Multiparty Session Types and their Applications

http://mrg.doc.ic.ac.uk/

Nobuko Yoshida, Rumyana Neykova
Imperial College London
The Kohei Honda Prize for Distributed Systems  Queen Mary, University of London

Posted with permission from QMUL on 17th Dec 2013. Original article written by Edmund Robinson.

This prize was instituted in 2013 and is awarded annually to one undergraduate student and one postgraduate student in recognition of their achievement in applying the highest quality scientific and engineering principles in the broad area of Distributed Systems. This is the area in which Dr Honda concentrated most of his teaching, and it is also the area in which he conducted his research. Its primary funding comes from a donation from his family, who wished to commemorate Dr Honda in this way. Additional funding has come from Dr Honda’s own ETAPS Award. This prize is sponsored by Springer Verlag, and awarded annually by the ETAPS committee in recognition of an individual’s research contribution. Dr Honda received the first such award posthumously, and the awarding panel expressed a wish that the funding be used to supplement this prize fund. The laudation for this award, written by Dr Honda’s colleague, Prof Vladimiro Sassone is included later.

About Dr Honda

Kohei Honda was born and lived the first part of his life in Japan. Like many scientists he was fascinated by the idea of finding basic explanatory theories, like the physicists looking for grand unified theories of the universe. Kohei, though, was passionately interested in finding the right basic explanatory theory for the process of computation. Most academics agree that the basic theory

Winners 2013

Ms Anna Pawlicka
2013 winner (Undergraduate)  source: QMUL

Mr. Valmir Negacevshi
2013 winner (Postgraduate)  source: QMUL
Outline

➤ Idioms for Interaction
➤ Multiparty Session Types
➤ Scribble and Applications to a Large-scale Cyberinfrastructure
➤ Recent Results on Multiparty Session Types
Programming languages are tools which offer frameworks of abstraction for such activities – promoting or limiting them:

- Imperative
- Functional
- Logical
On Programs and Programming

- The most fundamental element of a PL
  in this context is a set of operations
  it is based on:

  Imperative: assignment, jump.
  Functional: \( \beta \)-reduction.
  Logical: unification.

- Another element is how we can combine,
  or structure, these operations:

  Imperative: sequential composition, if-then-else,
  while, procedures, module, ...

  Functional: application, product, union,
  recursion, modules, ...
UNSTRUCTURED:

STRUCTURED:

```
Var a: array[MAX] of int;

Procedure sort(l, r: int);

Var i, j, x: int;

x = (l+r) div 2;  // Choose a pivot.

While a[i] < x Do i := i+1 End
While a[j] > x Do j := j+1 End
If i >= j Then Swap (i, j); i := i+1; j := j-1; End
Until i < j;
If i < j Then Sort (l, j) End
If i > j Then Sort (i, r) End
Recursively sort two parts.

Procedure swap(i, j: int)
Var w: int;

w := a[i]; a[i] := a[j]; a[j] := w
End
```
Quicksort in pure lambda:

\[
((\lambda x. y((axy)((axy)((axy)((axy)((axy)((axy)))lq.l)l))((axy)))\)l\((\lambda x.l)l))l((axy))\]

Quicksort with combinators:

\[
Y((\lambda f. \lambda l. \\
\hspace{1cm} (\text{Isnile } l) \hspace{1cm} (\text{Concat } \hspace{0.1cm} f \hspace{0.1cm} \text{Filter } (\lambda y. \text{LT } y \hspace{0.1cm} (\text{Car } l))) \hspace{1cm} (\text{Cons } \hspace{0.1cm} \text{Car } l) \hspace{1cm} (\text{Filter } (\lambda y. \text{Mey } \hspace{0.1cm} (\text{Car } l) \hspace{0.1cm} (\text{Car } l))))
\]
**Quicksort in ML:**

\[
\begin{align*}
\text{fun} & \quad \text{qs} \quad \text{nil} : \text{int list} = \text{nil} \\
\text{let} & \quad \text{val} \quad \text{small} = \\
\quad & \quad \text{filter} \quad (\text{fn} \ y \Rightarrow y < x) \quad \text{and} \quad \text{large} = \\
\quad & \quad \text{filter} \quad (\text{fn} \ y \Rightarrow y \geq x) \\
\quad \text{in} & \quad \text{qs} \quad \text{small} @ [x] @ \text{qs} \quad \text{large} \\
\text{end} \\
\text{fun} & \quad \text{filter} \quad p \quad \text{nil} = \text{nil} \\
\text{let} & \quad \text{filter} \quad p \quad (x :: r) = \\
\quad & \quad \text{if} \quad p \ x \ \text{then} \quad x := \ \text{filter} \quad p \ r \ \text{else} \quad \text{filter} \quad p \ r
\end{align*}
\]
The π-calculus as a Descriptive Tool

\[ \pi \quad M :: = x \mid x. M \mid M N. \]

with \[ \pi :: = x(a) \mid x(a). \]

\[ \pi \]

\[ P :: = \Sigma_{E. P} \mid \Pi_{Q} \mid \omega P \mid ! P \mid \emptyset. \]

\[ \lambda \in \pi \]

\[ [x]_u \triangleq \pi(a). \]

\[ [x. M]_u \triangleq \pi(x.a). [M]_u. \]

\[ ([M] N)_u \triangleq (\nu x) ([M]_y [x/a] | [x=N]) \]

with \[ [x=N]_u \triangleq ! x(a). [M]_u. \]
Examples of Representable Computation

- λ-calculus [NPW89, Milner92, Milner82, ...]
- Concurrent Object [Walker81]
- 1-order term passing [Smyth92]
- Various data structures [Milner92, ...]
- Proof Nets [Abadi and Scott83]
- Arbitrary `constant` interaction [Hos94]
- Strategies on Games [HO95]
The Role of Types in TR-calculus.

- (classification) How can we classify name-passing interactive behaviours, i.e., behaviours representable in TR-calculus? What classes ("types") of behaviours can we find in the calculus?

- (safety) Is this program/system in the safe (or correct, relevant,...) classes of behaviours? Can the safety be preserved compositionally?
Functional Types

\[
\text{Nat} = \text{Nat} \\
\text{Succ, Div, Id, } \ldots \\
\text{Nat} = \text{Bool} \\
\text{Iszero, } \ldots \\
\text{Nat} = \text{Nat} \\
\text{And, Or, Not, } \ldots \\
\text{Nat} = \text{Nat} = \text{Nat} \\
\text{Add, Sub, Mult, Div, Eor, } \ldots \\
\text{Bool} = \text{Bool} \\
\text{True, False, } \ldots
\]

with operation:

\[
\begin{cases}
\text{if } d \in \Theta \Rightarrow e : d = f \circ e : \Theta.
\end{cases}
\]

else undefined.

function application.
When it comes to processes, composition becomes:

\[
\begin{align*}
3 + 5 &= 8 \\
\end{align*}
\]

But some composition is dangerous!

Therefore we type processes, the connection is prohibited.

Explosion!
- divergence
- deadlock
- runtime error
Implementing ATM

1. Put the card in.
2. The system asks what you want.
   - Withdraw?
   - Deposit?
   - Quit?
3. Select, for example, withdraw, then key in the amount.
4. Banking system.
5. \[\text{Ok! Take your money}\]
6. \[\text{Overdraft! Try again.}\]
Implementing ATM

ATM(cb) =

\[ \text{encode} \]

ENCODING

```
?ok;
  2! ok; !X; ATM(cb),

?overdraft:
  2! overdraft; ATM(cb),

?bad;
  2! overdraft; ATM(cb),

?bad;
  2! X; ATM(cb),
```

ENCODING

```
0. (\text{enc} \{0, 1, 2, 3\}) \text{if} \text{cb},
0. \text{enc}(\text{cb}) \text{if} \text{cb},
0. \text{enc}(\text{cb}) \text{if} \text{cb},
0. (\text{enc} \{0, 1, 2, 3\}) \text{if} \text{cb},
```

```
0. (\text{enc} \{0, 1, 2, 3\}) \text{if} \text{cb},
0. \text{enc}(\text{cb}) \text{if} \text{cb},
0. \text{enc}(\text{cb}) \text{if} \text{cb},
0. (\text{enc} \{0, 1, 2, 3\}) \text{if} \text{cb},
```
The way to organise software is increasingly based on communications (Cloud Computing, many cores, message-passing parallel computation, ...)

Question

- How to formally abstract/specify/implement/control communications?
- How to apply mobile processes and their type theories to real distributed applications and programming languages?
The way to organise software is increasingly based on communications (Cloud Computing, many cores, message-passing parallel computation, ...)

**Question**

- How to formally abstract/specify/implement/control communications?
- How to apply mobile processes and their type theories to real distributed applications and programming languages?

**large-scale cyberinfrastructure for e-Science**
Ocean Observatories Initiative

- A NSF project (400M$, 5 Years) to build a cyberinfrastructure for observing oceans around US and beyond.
- Real-time sensor data constantly coming from both off-shore and on-shore (e.g. buoys, submarines, under-water cameras, satellites), transmitted via high-speed networks.
Ocean Observatories Initiative
Challenges

➢ The need to specify, catalogue, program, implement and manage *multiparty message passing protocols*.

➢ Communication assurance
  ➢ Correct message ordering and synchronisation
  ➢ Deadlock-freedom, progress and liveness
  ➢ Dynamic message monitoring and recovery
  ➢ Logical constraints on message values

➢ Shared and used over a long-term period (e.g. 30 years in OOI).
Why Multiparty Session Types?

➡️ Robin Milner (2002): *Types are the leaven of computer programming; they make it digestible.*

★★→ Can describe communication protocols as *types*
★★→ Can be materialised as *new communications programming languages* and *tool chains.*

➡️ *Scalable* automatic verifications (deadlock-freedom, safety and liveness) without *state-space explosion problems* (*polynomial time complexity*).

➡️ Extendable to *logical verifications* and flexible *dynamic monitoring.*
Dialogue between Industry and Academia

Binary Session Types [PARL’94, ESOP’98]

↓

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

↓

Formalisation of W3C WS-CDL [ESOP’07]

↓

Scribble at $\pi_4$ Technology
CDL Equivalent

- Basic example:

```plaintext
package HelloWorld {
    roleType YouRole, WorldRole;
    participantType You{YouRole}, World{WorldRole};
    relationshipType YouWorldRel between YouRole and WorldRole;
    channelType WorldChannelType with roleType WorldRole;

    choreography Main {
        WorldChannelType worldChannel;

        interaction operation=hello from=YouRole to=WorldRole
            relationship=YouWorldRel channel=worldChannel {
                request messageType=Hello;
            }
    }
}
```

Dr Gary Brown (Pi4 Tech) in 2007
Scribble Protocol

- "Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling" - Kohei Honda 2007

- Basic example:

```r
protocol HelloWorld {
    role You, World;
    Hello from You to World;
}
```
Dialogue between Industry and Academia

Binary Session Types [PARL’94, ESOP’98]

↓

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

↓

Formalisation of W3C WS-CDL [ESOP’07]

↓

Scribble at \( \Pi^4 \) Technology

↓

Multiparty Session Types [POPL’08]
Dialogue between Industry and Academia

Binary Session Types [PARL’94, ESOP’98]

⇒

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

⇒

Formalisation of W3C WS-CDL [ESOP’07]

⇒

Scribble at π4 Technology

⇒

Multiparty Session Types [POPL’08]
Binary Session Types: Buyer-Seller Protocol

Buyer

- title
- quote
- ok
- address
- date
- quit

Seller

branch
! String ! Int ! { ok : ! String ! ? Date ; end , quit : end }
\[ \text{branch} \]

```
! String ; ? Int ; + { ok : ! String ; ? Date ; end , quit : end }

dual ? String ; ! Int ; & { ok : ? String ; ! Date ; end , quit : end }
```
Multiparty Session Types

Buyer 1

- title
- quote
- quote ÷ 2

Seller

- quote

Buyer 2

- ok
- address
- date
Multiparty Session Types

Buyer 1 → Seller
- title
- quote
- quote ÷ 2

Buyer 1 → Buyer 2
- ok
- address
- date
Session Types Overview

Properties
- Communication safety (no communication mismatch)
- Communication fidelity (the communication follow the protocol)
- Progress (no deadlock/stuck in a session)
Dynamic Monitoring
[RV’13, COORDINATION’14, FMSD’15]

Global Type

Projection

Local Type

Local Type

Local Type

Program Alice

Program Bob

Program Carol

Dynamic Monitoring

Dynamic Monitoring

Dynamic Monitoring

Dynamic Monitoring
Two Buyer Protocol in Scribble

type <java> "java.lang.String" from "rt.jar" as String

global protocol TwoBuyers(role A, role B, role S) {
    title(String) from A to S;
    quote(Integer) from S to A, B;
    rec LOOP {
        share(Integer) from A to B;
        choice at B {
            accept(address:String) from B to A, 
            date(String) from S to B;
            } or {
            retry() from B to A, S;
            continue LOOP;
            } or {
            quit() from B to A, S;
            }     }
}     }
module Bookstore_TwoBuyers_A;

type <java> "java.lang.Integer" from "rt.jar" as Integer;
type <java> "java.lang.String" from "rt.jar" as String;

local protocol TwoBuyers_A at A(role A, role B, role S) {
  title(String) to S;
  quote(Integer) from S;
  rec LOOP {
    share(Integer) to B;
    choice at B {
      accept(address: String) from B;
    } or {
      retry() from B;
      continue LOOP;
    } or {
      quit() from B;
    } } }
https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Negotiate+Protocol
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {

}
OOI agent negotiation 3/5 (choice)

type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
    propose(SAP) from C to P;

    choice at P {
        accept() from P to C;
        confirm() from C to P;
    } or {
        reject() from P to C;
    } or {
        propose(SAP) from P to C;
    }
}
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
  propose(SAP) from C to P;

  choice at P {
    accept() from P to C;
    confirm() from C to P;
  } or {
    reject() from P to C;
  } or {
    propose(SAP) from P to C;
    choice at C {
      accept() from C to P;
      confirm() from P to C;
    } or {
      reject() from C to P;
    } or {
      propose(SAP) from C to P;
    }
  }
}
OOI agent negotiation 5/5 (recursion)

type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
   propose(SAP) from C to P;
   rec X {
      choice at P {
         accept() from P to C;
         confirm() from C to P;
      } or {
         reject() from P to C;
      } or {
         propose(SAP) from P to C;
         choice at C {
            accept() from C to P;
            confirm() from P to C;
         } or {
            reject() from C to P;
         } or {
            propose(SAP) from C to P;
            continue X;
         }
      }
   }
}
Local protocol projection (Negotiation Consumer)

// Global
propose(SAP) from C to P;
rec START {
    choice at P {
        accept() from P to C;
        confirm() from C to P;
    } or {
        reject() from P to C;
    } or {
        propose(SAP) from P to C;
        choice at C {
            accept() from C to P;
            confirm() from P to C;
        } or {
            reject() from C to P;
        } or {
            propose(SAP) from C to P;
            continue START;
        }
    }
}

// Projection for Consumer
propose(SAP) to P;
rec START {
    choice at P {
        accept() from P;
        confirm() to P;
    } or {
        reject() from P;
    } or {
        propose(SAP) from P;
        choice at C {
            accept() to P;
            confirm() from P;
        } or {
            reject() to P;
        } or {
            propose(SAP) to P;
            continue START;
        }
    }
}
FSM generation (Negotiation Consumer)
Scribble Community

- **Webpage:**
  - www.scribble.org

- **GitHub:**
  - https://github.com/scribble

- **Tutorial:**
  - www.doc.ic.ac.uk/~rhu/scribble/tutorial.html

- **Specification (0.3)**
  - www.doc.ic.ac.uk/~rhu/scribble/langref.html
Figure 5: A coordinated set of autonomous underwater vehicles
Figure 3: Observatory comprised of ships, aircraft and autonomous vehicles linked to assimilation modeling capabilities on shore
Applications

OOI Governance

Protocol Verification

MPI code generations

ZDLC: Process Modeling
Dynamic Monitoring
[RV’13, COORDINATION’14, FMSD’15]

Global Type

Projection

Local Type

Dynamic Monitoring

Program Alice

Dynamic Monitoring

Program Bob

Dynamic Monitoring

Program Carol
Code Generation [CC’15]

Global Type

Projection

Local Type

Generation

Program Alice

Local Type

Generation

Program Bob

Local Type

Generation

Program Carol
Synthesis
[ICALP’13, POPL’15, CONCUR’15]
Multiparty Compatibility in Communicating Automata

Synthesis and Characterisation of Multiparty Session Types

Nobuko Yoshida

Pierre-Malo Denielou

ICALP’13
1. Deterministic
2. No-Mixed State
3. Compatible

[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.
Zero Deviation Life Cycle Platform
A complete parallel programming workflow

- Captures parallel interaction patterns by Pabble language
- Combines with sequential computation kernels in C
- Generates communication safe & deadlock free MPI programs
- Optimisation as part of merging technique

Diagram:
- Communication protocol
  - Custom Pabble global protocols
  - Common protocols or repository

- Sequential code
  - Sequential kernels (C99)

Output(s)
- Optimised MPI application
- Non-Optimised MPI application
Example: Simple search engine

Scatter-Gather protocol

- Distribute query to all nodes
- Nodes collect relevant records
- Results gathered and merged
- Display results to user

```cpp
const N = 2..max;
global protocol ScatterGather(role Worker[1..N]) {
    Init() from __self to __self;
    Map(S) from Worker[1] to __All;
    Reduce(T) from __All to Worker[1];
    Finish() from __self to __self;
}
```
Example: search engine

Merging backbone with kernels

```
#pragma pabble type S
typedef void S; MPI_Datatype MPI_S = MPI_BYTE;
#pragma pabble type T
typedef void T; MPI_Datatype MPI_T = MPI_BYTE;
#pragma pabble Init
  bufMap0_r = calloc(meta.buflen(Map), sizeof(S));
#pragma pabble Map
  bufMap0_s = pabble_sendq_dequeue();
  MPI_Scatter( ..., MPI_S, Worker_RANK(1), ... );
  free(bufMap0_s);
  pabble_recvq_enqueue(Map, bufMap0_r);
#pragma pabble Reduce
  bufReduce1_r = calloc(meta.buflen(Reduce)*meta.nprocs, sizeof(T));
#pragma pabble Finish
```
Evaluation

Productivity: Flexibility

Reusable protocols
- e.g. scatter-gather
- e.g. stencil

Berkeley Dwarfs [CACM’09]
- Representative parallel computing patterns
- 4 of 5 HPC patterns

<table>
<thead>
<tr>
<th>Berkeley HPC Dwarfs</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Grid</td>
<td>Yes</td>
</tr>
<tr>
<td>Particle Methods</td>
<td>Yes</td>
</tr>
<tr>
<td>Structured Grid</td>
<td>Yes</td>
</tr>
<tr>
<td>Dense Matrix</td>
<td>Yes</td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Berkeley HPC Dwarfs</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Grid</td>
<td>Yes</td>
</tr>
<tr>
<td>Particle Methods</td>
<td>Yes</td>
</tr>
<tr>
<td>Structured Grid</td>
<td>Yes</td>
</tr>
<tr>
<td>Dense Matrix</td>
<td>Yes</td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

π
**Evaluation**

**Productivity: Effort**

Protocols in repository
- Use backbone directly
- Write kernel
- Effort = \( \frac{K}{B+K} \)

Custom protocols
- Write Pabble protocol
- Tool generate backbone
- Write kernel
- Effort = \( \frac{P+K}{B+K} \)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Protocol Type</th>
<th>Pabble LOC (P)</th>
<th>Backbone LOC (B)</th>
<th>Kernel LOC (K)</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>heateq</td>
<td>stencil*</td>
<td>15</td>
<td>154</td>
<td>335</td>
<td>0.69</td>
</tr>
<tr>
<td>nbody</td>
<td>ring*</td>
<td>15</td>
<td>93</td>
<td>228</td>
<td>0.71</td>
</tr>
<tr>
<td>wordcount</td>
<td>scatter-gather*</td>
<td>8</td>
<td>76</td>
<td>176</td>
<td>0.70</td>
</tr>
<tr>
<td>adpredictor</td>
<td>scatter-gather*</td>
<td>8</td>
<td>76</td>
<td>182</td>
<td>0.71</td>
</tr>
<tr>
<td>montecarlo</td>
<td>scatter-gather*</td>
<td>8</td>
<td>76</td>
<td>70</td>
<td>0.48</td>
</tr>
<tr>
<td>montecarlo-mw</td>
<td>master-worker*</td>
<td>10</td>
<td>82</td>
<td>70</td>
<td>0.46</td>
</tr>
<tr>
<td>LEsolver</td>
<td>wraparound mesh</td>
<td>15</td>
<td>132</td>
<td>208</td>
<td>0.66</td>
</tr>
<tr>
<td>matvec</td>
<td>custom</td>
<td>15</td>
<td>130</td>
<td>117</td>
<td>0.41</td>
</tr>
<tr>
<td>fft64</td>
<td>6-step butterfly</td>
<td>11</td>
<td>64</td>
<td>134</td>
<td>0.68</td>
</tr>
</tbody>
</table>

**Effort ratio**
\[ \pi = \frac{LOC \text{ savings}}{LOC} \]
Language and Implementations

➤ Carrying out large-scale experiences with OOI, VMWare, Red Hat, Congnizant, Pivotal, Amazon, AMQP, RabbitMQ

➤ JBoss SCRIBBLE [ICDCIT’10,COB’12,TGC’13] and ZDLC projects

➤ High-performance computing
Session Java [ECOOP’08,ECOOP’10,Coordination’11]
⇒ Multiparty Session C and MPI
[TOOLS’12,Hearts’12,EuroMPI’12,PDP’14,CC’15,OOPSLA’15]

➤ Multiparty session languages Ocaml, Java, C, MPI, Python, Scala, Jolie, Haskell, Erlang

➤ Effect and Concurrent Haskell [POPL’16]

➤ Practical interruptible conversations: Distributed dynamic verification with session types and Python [RV’13,FMCD’15]

➤ Multiparty Session Actors [COORDINATION’14]
Multiparty Session Type Theory

Multiparty Asynchronous Session Types [POPL’08,JACM]

Progress

- Inference of Progress Typing [Coordination’13]

Asynchronous Optimisations and Resource Analysis

- Global Principal Typing in Partially Commutative Asynchronous Sessions [ESOP’09]
- Higher-Order Pi-Calculus [TLCA’07,TLCA’09,Info.&Comp]
- Buffered Communication Analysis in Distributed Multiparty Sessions [CONCUR’10]
Extensions of Multiparty Session Types

- Multiparty Symmetric Sum Types [Express’10]
- Trustworthy Pervasive Healthcare Services via Multi-party Session Types [FHIES’12]
- Parameterised Multiparty Session Types [FoSSaCs’10, LMCS, SPLASH’15]
- Dynamic Multirole Session Types [POPL’11]
- Nested Multiparty Sessions [CONCUR’12]
- Timed Multiparty Session Types [CONCUR’14]

Dynamic Monitoring

- Monitoring Networks through Multiparty Sessions [TGC’11] [FORTE’13]
Automata Theories
- Multiparty Session Automata [ESOP’12]
- Synthesis in Communicating Automata [ICALP’13]
- From communicating machines to graphical choreographies [POPL’15]
- Meeting Deadlines Together [CONCUR’15]

Denotational and Trace Semantics
- Expressiveness of Multiparty Session Types [FSTTCS’15]

Petri Nets
- Multiparty Session Nets [TGC’14]

Typed Behavioural Theories
- On Asynchronous Eventful Session Semantics [FORTE’11]
- Governed Session Semantics [CONCUR’13]
- Characteristic Bisimulations for Higher-Order Session Processes [CONCUR’15]
Choreography Languages

- Compositional Choreographies [CONCUR’13]

Logics

- Design-by-Contract for Distributed Multiparty Interactions [CONCUR’10]
- Specifying Stateful Asynchronous Properties [CONCUR’12]
- Multiparty, Multi-session Logic [TGC’12]
- Multiparty Session Types as Coherence Proofs [CONCUR’15]
Session Type Reading List

➤ Home Page http://mrg.doc.ic.ac.uk/

➤ [ESOP’98] Language Primitives and Type Disciplines for Structured Communication-based Programming, Honda, Vasconcelos and Kubo

➤ [SecRet’06] Language Primitives and Type Disciplines for Structured Communication-based Programming Revisited, Yoshida and Vasconcelos, ENTCS.


➤ [POPL’15] From communicating machines to graphical choreographies, Lange, Tuosto and Yoshida.

[ECOOP’08] Session-Based Distributed Programming in Java, Hu, Yoshida and Honda.


[CC’15] Protocols by Default: Safe MPI Code Generation based on Session Types, Ng, Coutinho and Yoshida.
A rare cluster of qualities

From the team of OOI CI:

*Kohei has lead us deep into the nature of communication and processing. His esthetics, precision and enthusiasm for our mutual pursuit of formal Session (Conversation) Types and specifically for our OOI collaboration to realize this vision in very concrete terms were, as penned by Henry James, lessons in seeing the nuances of both beauty and craft, through a rare cluster of qualities - curiosity, patience and perception; all at the perfect pitch of passion and expression.*