ faithfully implements G if it produces only logs possibly generated by G

On correctness

(S, ℓ) faithfully implements G if it produces only logs possibly generated by G

Exercise

Take the swarm $S = \texttt{P}\,[\,] \parallel \texttt{T}\,[\,] \parallel \texttt{O}\,[\,] \parallel \texttt{T}\,[\,]$ implementing



(i.e., the swarm protocol G on slide 34). Check that S generates the log

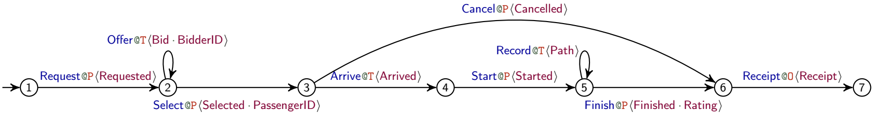
$$\ell_{\mathsf{auc}} = \mathit{requested} \cdot \mathit{bid} \cdot \mathit{bidderID} \cdot \mathit{selected} \cdot \mathit{bid} \cdot \mathit{bidderID} \cdot \mathit{passengerID}$$

On correctness

$(\mathtt{S}, \ell)$ faithfully implements G if it produces only logs possibly generated by G

Exercise

Take the swarm $\mathtt{S} = \mathtt{P}\ [\,] \mid \mathtt{T}\ [\,] \mid \mathtt{O}\ [\,] \mid \mathtt{T}\ [\,]$ implementing



(i.e., the swarm protocol G on slide 34). Check that S generates the log

$$\ell_{\mathsf{auc}} = \textit{requested} \cdot \textit{bid} \cdot \textit{bidderID} \cdot \textit{selected} \cdot \textit{bid} \cdot \textit{bidderID} \cdot \textit{passengerID}$$

Too strong a requirement!

Let's consider only "good enough" logs, i.e., those typeable with G's log types

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \vdash_\sigma \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{}{\mathsf{G}, \epsilon \vdash_\sigma e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \vdash_\sigma \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{\vdash \mathtt{e} : \mathtt{t}}{\mathsf{G}, \epsilon \vdash_\sigma \mathtt{e} \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \vdash_\sigma \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{\vdash \mathsf{e} : \mathtt{t} \in \sigma(\mathsf{roles}(\mathsf{G}, \sigma)) \qquad \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{t}\cdot\mathtt{l}'} \mathsf{G}'}{\mathsf{G}, \epsilon \vdash_\sigma \mathsf{e}\cdot\ell \triangleright \mathtt{t}\cdot\mathtt{l}}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \vdash_\sigma \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{\vdash e : \mathtt{t} \in \sigma(\mathsf{roles}(\mathsf{G}, \sigma)) \qquad \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{t}\cdot\mathtt{l}'} \mathsf{G}' \qquad \mathsf{G}', \mathsf{filter}(\mathtt{l}', \sigma(\mathsf{active}(\mathsf{G}'))) \vdash_\sigma \ell \triangleright \mathtt{l}}{\mathsf{G}, \epsilon \vdash_\sigma e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \vdash_\sigma \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{\vdash e : \mathtt{t} \in \sigma(\mathsf{roles}(\mathsf{G}, \sigma)) \qquad \mathsf{G} \xrightarrow{\mathtt{c} \,/\, \mathtt{t} \cdot \mathtt{l}'} \mathsf{G}' \qquad \mathsf{G}', \mathsf{filter}(\mathtt{l}', \sigma(\mathsf{active}(\mathsf{G}'))) \vdash_\sigma \ell \triangleright \mathtt{l}}{\mathsf{G}, \epsilon \vdash_\sigma e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}}$$

$$\frac{\vdash e : \mathtt{t} \qquad \mathsf{G}, \mathtt{l} \vdash_\sigma \ell \triangleright \mathtt{l}'}{\mathsf{G}, \mathtt{t} \cdot \mathtt{l} \vdash_\sigma e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}'}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \vdash_\sigma \ell \rhd \mathtt{l}$ is provable; where

$$\frac{\vdash \mathtt{e} : \mathtt{t} \in \sigma(\mathsf{roles}(\mathsf{G}, \sigma)) \qquad \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{t}\cdot\mathtt{l}'} \mathsf{G}' \qquad \mathsf{G}', \mathsf{filter}(\mathtt{l}', \sigma(\mathsf{active}(\mathsf{G}'))) \vdash_\sigma \ell \rhd \mathtt{l}}{\mathsf{G}, \epsilon \vdash_\sigma e \cdot \ell \rhd \mathtt{t} \cdot \mathtt{l}}$$

$$\frac{\vdash \mathtt{e} : \mathtt{t} \qquad \mathsf{G}, \mathtt{l} \vdash_\sigma \ell \rhd \mathtt{l}'}{\mathsf{G}, \mathtt{t}\cdot\mathtt{l} \vdash_\sigma e \cdot \ell \rhd \mathtt{t} \cdot \mathtt{l}'} \qquad \frac{}{\mathsf{G}, \mathtt{l} \vdash_\sigma \epsilon \rhd \epsilon}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \ \vdash_\sigma \ \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{\vdash e : \mathtt{t} \in \sigma(\mathsf{roles}(\mathsf{G}, \sigma)) \qquad \mathsf{G} \xrightarrow{\mathtt{c} \,/\, \mathtt{t} \cdot \mathtt{l}'} \mathsf{G}' \qquad \mathsf{G}', \mathsf{filter}(\mathtt{l}', \sigma(\mathsf{active}(\mathsf{G}'))) \ \vdash_\sigma \ \ell \triangleright \mathtt{l}}{\mathsf{G}, \epsilon \ \vdash_\sigma \ e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}}$$

$$\frac{\vdash e : \mathtt{t} \qquad \mathsf{G}, \mathtt{l} \ \vdash_\sigma \ \ell \triangleright \mathtt{l}'}{\mathsf{G}, \mathtt{t} \cdot \mathtt{l} \ \vdash_\sigma \ e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}'} \qquad \frac{}{\mathsf{G}, \mathtt{l} \ \vdash_\sigma \ \epsilon \triangleright \epsilon}$$

$$\frac{\mathsf{G}, \mathtt{l} \ \vdash_\sigma \ \ell \triangleright \mathtt{l}' \qquad \text{none of the other rules applies}}{\mathsf{G}, \mathtt{l} \ \vdash_\sigma \ e \cdot \ell \triangleright \mathtt{l}'}$$

Effective types

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

ℓ has effective type $\mathtt{l}$ wrt G and σ if $\mathsf{G}, \epsilon \ \vdash_\sigma \ \ell \triangleright \mathtt{l}$ is provable; where

$$\frac{\vdash e : \mathtt{t} \in \sigma(\mathsf{roles}(\mathsf{G}, \sigma)) \qquad \mathsf{G} \xrightarrow{\mathtt{c}\,/\,\mathtt{t}\cdot\mathtt{l}'} \mathsf{G}' \qquad \mathsf{G}', \mathsf{filter}(\mathtt{l}', \sigma(\mathsf{active}(\mathsf{G}'))) \ \vdash_\sigma \ \ell \triangleright \mathtt{l}}{\mathsf{G}, \epsilon \ \vdash_\sigma \ e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}}$$

$$\frac{\vdash e : \mathtt{t} \qquad \mathsf{G}, \mathtt{l} \ \vdash_\sigma \ \ell \triangleright \mathtt{l}'}{\mathsf{G}, \mathtt{t} \cdot \mathtt{l} \ \vdash_\sigma \ e \cdot \ell \triangleright \mathtt{t} \cdot \mathtt{l}'} \qquad \frac{}{\mathsf{G}, \mathtt{l} \ \vdash_\sigma \ \epsilon \triangleright \epsilon}$$

$$\frac{\mathsf{G}, \mathtt{l} \ \vdash_\sigma \ \ell \triangleright \mathtt{l}' \qquad \text{none of the other rules applies}}{\mathsf{G}, \mathtt{l} \ \vdash_\sigma \ e \cdot \ell \triangleright \mathtt{l}'}$$

Exercise

For the swarm protocol G on slide 34, find a condition on σ so that

$$\mathsf{G}, \epsilon \vdash_\sigma \ell_{\mathsf{auc}} \triangleright \mathsf{Requested} \,.\, \mathsf{Bid} \,.\, \mathsf{BidderID} \,.\, \mathsf{Selected} \,.\, \mathsf{PassengerID}$$

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 $(\mathtt{S}, \epsilon)$ of size n such that, for each $1 \leq i \leq n$, 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 $(\mathtt{S}, \epsilon)$ of size n such that, for each $1 \leq i \leq n$, there exists a role $\mathtt{R} \in \mathsf{roles}(\mathsf{G}, \sigma)$ 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* $\left(\delta(\mathsf{G} \downarrow^{\sigma}_{\mathtt{R}}, \ell)\right) \downarrow_{\mathtt{c} \,/\, \mathtt{1}}$ *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 $\ell_1' \cap \ell_2' = \emptyset$

On correct realisations

$$\begin{array}{ccc} & & (\mathtt{S}, \ell_1) \\ & \nearrow\!\!\!\nearrow & \\ (\mathtt{S}, \epsilon) & & \\ & \searrow\!\!\!\searrow & \\ & & (\mathtt{S}, \ell_2) \end{array}$$

consistent if there is ℓ s.t. $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}, \ell)$ $\begin{array}{c} \nearrow\!\!\!\nearrow (\mathtt{S}, \ell'_1) \\ \searrow\!\!\!\searrow (\mathtt{S}, \ell'_2) \end{array}$ with $\ell_1 = \ell \cdot \ell'_1$ and $\ell_2 = \ell \cdot \ell'_2$ and $\ell'_1 \cap \ell'_2 = \emptyset$

Notation

For $(\mathtt{G}, \epsilon) \xrightarrow{\mathtt{c_1} / \mathtt{l_1}} (\mathtt{G}, \ell_1) \xrightarrow{\mathtt{c_2} / \mathtt{l_2}} \cdots \xrightarrow{\mathtt{c}_n / \mathtt{l}_n} (\mathtt{G}, \overbrace{\ell_n \cdot \cdots \cdot \ell_2 \cdot \ell_1}^{=\ell})$ let $\ell^{(j)} = \ell_j \cdot \cdots \cdot \ell_1$

Admissible log

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

- ℓ' is $\mathtt{G} - \sigma$ equivalent to $\ell = \bigcup_{1 \le i \le k} \ell_i$, and
- $\ell_i^{(j)} \sqsubseteq \ell$ for all $1 \le i \le k$

Hereafter, $\mathtt{G}$ be a σ-WF protocol

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

Results

Lemma (Well-formedness generates any admissible)

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

Lemma (Admissibility is preserved when extending partial views)

Let ℓ_1 and $\ell_2 \subseteq \ell_1$ be admissible logs for G*. If* $(\mathsf{G}, \ell_2) \xrightarrow{\mathtt{c}\,/\,\mathtt{l}} (\mathsf{G}, \ell_2 \cdot \ell_3)$ *and* $\ell \in \ell_1 \bowtie (\ell_2 \cdot \ell_3)$ *then ℓ is admissible for* G

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

On complete realisations

Complete realisations

A $(\sigma, \mathtt{G})$-realisation $(\mathtt{S}, \epsilon)$ of size n is complete if for all $\mathtt{R} \in \mathsf{roles}(\mathtt{G}, \sigma)$ there exists $1 \leq i \leq n$ such that $\mathtt{S}(i) = \mathtt{G} \downarrow^{\sigma}_{\mathtt{R}}$ $\square$

Lemma (Projections reflect swarm protocols)

If $(\mathtt{G}, \epsilon) \Longrightarrow (\mathtt{G}, \ell)$ *then* $\delta(\mathtt{G} \downarrow^{\sigma}_{\mathtt{R}}, \ell) = \delta(\mathtt{G}, \ell) \downarrow^{\sigma}_{\mathtt{R}}$ *for all* $\mathtt{R} \in \mathsf{roles}(\mathtt{G}, \sigma)$

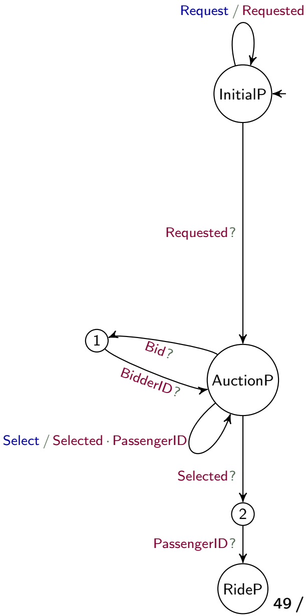
Theorem (Complete realisations reflect the protocol)

Let $(\mathtt{S}, \epsilon)$ *be a complete realisation of* $\mathtt{G}$*. If* $(\mathtt{G}, \epsilon) \Longrightarrow (\mathtt{G}, \ell)$ *then there is a swarm* $\mathtt{S}'$ *such that* $(\mathtt{S}, \epsilon) \Longrightarrow (\mathtt{S}', \ell)$

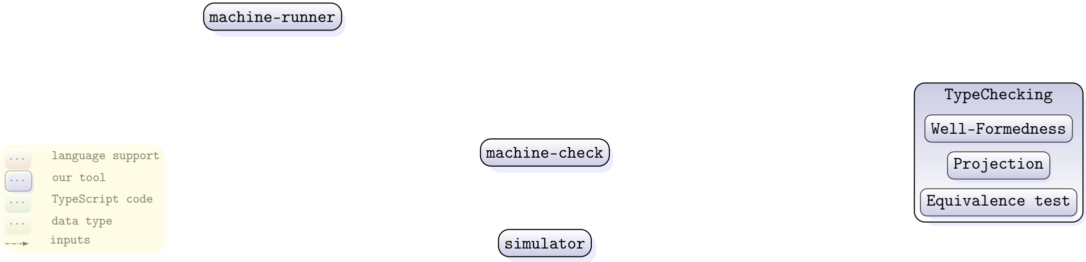
– Tooling –

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

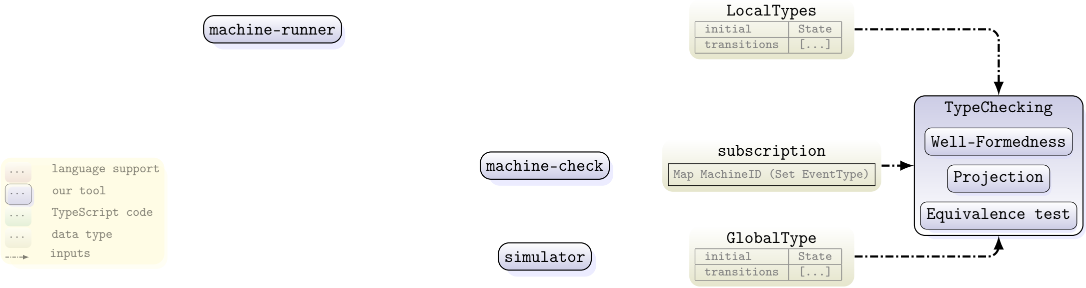


Architecture



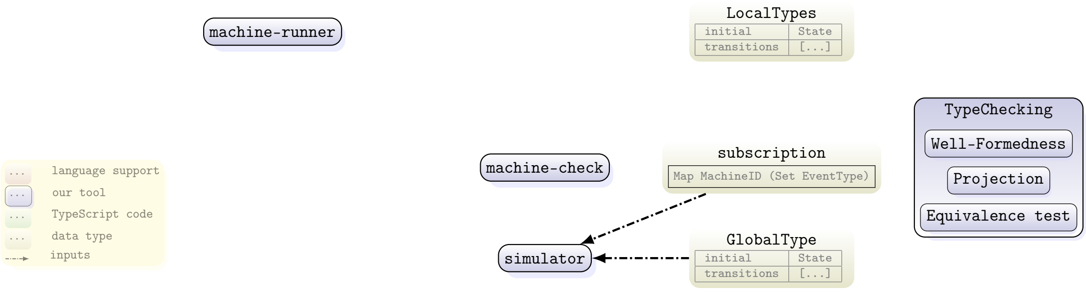
- TypeChecking implements the functionalities of our typing discipline
- simulator simulates the semantics of swarm realisations
- machine-check and machine-runner integrate our framework in the Actyx platform

Architecture



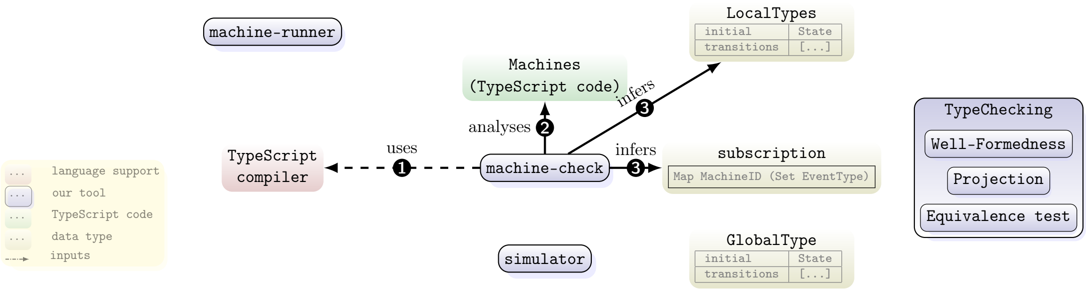
- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` and `machine-runner` integrate our framework in the Actyx platform

Architecture



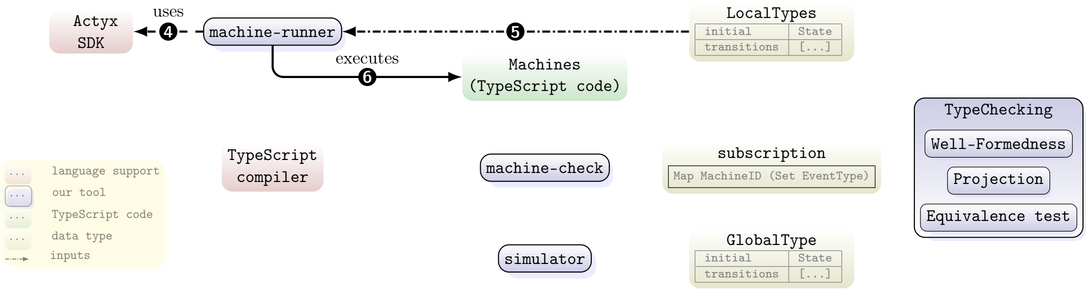
- TypeChecking implements the functionalities of our typing discipline
- simulator simulates the semantics of swarm realisations
- machine-check and machine-runner integrate our framework in the Actyx platform

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

– Epilogue –

Summary

An interesting paradigm grounded on principles for local-first software

Summary

An interesting paradigm grounded on principles for local-first software

We defined an operational semantics that captures the platform of Actyx AG

Summary

An interesting paradigm grounded on principles for local-first software

We defined an operational semantics that captures the platform of Actyx AG

We introduced behavioural types to specify and verify eventual consistency

Summary

An interesting paradigm grounded on principles for local-first software

We defined an operational semantics that captures the platform of Actyx AG

We introduced behavioural types to specify and verify eventual consistency

The key idea is to trade consistency for availability: temporary inconsistency are tolerated provided that they can be resolved at some point

To be continued....

There are a number of future directions to explore:

To be continued....

There are a number of future directions to explore:

Of course, identify weaker conditions for well-formedness

To be continued....

There are a number of future directions to explore:

Of course, identify weaker conditions for well-formedness

Subscriptions are hard to determine

To be continued....

There are a number of future directions to explore:

Of course, identify weaker conditions for well-formedness

Subscriptions are hard to determine

Compensations

To be continued....

There are a number of future directions to explore:

Of course, identify weaker conditions for well-formedness

Subscriptions are hard to determine

Compensations

Unreliable propagation

To be continued....

There are a number of future directions to explore:

Of course, identify weaker conditions for well-formedness

Subscriptions are hard to determine

Compensations

Unreliable propagation

Adversarial contexts

To be continued....

There are a number of future directions to explore:

Of course, identify weaker conditions for well-formedness

Subscriptions are hard to determine

Compensations

Unreliable propagation

Adversarial contexts

....

Thank you!