Building Graphical Choreographies from Communicating Machines
Principles and Applications

Julien Lange

based on works with L. Bocchi, N. Ng, E. Tuosto, and N. Yoshida

May 2017
Session Type Mobility Group

www.mrg.doc.ic.ac.uk
Us ∈ Mobility Research Group

NEWS

SELECTED PUBLICATIONS

2017


Julien Lange, Nicholas Ng, Bernardo Toninho, Nobuko Yoshida: Fencing off Go: Liveness and Safety for Channel-based Programming. *POPL 2017.*


http://mrg.doc.ic.ac.uk/
**OOI Collaboration**

- **TCS’16**: Monitoring Networks through Multiparty Session Types. Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, Nobuko Yoshida
- **LMCS’16**: Multiparty Session Actors. Rumyana Neykova, Nobuko Yoshida
- **FMSD’15**: Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, Nobuko Yoshida
- **TGC’13**: The Scribble Protocol Language. Nobuko Yoshida, Raymond Hu, Rumyana Neykova, Nicholas Ng
Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.
module examples;

global protocol HelloWorld(role Me, role World) {
   hello() from Me to World;
   choice at World {
      goodMorning1() from World to Me;
   } or {
      goodMorning1() from World to Me;
   }
}
Interactions with Industries

Strange Loop
SEPTEMBER 15-17 2016 / PEABODY OPERA HOUSE / ST. LOUIS, MO

Adam Bowen @adamnbowen · Sep 15
I didn’t even know that session types existed an hour ago, but thanks to Nobuko Yoshida’s great talk at #pwiconf, I want to learn more.

Nobuko Yoshida
Imperial College, London

DoC researcher to speak at Golang UK conference
by Vicky Kappogianni
20 July 2016

DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

The Golang UK Conference
Interactions with Industries

F#unctional Londoners Meetup Group

6 days ago · 6:30 PM
Session Types with Fahd Abdeljallal

43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the...

LEARN MORE

Distributed Systems vs. Compositionality

Dr. Roland Kuhn
@rolandkuhn — CTO of Actyx

Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)
Go concurrency verification research at DoC grabs headline

A paper by DoC researchers at POPL on Go concurrency verification was featured in a tech blog and generates a buzz outside of the research community.

A paper by researchers at the department was recently featured in the morning paper, a blog by venture capitalist Adrian Colye, which summarises an important, influential, topical or otherwise interesting paper in the field of computer science every weekday in an easily digestible way by non-researchers. On the 2 Feb 2017 issue of the morning paper, it was highlighted as "the true spirit of POPL (Principles of Programming Languages)".
Selected Publications 2016/2017

- **[ECOOP’17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- **[COORDINATION’17]** Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS’17]** Julien Lange, NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE’17]** Raymond Hu, NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC’17]** Rumyana Neykova, NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL’17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- **[ECOOP’16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC’16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE’16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS’16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[POPL’16]** Dominic Orchard, NY: Effects as sessions, sessions as effects.
Selected Publications 2016/2017

• [CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
Building Graphical Choreographies from Communicating Machines
Principles and Applications

Julien Lange

based on works with L. Bocchi, N. Ng, E. Tuosto, and N. Yoshida

May 2017
Single bird \(\sim\) local behaviour \(\sim\) CFSM
Flock $\simeq$ global behaviour $\simeq$ choreography
Introduction

- Parts of distributed systems change/evolve, not always in a coordinated way,
- these changes are often *not* documented.
- Service oriented systems are sometimes composed dynamically,
- it is often unclear how complex the overall system has become.
- Cognizant’s Zero Deviation Lifecycle.

A *global* point of view of a distributed system is *essential* for top-level management.
Choreography-driven development, cf. Multiparty Session Types top-down approach (POPL’08 & ESOP’12)

Not applicable without *a priori knowledge* of a choreography
Choreography-driven development, cf. Multiparty Session Types

- top-down approach (POPL’08 & ESOP’12)
- Not applicable without a priori knowledge of a choreography
- Our goal: from Communicating Finite-State Machines to Global Graphs
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

"On Communicating Finite-State Machines", Brand & Zafiropulo ('83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

"On Communicating Finite-State Machines", Brand & Zafiropulo (’83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo ('83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Background: CFSMs

“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)
Global Graphs

Four Player Game
Global Graphs

Four Player Game

J. Lange
Building Graphical Choreographies from Communicating Machines
Objectives

Two main objectives:

- **Sound Condition for Safety**: generalised multiparty compatibility
  
  If $S = (M_1, \ldots, M_k)$ is *compatible* then $S$ is “safe”, i.e., every sent message is eventually received and no deadlock.

- **Construction of a Global Graph**: 
  
  If $G$ is the global graph constructed from $S$, then

  $$S = (M_1, \ldots, M_k) \equiv (G_{|_1}, \ldots, G_{|_k})$$
The Plan

1. Build $TS(S)$, the transition system of all *synchronous* executions
2. Check for safety on $TS(S)$ to
   - ensure equivalence between original system and the projections of the choreography,
   - guarantee safety (no deadlock, no orphan message)
3. Build a choreography (global graph) from $TS(S)$, relying on
   - the theory of regions, and
   - Petri nets.
1. Synchronous Transition System of CFSMs
Synchronous Transition System (TS(S))
2. Check for Safety: Generalised Multiparty Compatibility (GMC)

1. Representability
2. Branching Property
Representability

- For each participant: projection of $T_S(S) \approx$ original machine
Checking Compatibility (i) – Representability

Representability

- For each participant: projection of $TS(S) \approx$ original machine
Representability

- For each participant: projection of TS(S) ≈ original machine

\[
\begin{align*}
(A \rightarrow B : \text{bwin}) |_B &= AB?\text{bwin} \\
(B \rightarrow A : \text{sig}) |_B &= BA!\text{sig} \\
(C \rightarrow D : \text{busy}) |_B &= \varepsilon
\end{align*}
\]
Checking Compatibility (ii) – Branching Property

Each branching $n_1$ in TS must be either

$$\xrightarrow{e} \xleftarrow{e'}$$
Checking Compatibility (ii) – Branching Property

Each branching in TS must be either commuting:

- commuting:

\[ n_1 \xrightarrow{e} n_2 \xrightarrow{e'} n_3 \xrightarrow{e} n_4 \xrightarrow{e'} n_1 \]
Checking Compatibility (ii) – Branching Property

Each branching $e_1$ in TS must be either

- or, each last node $n_k$ must be a “well-formed” choice, i.e.,
  - for each participant
    - it receives a different message in each branch, or
    - it is not involved in the choice
  - there is a unique sender

Last node, reachable from $n_1$, from which $e$ and $e'$ can be fired.
J. Lange
Building Graphical Choreographies from Communicating Machines
A → C: cwin
A → B: bwin
C → D: busy
B → C: close
C → A: sig
B → A: free

(A₀, B₀, C₀, D₀)
(A₁, B₀, C₁, D₀)
(A₀, B₀, C₂, D₁)
(A₁, B₀, C₄, D₁)
(A₁, B₁, C₂, D₁)
(A₁, B₁, C₄, D₁)
(A₁, B₂, C₃, D₀)
(A₁, B₂, C₅, D₁)
(A₂, B₀, C₀, D₀)
(A₂, B₀, C₁, D₀)
(A₂, B₀, C₃, D₀)
A → C: cwin
B → C: bwin
C → D: busy

(A0, B0, C0, D0)

(A1, B0, C1, D0)

(C → B: close)

(A1, B0, C4, D1)

(A1, B1, C0, D0)

(A1, B1, C2, D1)

(A1, B1, C5, D1)

(A1, B2, C3, D0)

(A1, B2, C5, D1)

(B → C: close)

(B → A: sig)

(C → D: busy)

(B → C: close)

(B → A: sig)

(C → D: busy)

(C → D: busy)

(C → D: busy)
A : $AB!\text{bwin} \neq AC!\text{cwin}$

D : not involved in the choice

B : $AB?\text{bwin} \neq CB?\text{blose}$

C : $AC?\text{cwin} \neq BC?\text{close}$
3. Build a global graph
We use the work of Cortadella et al. (1998), based on the **theory of regions**, to derive a **safe** and **extended free-choice** Petri net from the Synchronous Transition System.
2: From Petri Net to One-source Petri Net
3: From One-source Petri Net to Joined Petri Net

A → B: bwin
A → C: cwin
B → C: close
C → B: blose
B → A: sig
C → A: msg
A → D: free

A → B: bwin
A → C: cwin
B → C: close
C → B: blose
B → A: sig
C → A: msg
A → D: free
4 & 5 : From Joined Petri Net to Global Graph
Prototype Implementation

https://bitbucket.org/julien-lange/gmc-synthesis

J. Lange
Building Graphical Choreographies from Communicating Machines
Applications of choreography synthesis

1. *Timed systems:*
   
   **Meeting Deadlines Together** (CONCUR 2015)
   Laura Bocchi, Julien Lange & Nobuko Yoshida

2. *Deadlock detection in Go:*
   
   **Static Deadlock Detection for Go by Global Session Graph Synthesis** (CC 2016)
   Nicholas Ng & Nobuko Yoshida:

3. *Analysis of Erlang libraries:*
   
   **Choreography-Based Analysis of Distributed Message Passing Programs** (PDP 2016)
   Ramsay Taylor, Emilio Tuosto, Neil Walkinshaw, & John Derrick
Meeting Deadlines Together (CONCUR 2015)
Laura Bocchi, Julien Lange & Nobuko Yoshida
From Communicating Timed Automata
to Timed Choreographies

Each machine owns several clocks (not shared)

Time elapses at the same pace for each clock
- Construction as in the untimed setting
- Additional safety checks for time constraints
Safety checks for CTAs

- MC: (Generalised) Multiparty Compatibility
- IE: Interaction Enabling
- CE: Cycle Enabling

| Property     | $S \sim (TS(S) | _P \in P)$ | Safety | Progress | Non-Zeno | Eventual Reception |
|--------------|----------------|--------|----------|----------|-------------------|
| MC           | ✓              | ✓      | X        | X        | X                 |
| MC+IE        | ✓              | ✓      | ✓        | X        | X                 |
| MC+CE        | ✓              | ✓      | X        | ✓        | X                 |
| MC+IE+CE     | ✓              | ✓      | ✓        | ✓        | ✓                 |

More at www.doc.ic.ac.uk/~jlange/cta/
Static Deadlock Detection for Go by Global Session Graph Synthesis (CC 2016)
Nicholas Ng & Nobuko Yoshida:
Choreography Synthesis for Go (i)

Some Go facts:

- Developed by Google for multi-core programming
- Concurrency model built on CSP (process calculi)
- Message-passing communication over channels

“Do not communicate by sharing memory; instead, share memory by communicating” – Effective Go (developer guide)
Choreography Synthesis for Go (ii)

```go
default deepThought(replyCh chan int) {
    time.Sleep(75 * time.Millisecond)
    replyCh <- 42
}

func main() {
    ch := make(chan int)
    go deepThought(ch)
    answer := <-ch
    fmt.Printf("The answer is %d\n", answer)
}
```

Go has a **runtime** deadlock detector, but it is very limited!
Choreography Synthesis for Go (iii)

https://www.doc.ic.ac.uk/~cn06/pub/2016/dingo
Summing up

- An effective way of checking properties of CFSMs, and whether one can construct a global graph from them.
- An algorithm based on the theory of regions.
- A CFSMs characterisation of well-formed *generalised global types*.
- Applications:
  - An extension for communicating timed automata
  - Deadlock detection in Go
  - Analysis of Erlang libraries
Thanks!

Any questions?

https://bitbucket.org/julien-lange/gmc-synthesis
Violating the no-race condition
Violating the no-race condition
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
A → B : cake
A → B : cake

AB : apple
CB : ε
AC : ε
BA : ε
Violating the no-race condition
Violating the no-race condition
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
B → A : biscuit

AB : apple
CB : date
AC : ε
BA : ε
Violating the no-race condition

A → B: apple, A → C: carrot
A → C: cabbage, C → B: date
C → B: date, B → A: banana
B → A: banana, A → B: apple
A → B: cake, B → A: biscuit

AB : apple
CB : date
AC : ε
BA : ε
Violating the no-race condition

J. Lange
Building Graphical Choreographies from Communicating Machines
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
A → B : cake
B → A : biscuit
B → A : biscuit

AB : apple
CB : ε
AC : ε
BA : banana
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
A → B : cake
B → A : biscuit
B → A : biscuit

AB !apple
AC !carrot
AC !cabbage
BA ?banana
BA ?banana
AB !cake
BA ?biscuit
AB !cake
BA ?biscuit

AB ?apple
CB ?date
BA !banana
BA !banana
AB ?cake
BA !biscuit
AB ?cake
BA !biscuit

AC ?cabbage
AC ?carrot
CB ?date
BA !banana
BA !banana
AC ?cabbage
AC ?carrot
CB ?date

A B C
A Ñ B Ñ C
A Ñ B Ñ C
A Ñ B Ñ C

J. Lange
Building Graphical Choreographies from Communicating Machines

AB : apple
CB : ε
AC : ε
BA : ε
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
B → A : biscuit

A → B : apple 
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
B → A : biscuit

AB : apple · cake
CB : ε
AC : ε
BA : ε
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
C → B : date
B → A : banana
B → A : banana
A → B : cake
A → B : cake
B → A : biscuit
B → A : biscuit
A → B : apple
A → B : apple

AB : cake
CB : ε
AC : ε
BA : ε
Violating the no-race condition

A → B: apple
A → C: carrot
A → C: cabbage
C → B: date
C → B: date
B → A: banana
B → A: banana
A → B: cake
A → B: cake
A → B: apple
B → A: biscuit
B → A: biscuit

AB: cake
CB: ε
AC: ε
BA: biscuit
Violating the no-race condition

\[
\begin{align*}
A & \rightarrow B : \text{cake} \\
A & \rightarrow C : \text{cabbage} \\
C & \rightarrow B : \text{date} \\
B & \rightarrow A : \text{banana} \\
A & \rightarrow B : \text{cake} \\
A & \rightarrow B : \text{apple} \\
B & \rightarrow A : \text{biscuit} \\
\end{align*}
\]

AB : cake
CB : \(\varepsilon\)
AC : \(\varepsilon\)
BA : biscuit
Violating the no-race condition

A → B : apple
A → C : cabbage
B → A : banana
A → B : cake
B → A : biscuit
A → C : carrot
C → B : date
B → A : banana
A → B : apple
B → A : biscuit

AB : cake
CB : ε
AC : ε
BA : biscuit
Violating the no-race condition

A → B : apple
A → C : carrot
A → C : cabbage
C → B : date
B → A : banana
B → A : banana
A → B : cake
A → B : cake
B → A : biscuit
B → A : biscuit

AB : cake
CB : ε
AC : ε
BA : biscuit
References

- Julien Lange, Emilio Tuosto, and Nobuko Yoshida. “From Communicating Machines to Graphical Choreographies”. In: *POPL*. 2015
- Laura Bocchi, Julien Lange, and Nobuko Yoshida. “Meeting Deadlines Together”. In: *CONCUR*. 2015
- Ramsay Taylor et al. “Choreography-Based Analysis of Distributed Message Passing Programs”. In: *PDP 2016*
- Nicholas Ng and Nobuko Yoshida. “Static deadlock detection for concurrent go by global session graph synthesis”. In: *CC 2016*
Related work (i)


- Julien Lange and Emilio Tuosto. “Synthesising Choreographies from Local Session Types”. In: *CONCUR*. LNCS. Springer, 2012


Related work (ii)


