Open Problems in Choreographic Development of Message-Passing Applications

Emilio Tuosto @ GSSI

ASYDE 2020 September 15, 2020

Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233.
This talk in 1 slide
Take-home message

This talk in 1 slide

**Choreographic development of distributed (message-passing) systems**

- exploits global & local specifications
- supports correctness-by-construction
- facilitates SDLC

Nonetheless, choreographies lack support for modularity/compositionality to be complemented by testing generalisations to (more abstract) coordination paradigm

Goal
Generate interest and/or criticisms & possibly collaborations
### Take-home message

This talk in 1 slide

<table>
<thead>
<tr>
<th>Choreographic development of distributed (message-passing) systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>- exploits global &amp; local specifications</td>
</tr>
<tr>
<td>- supports correctness-by-construction</td>
</tr>
<tr>
<td>- facilitates SDLC</td>
</tr>
</tbody>
</table>

### Nonetheless, choreographies

| - lack support for modularity/compositionality                |
| - to be complemented by testing                               |
| - generalisations to (more abstract) coordination paradigm    |
| - ...                                                        |
Take-home message

This talk in 1 slide

**Choreographic development of distributed (message-passing) systems**
- exploits global & local specifications
- supports **correctness-by-construction**
- facilitates SDLC

**Nonetheless, choreographies**
- lack support for modularity/compositionality
- to be complemented by testing
- generalisations to (more abstract) coordination paradigm
- ...

**Goal**
Generate interest and/or criticisms & possibly collaborations
— Prologue —

[ Choreographies, informally ]
What do I mean by “choreography”? 

Choreography = Global spec + Local spec
Model-driven development...by nature

Choreography = Global spec + Local spec

Quoting W3C:
"Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...], observable behaviour of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]."
Model-driven development...by nature

Quoting W3C:

“Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...] observable behaviour of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...].”
Quoting W3C:

“Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...] observable behaviour of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]”

specs, not code
Model-driven development...by nature

Quoting W3C:

“Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...] observable behaviour of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]”

specs, not code
Model-driven development...by nature

Quoting W3C:

"Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...] observable behaviour of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]"

specs, not code
Some advantages

A (possibly) useful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]
Some advantages

A (possibly) useful equation

\[
\text{Distribution} = \text{Local computation} + \text{Communication}
\]

- Separation of concerns
Some advantages

A (possibly) useful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]

- Separation of concerns
- Global specs support program comprehension
Some advantages

A (possibly) useful equation

\[
\text{Distribution} = \text{Local computation} + \text{Communication}
\]

- Separation of concerns
- Global specs support program comprehension
- “Distributed / DevOP-ish” development
  - Projections yield specs of local components
  - Developers can “test” each component against the local spec
  - \text{if cond}(\text{global artefact}) \text{ then behave}(\text{projection}(\text{global artefact}))
Some advantages

A (possibly) useful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]

- Separation of concerns
- Global specs support program comprehension
- “Distributed / DevOP-ish” development
  - Projections yield specs of local components
  - Developers can “test” each component against the local spec
    - if \( \text{cond}(\text{global artefact}) \) then \( \text{behave}(\text{projection}(\text{global artefact})) \)
- No centralisation / full distribution / Scalability
- ...
Some drawbacks

A (possibly) painful equation

\[
\text{Distribution} = \text{Local computation} + \text{Communication}
\]
Some drawbacks

A (possibly) painful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]

- The “right” global spec could be difficult to be found
Some drawbacks 😞

A (possibly) painful equation

**Distribution = Local computation + Communication**

- The “right” global spec could be difficult to be found
- Message-passing: “unusual” paradigm & asynchrony $\implies$ complexity
Some drawbacks

A (possibly) painful equation

\[\text{Distribution} = \text{Local computation} + \text{Communication}\]

- The “right” global spec could be difficult to be found
- Message-passing: “unusual” paradigm & asynchrony $\Rightarrow$ complexity
- “coordination = communication” / distributed consensus / dependencies
Some drawbacks

A (possibly) painful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]

- The “right” global spec could be difficult to be found
- Message-passing: “unusual” paradigm & asynchrony \( \implies \) complexity
- “coordination = communication” / distributed consensus / dependencies
  - local decisions (sometimes) need to be propagated
  - global state “scattered” across components
  - hence, components (must!) have a partial view of the global state
Some drawbacks

A (possibly) painful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]

- The “right” global spec could be difficult to be found
- Message-passing: “unusual” paradigm & asynchrony \( \implies \) complexity
- “coordination = communication” / distributed consensus / dependencies
  - local decisions (sometimes) need to be propagated
  - global state “scattered” across components
  - hence, components (must!) have a partial view of the global state
- Expose behaviour
Some drawbacks

A (possibly) painful equation

\[ \text{Distribution} = \text{Local computation} + \text{Communication} \]

- The “right” global spec could be difficult to be found
- Message-passing: “unusual” paradigm & asynchrony \( \implies \) complexity
- “coordination = communication” / distributed consensus / dependencies
  - local decisions (sometimes) need to be propagated
  - global state “scattered” across components
  - hence, components (must!) have a partial view of the global state
- Expose behaviour
- Code reuse maybe problematic
Some drawbacks

A (possibly) painful equation

\[
\text{Distribution} = \text{Local computation} + \text{Communication}
\]

- The “right” global spec could be difficult to be found
- Message-passing: “unusual” paradigm & asynchrony \( \rightarrow \) complexity
- “coordination = communication” / distributed consensus / dependencies
  - local decisions (sometimes) need to be propagated
  - global state “scattered” across components
  - hence, components (must!) have a partial view of the global state
- Expose behaviour
- Code reuse maybe problematic
- ...

(wait for the last part 😊)
– Act I –

[ A bird-eye view of choreographic design ]
There’re many\(^1\) (“formal”) models…

Reminder
I’m not advertising: my goal is to generate interest, criticisms & possibly collaborations

\(^1\)No systematic comparative study yet
Choosing a model of global specs

\[
G ::= (o) \quad \text{empty}
\]

\[
A \rightarrow B : m \quad \text{interaction}
\]

\[
G ; G \quad \text{sequential}
\]

\[
G \mid G \quad \text{parallel}
\]

\[
\text{sel} \{ G + \cdots + G \} \quad \text{branch}
\]
Choosing a model of global specs

\[
G ::= (o) \quad \text{empty}
\]

\[
A \rightarrow B : m
\quad \text{interaction}
\]

\[
G ; G \quad \text{sequential}
\]

\[
G \mid G \quad \text{parallel}
\]

\[
sel \{ G + \cdots + G \}
\quad \text{branch}
\]
Some examples

\[ A \xrightarrow{\cdot} B: x; B \xrightarrow{\cdot} C: z \]
\[ A \xrightarrow{\cdot} C: y; C \xrightarrow{\cdot} B: w \]
Setting-up a communication model

We’re going to review some results about a specific communication model
- channel-based
- asynchronous (most often)
- point-to-point
Setting-up a communication model

We’re going to review some results about a **specific** communication model
- channel-based
- asynchronous (most often)
- point-to-point

**Communicating Finite-State Machines**

Global specs can be projected (i.e., compiled) on CFSMs
Setting-up a communication model

We’re going to review some results about a specific communication model:
- channel-based
- asynchronous (most often)
- point-to-point

Communicating Finite-State Machines

Global specs can be projected (i.e., compiled) on CFSMs
Setting-up a communication model

We're going to review some results about a specific communication model

- channel-based
- asynchronous (most often)
- point-to-point

Communicating Finite-State Machines

Global specs can be projected (i.e., compiled) on CFSMs
Setting-up a communication model

We’re going to review some results about a specific communication model
- channel-based
- asynchronous (most often)
- point-to-point

Communicating Finite-State Machines

Global specs can be projected (i.e., compiled) on CFSMs
Setting-up a communication model

We’re going to review some results about a specific communication model

- channel-based
- asynchronous (most often)
- point-to-point

Communicating Finite-State Machines

Global specs can be projected (i.e., compiled) on CFSMs
Setting-up a communication model

We’re going to review some results about a specific communication model
- channel-based
- asynchronous (most often)
- point-to-point

Communicating Finite-State Machines

Global specs can be projected (i.e., compiled) on CFSMs
Setting-up a communication model

We’re going to review some results about a specific communication model:
- channel-based
- asynchronous (most often)
- point-to-point

Communicating Finite-State Machines

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

Given a global specification, is it realisable distributively?
Examples

Not all specs can be “faithfully” executed distributively...
Examples

Not all specs can be “faithfully” executed distributively...

Trivial non-realisability

A B?m → B C?n

[Alur et al. 2003]
Examples

Not all specs can be “faithfully” executed distributively...

**Trivial non-realisability**

\[ \text{A B?m} \rightarrow \text{B C?n} \]

Communicating systems “start” with outputs!
Examples

Not all specs can be “faithfully” executed distributively...

**Trivial non-realisability**

A B?m → B C?n

Communicating systems “start” with outputs!

**Non-trivial non-realisability**

[Alur et al. 2003]
Realisability

Put simply...

A global spec $G$ is realizable if there is a deadlock-free\(^a\) communicating system whose language “has 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 $s$.
Realisability

Put simply...

A global spec $G$ is realizable if there is a deadlock-free\(^a\) communicating system whose language “has 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 $s$.

A recipe for theorems

1. Define projections and the semantics of global and local specs
2. Show the global spec $G$ is well-formed (for some def of well-formedness)
3. Show that $G$ and its projections have a “suitable” relation

\[ \text{[GT19]} \]

$G$ whole-spectrum iff $G$ cannot drop mandatory beh.; then projections cover $G$\[ \text{[BMT20]} \]

$G$ well-asserted iff $G$ temporal satisfiable & history sensitive; then projections simulates $G$\[ \text{[BHTY10]} \]
Realisability

Put simply...

A global spec $G$ is **realizable** if there is a deadlock-free$^a$ communicating system whose language “has some relation with” $G$.

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

A recipe for theorems

1. Define projections and the semantics of global and local specs
2. Show the global spec $G$ is **well-formed** (for some def of well-formedness)
3. Show that $G$ and its projections have a “suitable” relation

Some instances

- $G$ well-formed iff $\mathcal{L}(G)$ closed; then usual projections yield a language included in $\mathcal{L}(G)$ [GT19]
- $G$ whole-spectrum iff $G$ cannot drop mandatory beh.; then projections cover $G$ [BMT20]
- $G$ well-asserted iff $G$ temporal satisfiable & history sensitive; then projections simulates $G$ [BHTY10]
Distributed consensus

In a distributed choice $G_1 + G_2 + \cdots$

- there should be one active participant
- any non-active participant should be passive decides which branch to take in a choice
### Distributed consensus

In a distributed choice $G_1 + G_2 + \cdots$

- There should be **one active** participant.
- Any non-active participant should be **passive** decides which branch to take in a choice.

**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 through the information received on each branch.

---

A (main) source of problems: Well-formedness (intuitively)
A (main) source of problems: Well-formedness (intuitively)

**Distributed consensus**

In a distributed choice $G_1 + G_2 + \cdots$

- there should be **one active** participant
- any non-active participant should be **passive** decides which branch to take in a choice

**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 through the information received on each branch

**Well-branchedness**

When the above holds true for each choice, the choreography is **well-branched**. This enables **correctness-by-design**.
Figure out the graphical structure of the following terms and for each of them say which one is well-branched

- $G_1 = A \rightarrow B: \text{int} + A \rightarrow B: \text{str}$
- $G_2 = A \rightarrow B: \text{int} + A \rightarrow C: \text{str}$
- $G_3 = \left( \begin{array}{c}
A \rightarrow C: \text{int} ; A \rightarrow B: \text{bool} \\
+ \\
A \rightarrow C: \text{str} ; A \rightarrow C: \text{bool} ; A \rightarrow B: \text{bool}
\end{array} \right) ; B \rightarrow D: \text{str}$

\[ \text{; ; has precedence over ; + ;} \]
Figure out the graphical structure\(^2\) of the following terms and for each of them say which one is well-branched

\[\text{G}_1 = A \rightarrow B : \text{int} + A \rightarrow B : \text{str}\]

\[\text{G}_2 = A \rightarrow B : \text{int} + A \rightarrow C : \text{str}\]

\[\text{G}_3 = \left( \begin{array}{c} A \rightarrow C : \text{int}; A \rightarrow B : \text{bool} \\ + \\ A \rightarrow C : \text{str}; A \rightarrow C : \text{bool}; A \rightarrow B : \text{bool} \end{array} \right); B \rightarrow D : \text{str}\]

\(^2\); \text{.} \text{ has precedence over } + \text{.}
Figure out the graphical structure\(^2\) of the following terms and for each of them say which one is well-branched

\[ G_1 = A \rightarrow B: \text{int} + A \rightarrow B: \text{str} \]
\[ G_2 = A \rightarrow B: \text{int} + A \rightarrow C: \text{str} \]
\[ G_3 = \left( \begin{array}{c}
A \rightarrow C: \text{int}; A \rightarrow B: \text{bool} \\
+ \\
A \rightarrow C: \text{str}; A \rightarrow C: \text{bool}; A \rightarrow B: \text{bool}
\end{array} \right); B \rightarrow D: \text{str} \]

\(^2\); \ has precedence over \ + \
– Act II –

[ Some open problems ]
– Scene 1 –

[ Beyond holistic global specs ]
Modular development

Problem I(a): compositionality

How to compose global specs so to preserve “good” properties?

- Projections support modularity of local specs
- Global specs are typically holistic
  - Compositionality of global spec is harder
  - It is not clear what is to be used as an *interface*
An attempt: Preserving (dead)lock-freedom [BDLT20,BLT20]

Idea: compatible interface + gateways

A simple example

Composing global specs

\[
\begin{align*}
\text{(A → B): req} & \quad \text{(C → D): req} \\
\text{(A → B): adv} & \quad \text{(C → D): adv} \\
\text{(B → Sorry): a} & \quad \text{(D → C): adv}
\end{align*}
\]

where

- \text{Band C are the interfaces}
- \text{proj(B) compatible with dual(proj(C)) ... once channels are forgotten}
- \text{Band C are replaced by their gateways B’ and C’}

\[^a\text{We are also looking at similar results for local specs}\]
Some initial results

- (Dead)lock-freedom, compositionally
  - Typable systems are lock-free &
  - \( \Rightarrow \) preserves lock-freedom
  - \( \Rightarrow \) the composition of typable systems is lock-free

- Gateways may be “merged” (semi-direct composition) or even removed (direct composition)!

- Oddly, the synchronous case for local specs is more involved than the asynchronous one
E. W. Dijkstra: Notes on Structured Programming

“The basic pattern of my approach will be to compose the program in minute steps, deciding each time as little as possible. As the problem analysis proceeds, so does the further refinement of my program”

Problem I(b): refinement

How to support step-wise refinement of choreographies?
Adding **refinable** (and multiple) interaction:

\[ G ::= \cdots | A^{m_1 \ldots m_n} \rightarrow B_1 \ldots B_n \quad \text{where } n > 0 \]

to be replaced by a well-formed *ground* \( \hat{G} \) such that

- **unique initiator**: \( A \) executes any first communication in \( \hat{G} \)
- **eventual reception**: for all \( 1 \leq i \leq n \), the last action of \( B_i \) in any branch of \( \hat{G} \) is an input of message \( m_i \)
Examples

Which are legal refinements of the following?

\[ C \xrightarrow{md} S \quad \text{and} \quad C \xrightarrow{req} S \quad \text{and} \quad S \xrightarrow{\text{done}} C \]

Sound refinements may be “wrong”:

- \( C \xrightarrow{\text{S}} \text{md} + C \xrightarrow{\text{S}} \text{req} \); (\( S \xrightarrow{\text{C}} \text{stats} \); \( S \xrightarrow{\text{C}} \text{done} \)
Examples

Which are legal refinements of the following?

\[ C \xrightarrow{\text{md}} S + C \xrightarrow{\text{req}} S; S \xrightarrow{\text{done}} C \]

Sound refinements may be “wrong”:

- \( C \xrightarrow{} S: \text{md} + C \xrightarrow{} S: \text{req}; (S \xrightarrow{} C: \text{stats}; S \xrightarrow{} C: \text{done}) \)

- \( (C \xrightarrow{} B: \text{md}; B \xrightarrow{} S: \text{md}) + C \xrightarrow{} S: \text{req}; (S \xrightarrow{} C: \text{stats}; S \xrightarrow{} C: \text{done}) \)
Examples

Which are legal refinements of the following?

\[
C \xrightarrow{\text{md}} S + C \xrightarrow{\text{req}} S; S \xrightarrow{\text{done}} C
\]

Sound refinements may be “wrong”:

- \( C \xrightarrow{\text{md}} S \) + \( C \xrightarrow{\text{req}} S \); \( S \xrightarrow{\text{stats}} C \); \( S \xrightarrow{\text{done}} C \)

- \( (C \xrightarrow{\text{md}} B; B \xrightarrow{\text{md}} S) \) + \( C \xrightarrow{\text{req}} S \); \( S \xrightarrow{\text{stats}} C \); \( S \xrightarrow{\text{done}} C \)

- \( (C \xrightarrow{\text{md}} B; B \xrightarrow{\text{md}} S) \) + \( (C \xrightarrow{\text{start}} B; B \xrightarrow{\text{req}} S) \); \( S \xrightarrow{\text{stats}} C \); \( S \xrightarrow{\text{done}} C \)
Checking refinements

Idea
Devise a typing discipline *sound* w.r.t. well-formedness

Typing judgement

$$\Pi \vdash G : \langle \phi, \Lambda \rangle$$

where
- $\Pi$ are the participants in $G$,
- $\phi$ and $\Lambda$ are the minimal and maximal actions in $G$
Preliminary results

- Ground specs have unique type
- Typable ground global specs are well-formed,
- ...but the vice versa does not hold
- Type inference is decidable for ground specs,
- ...but this is open for refinable specs
Scene 2

Beyond top-down development
“Top-down”

Choreography G
global viewpoint

Local viewpoint

Local viewpoint

Local viewpoint

Synchrony

Asynchrony

Software

- Correctness-by-Design makes a lot of sense when going top-down
“Top-down”

Choreography \( G \) in a global viewpoint

Local viewpoints \( M_1, M_2, \ldots, M_n \)

Synchrony

Asynchrony

Project

Project

Software

- Correctness-by-Design makes a lot of sense when going top-down
“Top-down”

Software

- Correctness-by-Design makes a lot of sense when going top-down
software evolves

- Correctness-by-Design makes a lot of sense when going top-down
“Top-down” & “Bottom-up” approach

Choreography G
    \[ \text{Global viewpoint} \]

\[ M_1 \to M_2 \to M_n \]

Local viewpoint

Synchrony

Asynchrony

Project

Validate

Component

Component

Component

\[ \text{Validate} \]

\[ \text{evolve/replace/compose} \]

Component'

Component'

Component'

New \( M'_1 \to M'_i \to M'_n \)

Local viewpoint

New \( M'_1 \to M'_i \to M'_n \)

Local viewpoint

New \( M'_1 \to M'_i \to M'_n \)

Local viewpoint

Software evolves

- Correctness-by-Design makes a lot of sense when going top-down

Choreographies may help also going bottom-up

[Heraclitus 6th century BC]

\[ \pi \alpha \rho \varepsilon \iota \]

Problem II(a): harnessing round-trip engineering

Are there more usages of global specs than for projecting local specs?
“Top-down” & “Bottom-up” approach

Choreographies may help also going bottom-up [LTY15]

Problem II(a): harnessing round-trip engineering

Are there more usages of global specs than for projecting local specs?

Software evolves

- Correctness-by-Design makes a lot of sense when going top-down
“Top-down” & “Bottom-up” approach

Choreography G: global viewpoint

Local viewpoint

M_1

M_i

M_n

Synchrony

Asynchrony

Project

Validate

Synthesise

Component_1

Component_i

Component_n

Component'_1

Component'_i

Component'_n

New M'_1

New M'_i

New M'_n

New choreography G: global viewpoint

evolve/replace/compose

Software evolves

Correctness-by-Design makes a lot of sense when going top-down
“Top-down" & “Bottom-up" approach

Software evolves

- Correctness-by-Design makes a lot of sense when going top-down
- παντα ρει [Heraclitus 6th century BC]
“Top-down” & “Bottom-up” approach

Choreography G

Global viewpoint

Local viewpoint

Component 1

Component i

Component n

Synchrony

Asynchrony

Project

Validate

Synthesise

Software evolves

- Correctness-by-Design makes a lot of sense when going top-down
- παντα ρει [Heraclitus 6th century BC]
- Choreographies may help also going bottom-up [LTY15]
“Top-down” & “Bottom-up” approach

Choreography $G$ – Global viewpoint

$M_1$, $M_2$, $M_n$ – Local viewpoints

Validate

Component 1, Component 2, Component n

Synchrony

Project

Asynchrony

Synthesise

New $M'_1$, $M'_2$, $M'_n$ – Local viewpoints

Component', Component', Component'

Software evolves

- Correctness-by-Design makes a lot of sense when going top-down
- παντα ρει [Heraclitus 6th century BC]
- Choreographies may help also going bottom-up [LTY15]
“Top-down” & “Bottom-up” approach

Software evolves

- Correctness-by-Design makes a lot of sense when going top-down
- \( \piαντα \rhoει \) [Heraclitus 6th century BC]
- Choreographies may help also going bottom-up [LTY15]

Problem II(a): harnessing round-trip engineering

Are there more usages of global specs than for projecting local specs?
Quite some work for binary ST

- Collingbourne, Kelly. Inference of Session Types From Control Flow. ENTCS 238 (2010)
- Imai, Yuen. Session Type Inference in Haskell. PLACES 2010.
- Graversen, Harbo, Hüttel, Bjerregaard, Poulsen, Wahl. Type Inference for Session Types in the $\pi$-calculus. WS-FM 2016.
- Spaccasassi, Koutavas. Type-based Analysis for Session Inference. FORTE 2016.
- Padovani. Context-Free Session Type Inference. TOPLAS, 41. 2019
- ...
Does retrieving global specs matter?

Some good reasons
- Analysis
- Program comprehension
- Systematic way of documenting software
- Reuse of software

Problems
- Type inference is not all: it requires source code
- Process mining / model learning
  - Analysis / Comparison of protocols [TTWD16]
  - Adaptation: “incompatible” components can be adapted (e.g., with coordination delegates [AIT18, ADGPT19])
– Scene 3 –

[ Choreographic-driven testing ]
Is correctness-by-construction sufficient?

- Local computations deal with data.

Example:

\[
G_{\text{fact}} = C \rightarrow S: \text{Req int}; S \rightarrow C: \text{Res int}
\]

\[
factorialServer(\text{Req, Res}) = \text{Req? } n. \text{Res!fact}(n)
\]

where

\[
\text{fact: int } \rightarrow \text{ int } \quad \text{fact(int } n) = \text{ if } 0 \leq n \leq 1 \text{ then 1 else } n \ast \text{fact(n + 1)}
\]
Is correctness-by-construction sufficient?

- Local computations deal with data.

Example:

\[
G_{\text{fact}} = C \to S : \text{Req int}; S \to C : \text{Res int}
\]

\[
factorialServer(\text{Req}, \text{Res}) = \text{Req? n}. \text{Res!fact(n)}
\]

where

\[
\text{fact: int} \to \text{int} \quad \text{fact(int n)} = \begin{cases} 
1 & \text{if } 0 \leq n \leq 1 \\
 n \ast \text{fact(n - 1)} & \text{else}
\end{cases}
\]

Evolution of components may alter communication patterns. Openness enables changes to the execution context that may alter "compatibility". Example: another server

\[
factorialServer(\text{Req}, \text{Res}) = \text{Req? n}. \text{Res!fact(n)}
\]

\[
\text{if } n < 0 \text{ then } \text{Res!"error"} \text{ else } \text{Res!fact(n)}
\]
Is correctness-by-construction sufficient?

- Local computations deal with data.

Example:

\[ G_{\text{fact}} = \text{C} \to \text{S} : \text{Req int}; \text{S} \to \text{C} : \text{Res int} \]

\[ \text{factorialServer}(\text{Req}, \text{Res}) = \text{Req}? n. \text{Res}! \text{fact}(n) \]

where

\[ \text{fact} : \text{int} \to \text{int} \quad \text{fact}(\text{int} \ n) = \begin{cases} 1 & \text{if } 0 \leq n \leq 1 \\ n \times \text{fact}(n - 1) & \text{else} \end{cases} \]

...and this is still not right! \[ \text{[BMT20,BHTY10]} \]
Is correctness-by-construction sufficient?

- Local computations deal with data.

Example:

\[ G_{\text{fact}} = C \rightarrow S : \text{Req int} ; S \rightarrow C : \text{Res int} \]

\[ \text{factorialServer}(\text{Req, Res}) = \text{Req}\? n. \text{Res}\! \text{fact}(n) \text{ where} \]

\[ \text{fact : int } \rightarrow \text{int} \quad \text{fact}(\text{int } n) = \text{if} \ 0 \leq n \leq 1 \text{ then } 1 \text{ else } n \times \text{fact}(n - 1) \]

...and this is still not right! [BMT20,BHTY10]

- Evolution of components may alter communication patterns
Is correctness-by-construction sufficient?

- **Local computations** deal with data.

  Example:
  \[
  G_{\text{fact}} = C \rightarrow S : \text{Req int}; S \rightarrow C : \text{Res int}
  \]
  \[
  \text{factorialServer}(\text{Req}, \text{Res}) = \text{Req? } n. \text{Res!} \text{fact}(n)
  \]
  where
  \[
  \text{fact}: \text{int} \rightarrow \text{int}; \quad \text{fact}(\text{int } n) = \text{if } 0 \leq n \leq 1 \text{ then } 1 \text{ else } n \ast \text{fact}(n - 1)
  \]

  ...and this is still not right! [BMT20, BHTY10]

- **Evolution** of components may alter communication patterns
- **Openness** enables changes to the execution context that may alter “compatibility”

  Example: another server
  \[
  \text{factorialServer}(\text{Req}, \text{Res}) = \text{Req? } n. \quad \text{if } n < 0
  \text{ then } \text{Res!} \text{”error”}
  \text{ else } \text{Res!} \text{fact}(n)
  \]
Why are choreographies good for testing

Problem II(b): harnessing round-trip engineering

Can global specs support software testing?
Problem II(b): harnessing round-trip engineering

Can global specs support software testing?

Choreographic models can be used

- as test case specifications
- to automatically generate executable tests
- to automatically generate mock components
- to assess coverage of test cases
An abstract framework for model-based testing:

- **Test cases**: a composition of “some deterministic” CFSMs
- Automatic test generation
  \[
  \prod(\text{split(proj(global spec))})
  \]
- **Test compliance**: a criterion for test success (oracle problem)
- **Suitable** tests (not all tests make sense!)
- **Theorem**:
  
  if the global spec is well-formed then generated tests are suitable
– Scene 4 –

[ Beyond Channel-based communication ]
Abstract communication paradigms

Channel based communication could be too “low level”

Often other mechanisms are more appropriate

- Event-Notification
- Publish-Subscribe
- Generative communication
  - Distributed tuple spaces
  - Attribute-based
Abstract communication paradigms

Channel based communication could be too “low level”
Often other mechanisms are more appropriate

- Event-Notification
- Publish-Subscribe
- Generative communication
  - Distributed tuple spaces
  - Attribute-based

Problem III: Abstract coordination mechanisms

Develop new choreographic frameworks
for sophisticated communication
mechanism
A few (natural) questions

1. What safe assumptions on the (distributed) state after interactions?
2. What (behavioural) properties a given communication mechanism enforces?
3. How about statically guaranteeing such properties?
4. What are the relations between message-passing and more abstract communications?
5. Can behavioural abstractions support or improve run-time execution?
6. Can behavioural specifications foster quantitative analysis of CAS?
Drifting away from control-flow... [BCGMMT19,FMMT20,ITT20]

The emphasis is no longer on (dead)lock-freedom: progress becomes data-driven

Generalised interactions

\[ A \rho \xrightarrow{e} e' \rightarrow B \rho' \]

*any* \( A \) satisfying \( \rho \) *generates* data \( e \) *for any* \( B \) satisfying \( \rho' \) *with* \( e' \) *matching* \( e \).

Some benefits

- Weaker (hence more general) notions of correctness
- Choreographies for new domains (e.g., IoT, CPS, Autonomous Systems)
  - multi-roles: many instances may play many roles
  - correctness related to *emergent behaviour*
  - (limited) misbehaviour is toleratated
Some illustrative example

Market place

The broker triggers the protocol; for simplicity it does not take part in the rest of the choreography.

Sellers add items on sale to the market place after the notification from the broker.

Once its items have been advertised, a seller waits for buyers to flag interest in one of the item...

...when a buyer manifests interest in an item of a seller, buyer and seller start a subprocess for the bargaining phase. In this phase the buyer makes an offer and the seller decides if to quit the bargaining, ask for a better offer, or sell the item to the buyer.
Some illustrative example

Robots: possibly playing two roles

\[ \text{charge} > 0 \land \text{bl} \geq \text{eT} \]

\[ \text{reqEn} > 0 \land \text{id} \neq \epsilon \]

\[ \rho(e) \Rightarrow s \in \text{supp} \]

\[ e < \text{qt} \]

\[ \phi_{\text{confirm}} \]

\[ S.\text{id} = s \]

\[ \text{S} \]

\[ \phi_{\text{cancel}} \]
A positive side effect

“Data-driven” specs seem more faithful to actual implementations

- going beyond simulations
- from global specs to queuing networks
− Epilogue −

[ ... ]
Summing up

A quick journey in choreographies in order to discuss

In order to focus on some open issues
- Compositionality
- Refinement
- Choreographic-driven Testing
- Generalisations

I resisted to talk about tool support (a crucial open problem in BehAPI)
I am immensely grateful to my collaborators

[BDLT20] Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese, eM.
Composition and Decomposition of Multiparty Session. Submitted at JLAMP.

[BLT20] Franco Barbanera, Ivan Lanese, eM.
Composing Communicating Systems, Synchronously. ISoLA 2020

[BMT20] Laura Bocchi, Hernán C. Melgratti, eM.
Resolving Non-determinism in Choreographies. ESOP 2014. (Full version To appear on LMCS)

[CGT20] Alex Coto, Roberto Guanciale, eM.

[dLMT20] Ugo de’Liguoro, Hernán C. Melgratti, eM.
Towards Refinable Choreographies. ICE 2020 (To appear).

[FMMT20] Leonardo Frittelli, Facundo Maldonado, Hernán C. Melgratti, eM.
A Choreography-Driven Approach to APIs: The OpenDXL Case Study. COORDINATION 2020

[ITT20] Omar Inverso, Catia Trubiani, eM.
Abstractions for Collective Adaptive Systems. (ISoLA 2020)

[BHTY10] Laura Bocchi, Kohei Honda, eM, Nobuko Yoshida.
A Theory of Design-by-Contract for Distributed Multiparty Interactions. CONCUR 2010

[BCGMMT19] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Hernán C. Melgratti, Ugo Montanari, eM.

[GT19] Roberto Guanciale, eM.

Choreography-Based Analysis of Distributed Message Passing Programs. PDP 2016

[LTY15] Julien Lange, eM, Nobuko Yoshida
From Communicating Machines to Graphical Choreographies. POPL 2015
Thank you!