Automata for choreographies

Emilio Tuosto

Department of Computer Science

DI/NOVA LINCS

July 27, 2022
Take-home message

- Choreographies for communicating systems
- Two automata-based models
  - point-to-point communications
  - event-notification coordination
- An emerging pattern
  - fix a communication model
  - find suitable global and local specs
  - define well-formedness
  - get correct realisations by projection
– Act I –

Formal Choreographies, informally

(joint work with Roberto Guanciale)
Top-down model-driven development

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”
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”
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”
Top-down model-driven development

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”

well-formedness

specs, not code
A model of global specs

\[ G, \ G' ::= (o) \quad \text{empty} \]
\[ A \to B : m \quad \text{interaction} \]
\[ G; G' \quad \text{sequential} \]
\[ G \mid G' \quad \text{parallel} \]
\[ G + G' \quad \text{branch} \]
\[ G^* \quad \text{iteration} \]
A model of global specs

\[ G, G' ::= (o) \quad \text{empty} \]
\[ A \rightarrow B : m \quad \text{interaction} \]
\[ G; G' \quad \text{sequential} \]
\[ G | G' \quad \text{parallel} \]
\[ G + G' \quad \text{branch} \]
\[ G^* \quad \text{iteration} \]
Some examples

A $\rightarrow$ B: int  
B $\rightarrow$ C: string  
C $\rightarrow$ B: int

A $\rightarrow$ B: string  
B $\rightarrow$ C: int  
C $\rightarrow$ B: string
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
A model of local specs

Communicating Finite-State Machines [Brand&Zafiropulo 1983]

Global specs can be projected (i.e., compiled) on CFSMs
An obvious (fundamental) question

Given a global specification, is it realisable distributively?
An obvious (fundamental) question

**Given a global specification, is it realisable distributively?**

**Put simply...**

A global spec $G$ is **realizable** if there is a **deadlock-free** system of CFSMs whose traces “have some relation with” $G$.

---

$a$A system $S$ is *deadlock-free* if none of its reachable configurations $s$ is a deadlock, that is $s \not\rightarrow$ and either some buffers are not empty or some CFSMs have transitions from their state in $s$. 
Class test

Revisiting our examples

A → B: int
A → C: string
B → C: string
C → B: int

A → B: string
A → C: int
C → B: string

A → B: int
C → B: int
Class test

Revisiting our examples

A \rightarrow B: \text{int}
A \rightarrow C: \text{string}
B \rightarrow C: \text{string}
C \rightarrow B: \text{int}

A \rightarrow B: \text{int}
C \rightarrow B: \text{string}

A \rightarrow B: \text{string}
A \rightarrow B: \text{int}
C \rightarrow B: \text{int}
A (main) source of problems: Well-branchedness

**Distributed consensus**

A distributed choice $G_1 + G_2 + \cdots$ is well-branched if

- there is one active participant
- any non-active participant is passive
A (main) source of problems: Well-branchedness

**Distributed consensus**

A distributed choice $G_1 + G_2 + \cdots$ is well-branched if

- there is one active participant
- any non-active participant is passive

**Def.** A is active when it locally decides which branch to take in a choice

**Def.** B is passive when

- either B behaves uniformly in each branch
- or B “unambiguously understands” which branch A opted for from some inputs
A (main) source of problems: Well-branchedness

Distributed consensus

A distributed choice $G_1 + G_2 + \cdots$ is well-branched if

- there is one active participant
- any non-active participant is passive

Def. A is active when it locally decides which branch to take in a choice

Def. B is passive when

- either B behaves uniformly in each branch
- or B "unambiguously understands" which branch A opted for from some inputs

Well-branchedness

When the above holds true for each choice, the choreography is well-branched. This enables correctness-by-design.
– Act II –

Choreography Automata

(joint work with Franco Barbanera, Ivan Lanese)
The online-wallet protocol

1. Customer → Wallet: login
2. Customer → Wallet: pin
3. Wallet → Customer: retry
4. Wallet → Customer: loginOK
5. Wallet → Vendor: loginOK
6. Vendor → Customer: request
7. Customer → Wallet: authorise
8. Customer → Wallet: reject
9. Customer → Vendor: pay
10. Customer → Vendor: reject
11. Wallet → Customer: loginDenied
The online-wallet protocol

...some modelling problems

What about Vendor?
The online-wallet protocol ...some modelling problems

What about Vendor?

What about payloads?
Our global & local specs

**Choreography automata: Interaction, globally**

- Transition rules:
  - $C \xrightarrow{W} login$ from $q_0$ to $q_1$
  - $W \xrightarrow{C} loginOK$ from $q_4$ to $q_5$
  - $V \xrightarrow{C} request$ from $q_5$ to $q_6$
  - $C \xrightarrow{W} reject$ from $q_6$ to $q_7$
  - $C \xrightarrow{W} authorise$ from $q_7$ to $q_0$
  - $V \xrightarrow{W} loginOK$ from $q_4$ to $q_5$
  - $W \xrightarrow{C} loginDenied$ from $q_2$ to $q_3$
  - $C \xrightarrow{W} reject$ from $q_3$ to $q_4$
  - $C \xrightarrow{W} pay$ from $q_7$ to $q_6$
  - $W \xrightarrow{C} retry$ from $q_1$ to $q_2$
Our global & local specs

Intermediate automata: from interactions to communications
Our global & local specs

Intermediate automata: from interactions to communications

CFSMs locally: determinise the intermediate automaton
Theorem. Choreography automata are bisimilar to their projections

⇒ traces equivalence
Flexibility by example

Selective participation in OLW

\[
\begin{align*}
q_0 & \xrightarrow{\text{C} \rightarrow \text{W}: \text{login}} q_1 \\
q_1 & \xrightarrow{\text{C} \rightarrow \text{W}: \text{pin}} q_2 \\
q_2 & \xrightarrow{W \rightarrow C: \text{loginOK}} q_3 \\
q_3 & \xrightarrow{W \rightarrow C: \text{loginDenied}} q_4 \\
q_4 & \xrightarrow{W \rightarrow V: \text{loginOK}} q_5 \\
q_5 & \xrightarrow{V \rightarrow C: \text{request}} q_6 \\
q_6 & \xrightarrow{C \rightarrow W: \text{reject}} q_7 \\
q_7 & \xrightarrow{C \rightarrow V: \text{pay}} q_0
\end{align*}
\]
Selective participation in OLW

- At $q_2$ Wallet and Customer aware from the very beginning
at \( q_2 \) Wallet and Customer aware from the very beginning

- Vendor involved on one branch only, but that’s fine: Wallet is aware
Selective participation in OLW

- at $q_2$ Wallet and Customer aware from the very beginning
  - Vendor involved on one branch only, but that’s fine: Wallet is aware
- at $q_6$ Wallet and Customer aware from the very beginning
Selectivity participation in OLW

- at $q_2$ Wallet and Customer aware from the very beginning
  - Vendor involved on one branch only, but that’s fine: Wallet is aware
- at $q_6$ Wallet and Customer aware from the very beginning
  - Vendor eventually informed by Customer on each branch
Correctness by construction

**Theorem.** Projections of well-formed choreography automata are deadlock-free

**Theorem.** Projections of well-formed choreography automata are lock-free
DbC vs. choreography automata

Asserting (an excerpt of) OLW

Consistency: history senesitiveness: in $q \lambda \rightarrow A q'$, $A$ predicates on known variables

Temporal satisfiability: the conjunction of the predicates on a path is satisfiable

Well-formedness of the underlying choreography automaton

$$r \cdot \text{try} \rightarrow 0 \quad 0 \leq \text{try} \leq 3$$

$$r \cdot \text{try} \rightarrow \text{try} + 1 \quad 0 \leq \text{try} \leq 3$$

$W \rightarrow C: \text{login}(\text{account int})$

$W \rightarrow C: \text{pin}(\text{pin int})$

$W \rightarrow C: \text{loginOk}()$

$W \rightarrow C: \text{loginDenied}(\text{msg string})$

$W \rightarrow C: \text{retry}(\text{msg string})$

$W \rightarrow C: \text{loginOk}()$

$0 \leq \text{try} < 3 \land \text{msg} = "fail"

$0 \leq \text{try} \leq 3$

$\text{try} \geq 3 \land \text{msg} = "5 min."$

$0 \leq \text{try} \leq 3$
DbC vs. choreography automata

Asserting (an excerpt of) OLW

Consistency

- **history sensitivity**: in $q \xrightarrow{\lambda} q'$, $A$ predicates on *known* variables
- **temporal satisfiability**: the conjunction of the predicates on a path is satisfiable
- **well-formedness** of the underlying choreography automaton
Theorems

Projections are a bit more complicated than for choreography automata

On consistent asserted choreography automata

**Theorem.** Asserted choreography automata are weakly bisimilar to their projections

\[ \Rightarrow \text{trace equivalence} \]

**Theorem.** Projections of WF choreography automata are deadlock-free
Theorems

Projections are a bit more complicated than for choreography automata

On consistent asserted choreography automata

**Theorem.** Asserted choreography automata are weakly bisimilar to their projections

\[ \Rightarrow \text{ trace equivalence} \]

**Theorem.** Projections of WF choreography automata are deadlock-free

And more...cf. [ECOOP 2022]

A tool chain for

- validating finitary Scribble protocols via choreography automata
- TypeScript web programming via API generation
– Act III –

Local-first!

(joint work with Daniela Marottoli, Hernán Melgratti, Roland Kuhn)
A completely different setting

Desiderata

- different features
  - arbitrary (and variable) number of instances
  - local-first principle!
    - As rock climbers say: “Don’t Be Afraid To Fail. Be Afraid Not To Try.”
- pub-sub (instead of point-to-point)
- different properties
  - progress despite unavailability
    - inconsistent views
  - eventual-consistency instead of “old” properties (eg. session fidelity)
Swarm protocols and machines by example

Global

1. request@P
2. select@P
3. arrive@T
4. start@P
5. finish@P
6. receipt@O
7. record@T

Request / requested
Bid / bid
Select / selected
Start / started
Finish / finished
Receipt / receipt

Cancel / cancelled

Local (projected)

M1 requested?
M2 bid?
M3 selected?
M4 arrived?
M5 started?
M6 finished?

Cancel / cancelled

(log types omitted for readability)
Swarm protocols and machines by example

Global

1. request@P
2. select@P
3. arrive@T
4. start@P
5. finish@P
6. receipt@O
7. (log types omitted for readability)

Local (projected)

M1: requested?
M2: bid?
M3: selected?
M4: arrived?
M5: started?
M6: finished?
M7: receipt?

local log: $r_1 \cdot r_2 \cdot b$
Swarm protocols and machines by example

Global

![Diagram of global protocol with nodes 1 to 7 and actions such as request, select, arrive, start, finish, receipt, and cancel.]

(log types omitted for readability)

Local (projected)

![Diagram of local protocol with machines M1 to M4 and actions such as request, select, start, and cancel.]

(local log: \( r_2 \cdot b \))
Swarm protocols and machines by example

Global

1. request@P
2. select@P
3. arrive@T
4. start@P
5. finish@P
6. receipt@O
7. request@O

(log types omitted for readability)

Local (projected)

- Requested (M1)
- Selected (M2)
- Started (M3)
- Finished (M4)

Local log: \cdot \cdot \cdot \cdot \cdot b
Swarm protocols and machines by example

Global

1. request@P → bid@T
2. select@P → arrive@T
3. record@T
4. finish@P
5. cancel@P
6. receipt@O
7. (log types omitted for readability)

Local (projected)

- request / requested
- select / selected
- start / started
- cancelled / cancelled
- receipt?

local log: · · and now select is enabled
Semantics, intuitively

- Types “produce/consume” events
  - swarm protocols: how/when roles produce events
  - machines: how/when instances consume events “skipping” the ones irrelevant to them

- Deterministic types only
  - swarm protocols: log types of branches have no common non-trivial prefixes and command/role pairs are pairwise distinct
  - machines: event types of branches are pairwise distinct

- Non-deterministic events’ propagation
Swarms

Machines, local logs, and global log (...a mirage)

Events are univocally associated to the machines generating them.

Def. swarm = global log + map from unique identities to pairs machines/local logs

\[ (S, l) = (M_1, l_1) | \ldots | (M_n, l_n) | l \]

such that \( l_i \sqsubseteq l \) where, \( l_i \sqsubseteq l \iff l_i = \ldots \ldots = l \)

i.e., there is an order-preserving and downward-total morphism from \( l_i \) into \( l \) on events of a same machine.
Swarms

Machines, local logs, and global log (...a mirage)

Events are univocally associated to the machines generating them.

**Def.** swarm = global log + map from unique identities to pairs machines/local logs

\[(S, l) = (M_1, l_1) \mid \ldots \mid (M_n, l_n) \mid l\]

such that \(l_i \sqsubseteq l\) where, \(l_i \sqsubseteq l \iff l_i = \ldots = e_{i,1}\) \(\ldots = e_{i,n}\) \(\ldots = e_m\)

i.e., there is an order-preserving and downward-total morphism from \(l_i\) into \(l\) on events of a same machine.
Swarms

Machines, local logs, and global log (...a mirage)

Events are univocally associated to the machines generating them.

**Def.** swarm = global log + map from unique identities to pairs machines/local logs

\[
(S, l) = (M_1, l_1) \mid \ldots \mid (M_n, l_n) \mid l
\]

such that \( l_i \sqsubseteq l \) where, \( l_i \sqsubseteq l \iff l_i = \ldots = l \)

i.e., there is an order-preserving and downward-total morphism from \( l_i \) into \( l \) on events of a same machine.
Swarms

Machines, local logs, and global log (...a mirage)

Events are univocally associated to the machines generating them.

**Def.** swarm = global log + map from unique identities to pairs machines/local logs

\[(S, l) = (M_1, l_1) | \ldots | (M_n, l_n) | l\]

such that \(l_i \sqsubseteq l\) where, \(l_i \sqsubseteq l \iff l_i = \ldots = l\)

i.e., there is an order-preserving and downward-total morphism from \(l_i\) into \(l\) on events of a same machine.
Swarms’ semantics...intuitively

- **Events’ generation**
  The local log of a machine is extended with the fresh events generated by (the execution of a command on) the machine

- **Events’ propagation**
  Emitted events propagate asynchronously & non-deterministically
Swarms’ semantics: formally

[LOCAL]

\[
S : i \mapsto (M, l) \quad (M, l) \xrightarrow{c/1} (M, l') \quad l'' \in l \otimes \hat{l}
\]

\[
(S, \hat{l}) \xrightarrow{c/1} (S[i \mapsto (M, l')], l'')
\]

where \( l_1 \otimes l_2 = \{ l \mid l \subseteq l_1 \cup l_2 \land l_1 \subseteq l \land l_2 \subseteq l \} \)

[PROP]

\[
S : i \mapsto (M, l) \quad l \subseteq l' \subseteq \hat{l} \quad l \subset l'
\]

\[
(S, \hat{l}) \xrightarrow{\tau} (S[i \mapsto (M, l')], \hat{l})
\]
Properties of our semantics

Coherence
A swarm \((M_1, l_1) | \ldots | (M_n, l_n) | l\) is coherent if

\[
\text{for all } i, \ l_i \sqsubseteq l \quad \text{and} \quad l = \bigcup_{i \in n} l_i
\]

Coherence preservation

[LOCAL] & [PROP] preserve coherence

Eventual Consistency
If

\[
S = (M_1, l_1) | \ldots | (M_n, l_n) | l \text{ is coherent}
\]

then

\[
S \xrightarrow{\tau}^* (M_1, l) | \ldots | (M_n, l) | l
\]
It is hard to get it right (even **without** multi-instances or choices!)

**A trivial protocol**

Take the swarm protocol

Are

`request@P(requested)`  
`bid@T(bid)`

and

`request / requested`  
`bid / bid`  
`requested?`

a realisation?
It is hard to get it right (even without multi-instances or choices!)

A trivial protocol

Take the swarm protocol

Are request / requested and bid / bid a (correct) realisation?

What does that actually mean?
Realisation

It is hard to get it right (even **without** multi-instances or choices!)

A trivial protocol

Take the swarm protocol

Are

\[ \text{request@P(requested)} \quad \text{bid@T(bid)} \]

and

\[ \text{request / requested} \quad \text{bid / bid} \]

\[ \overset{?}{\text{requested}} \]

a **(correct)** realisation? What does that actually mean?
Not so simple

A swarm correctly realises a swarm protocol if it generates only logs that the protocol can generate.

That’s impossible due to events’ skipping at local level but not at the global one.
Not so simple

A swarm correctly realises a swarm protocol if it generates only logs that the protocol can generate.

That’s impossible due to events’ skipping at local level but not at the global one.

A weaker condition

A swarm correctly realises a swarm protocol if it generates only logs that are admissible with some that the protocol can generate.

A log is admissible for a swarm protocol when its restriction to the events processed by the active machines is equivalent to a log of the protocol.
Realisation by projection

Well-formedness of swarm protocols

Each log type $l$ of a branch should be

- causal consistent
  - each selector in (the continuation of) $l$ reacts to $l$
  - each role involved in the continuation of $l$ cannot react to more events on $l$ than selectors on the branch

- determined
  - each role in the continuation of $l$ reacts to $l[0]$

- confusion-free
  - an event type cannot occur in more than one branch
Summing up

Automata models for choreography

Advantages

- increased flexibility
- good basis for (enhanced) tool support
- good also for practitioners

Plans

- weakening well-formedness conditions
- studying more complex communication models (eg non-atomic propagation of events)
Thank you!