Scribble, Runtime Verification and Multiparty Session Types

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

Nobuko Yoshida
Imperial College London
In collaboration with:

Matthew Arrott (OOI)
Gary Brown (Red Hat)
Stephen Henrie (OOI)
Bippin Makoond (Cognizant/Qualit-e)
Michael Meisinger (OOI)
Matthew Rawlings (ISOTC68/USB)
Alexis Richardson (RabbitMQ/Pivotal)
Steve Ross-Talbot (Cognizant/Qualit-e)

and all our academic colleagues

Laura Bocchi, Tzu-Chun Chen, Tiago Cogumbleiro, Romain Demangeon, Pierre-Malo Denielóu, Juliana Franco, Luca Fossati, Dimitrios Kouzapas, Julien Lange, Rumyana Neykova, Nicholas Ng, Weizhen Yang
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

➤ Background
➤ Multiparty Session Types
➤ Scribble and Applications to a Large-scale Cyberinfrastructure
➤ Recent Works
Communication is Ubiquitous

- Internet, the WWW, Cloud Computing, the next-generation manycore chips, message-passing parallel computations, large-scale cyberinfrastructure for e-Science.

- The way to organise software is increasingly based on communications.

- Applications need *structured* series of communications.

> **Question**

- How to formally abstract/specify/implement/control communications?
Communication is Ubiquitous

➤ Internet, the WWW, Cloud Computing, the next-generation manycore chips, message-passing parallel computations, large-scale cyberinfrastructure for e-Science.

➤ The way to organise software is increasingly based on communications.

➤ Applications need *structured* series of communications.

Question

➤ How to formally abstract/specify/implement/control communications?
Communication is Ubiquitous

- Internet, the WWW, Cloud Computing, the next-generation manycore chips, message-passing parallel computations, large-scale cyberinfrastructure for e-Science.

- The way to organise software is increasingly based on communications.

- Applications need structured series of communications.

Question \[\Rightarrow\] Multiparty session type theory

\[\Rightarrow\] How to formally abstract/specify/implement/control communications?
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
Pi calculus versus Petri nets: Let us eat “humble pie” rather than further inflate the “Pi hype”

W.M.P. van der Aalst

Abstract. In the context of Web Service Composition Languages (WSCLs) there is an ongoing debate on the best foundation for Process-Aware Information Systems (PAISs): Petri nets or Pi calculus. Example of PAISs are Workflow Management (WFM), Business Process Management (BPM), Business-to-Business (B2B), Customer Relationship Management (CRM), Enterprise Resource Planning (ERP) systems. Clearly, the web-service paradigm will change the architecture of these systems dramatically. Therefore, triggered by industry standards such as SOAP, WSDL, UDDI, etc., standards are being proposed for orchestrating web services. Examples of such WSCLs are BPEL4WS, BPML, WSFL, WSCI, and XLANG. In the debate on Petri nets versus Pi calculus many players in the “WSCL marketplace” are using demagogic arguments not based
Steve

Thanks for that. I believe the pi-calculus team ought to be able to do something with it -- you seem to be taking it in that direction already.

Nobuko, Kohei: I thought we ought to try to model use-cases in pi-calculus, with copious explanations in natural language, aiming at seeing how various concepts like role, transaction, .. would be modelled in pi. I am hoping to try this one when I get time; you might like to try too, and see if we agree!

Robin
CDL Equivalent

• Basic example:

```clojure
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:

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

Binary Session Types \([\text{PARL’94, ESOP’98}]\)

\[\Downarrow\]

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

\[\Downarrow\]

Formalisation of W3C WS-CDL \([\text{ESOP’07}]\)

\[\Downarrow\]

Scribble at \(\Pi^4\) Technology

\[\Downarrow\]

Multiparty Session Types \([\text{POPL’08}]\)
Dialogue between Industry and Academia

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

\[ \Downarrow \]

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

\[ \Downarrow \]

Formalisation of W3C WS-CDL [ESOP’07]

\[ \Downarrow \]

Scribble at \( \Pi^4 \) Technology

\[ \Downarrow \]

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

<table>
<thead>
<tr>
<th>Buyer</th>
<th>Seller</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td></td>
</tr>
<tr>
<td>quote</td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>address</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td></td>
</tr>
<tr>
<td>quit</td>
<td></td>
</tr>
</tbody>
</table>

branch
branch

Buyer

- title
- quote
- OK
- address
- date
- quit

Seller

String, Int, \{ OK: String, Date; end, quit: end \}
branch

Buyer

- title
- quote
- address
- date
- quit

Seller

- P has T
- Q has T dual
- PIQ typable

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

Buyer 1, Seller, Buyer 2

- title
- quote
- quote ÷ 2
- address
- date
- ok
Multiparty Session Types

Buyer 1

Seller

Buyer 2

title
quote
quote ÷ 2

quote

ok

address
date
Properties

- Communication safety (no communication mismatch)
- Communication fidelity (the communication follow the protocol)
- Progress (no deadlock/stuck in a session)
Evolution Of MPST

- Binary Session Types [THK98, HVK98]
- Multiparty Session Types [POPL’08]
- A Theory of Design-by-Contract for Distributed Multiparty Interactions [Concur’11]
- Multiparty Session Types Meet Communicating Automata [ESOP’12, ICALP’13]
- Network Monitoring through Multiparty Session Types [FMOODS’13]

- SPY: Local Verification of Global Protocols [RV’13]
- Distributed Runtime Verification with Session Types and Python [RV’13]
Ocean Observatory Initiative (OOI)

**OOI aims:** to deploy an infrastructure (global network) to expand the scientists’ ability to remotely study the ocean

**Usage:** Integrate real-time data acquisition, processing and data storage for ocean research,…
OOI: verification challenges

- applications written in different languages, running on heterogeneous hardware in an asynchronous network.
- different authentication domains, external untrusted applications
- various distributed protocols
- requires correct, safe interactions
Session Types for Runtime Verification

- **Methodology**
  - Developers design protocols in a dedicated language - Scribble
  - Well-formedness is checked by Scribble tools
  - Protocols are projected into local types
  - Local types generate monitors
2-level Verification

1. Writing correct global protocols with **Scribble Compiler**

2. Verify programs via **local monitors**
2-level Verification

1. Writing correct global protocols with **Scribble Compiler**

2. Verify programs via *local monitors*
What is Scribble?

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.

Find out more ...

Language Guide  Tools  Specification  Forum

An example

```
module examples;

global protocol helloworld(role Me, role World) {
  hello(Greetings) from Me to World;
  choice at World {
    hello(GoodMorning) from World to Me;
  } or {
    hello(GoodAfternoon) from World to Me;
  }
}
```

A very simple example, but this illustrates the basic syntax for a hello world interaction, where a party performing the role Me sends a message of type Greetings to another party performing the role World, who subsequently makes a decision which determines which path of the choice will be followed, resulting in a GoodMorning or GoodAfternoon message being exchanged.
A Global Protocol

type <python> "StringType" from "Lib/types.py" as str;

global protocol Negotiation(role P, role R, role A) {
    offer(string) from P to R;
    offer(string) from R to A;
    (string) from A to R;

    rec START {
        choice at R {
            accept() from R to P;
            confirm() from P to R;
        } or {
            offer(string) from R to P;
            (conditions:string) from P to R;
            continue START;
        } or {
            reject() from R to P;
            confirm() from R to P;
        }
    }
}
Two Buyer Protocol in Scribble

module Bookstore;

type <java> "java.lang.Integer" from "rt.jar" as Integer;
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, S;
            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;
  } }
}
Global protocol well-formedness 1/2

global protocol ChoiceAmbiguous(role A, role B, role C) {
  choice at A {
    m1() from A to B; // X
    m2() from B to C;
    m3() from C to A;
  } or {
    m1() from A to B; // X
    m5() from B to C;
    m6() from C to A;
  }
}

global protocol ChoiceNotCommunicated(role A, role B, role C) {
  choice at A {
    m1() from A to B;
    m2() from B to C; // X
  } or {
    m4() from A to B;
  }
}
global protocol ParallelNotLinear(role A, role B, role C) {
    par {
        m1() from A to B;  // X
        m2() from B to C;
    } and {
        m1() from A to B;  // X
        m4() from B to C;
    }
}

global protocol RecursionNoExit(role A, role B, role C, role D) {
    rec X {
        m1() from A to B;
        continue X;
    }
    m2() from A to B;  // Unreachable for A, B
    m3() from C to D;
}
// Direct specification

global protocol P3(role C, role S1, role S2, role S3, role S4)
{
  () from C to S1;
  () from S1 to S2;
  () from S2 to S1;
  () from S1 to S3;
    () from S3 to S4;
    () from S4 to S3;
    () from S3 to S4;
    () from S4 to S3;
  () from S3 to S1;
  () from S1 to C;
}
OOI agent negotiation 1/5

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) {

}
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;
        }
    }
}
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 {
      }
    }
  }
  continue X;
}
2-level Verification

1. Writing correct global protocols with Scribble Compiler

2. Verify programs via *local monitors*
**Local Protocol Conformance**

**PROJECTION**
(At design time)

**GLOBAL PROTOCOL**

**LOCAL PROTOCOL FOR P**

**LOCAL PROTOCOL FOR R**

```plaintext
local protocol Negotiation at R(role P, role A) {
    offer(string) from P;
    offer(string) to A;
    (string) from A;
    rec START {
        choice at R {
            accept() to R;
            (conditions:string) from R;
        } or {
            offer(string) to P;
            continue START;
        } or {
            reject() to P;
            (reason:string) from R;
        }
    }
}
```

**FSM GENERATION**
(At runtime)

**FSM FOR P**

**Verification**

**PROGRAM FOR P**

**LOCAL PROTOCOL FOR A**

**FSM FOR A**

**PROGRAM FOR A**

**PROGRAM FOR R**
The Scribble Framework

- Scribble global protocols
  - Well-formedness validation
- Scribble local protocols
  - FSM generation (for endpoint monitoring)
- (Heterogeneous) endpoint programs
  - Scribble Conversation API
  - (Interoperable) Distributed Conversation Runtime
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)
FSM Generator

**Scribble:**  \( \text{Order}(x:\text{int}) \to \text{Seller} @\{x==1\} \)

**AST:**

```
[AST diagram]
```

**FSM:**

```
FSM transition table:
(1, (send, order, seller)) ->
(2, assertion_object, \{'x':'int'\})
```

```
[FSM diagram]
```

**Spec Store**

**Parser (ANTLR)**

**Tree Traversal (ANTLR)**

**FSM Store**
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
A theory for network monitoring

- Formalise MPST-monitoring and asynchronous networks.
- Introduce monitors as first-class objects in the theory.
- Justify monitoring by soundness theorems.
  - Safety
    - monitors enforces specification conformance.
  - Transparency
    - monitors does not affect correct behaviours.
  - Fidelity
    - correspondence to global types is maintained.
Multithreaded Sessions for Runtime Monitors

\[
A ::= \text{tt} \mid \text{ff} \mid e_1 = e_2 \mid e_1 < e_2 \mid \neg A \mid A_1 \land A_2 \mid A_1 \lor A_2
\]
\[
e ::= v \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \ast e_2 \mid \text{e_1 mod e_2} \quad S ::= \text{bool} \mid \text{int} \mid \text{string}
\]
\[
G ::= r_1 \rightarrow r_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I} \mid G_1 \mid G_2 \mid G_1 ; G_2 \mid \mu t.G \mid t \mid \epsilon \mid \text{end}
\]
\[
T ::= r!\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid r?\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid T_1 \mid T_2 \mid T_1 ; T_2 \mid \mu t.T \mid t \mid \epsilon \mid \text{end}
\]
\[
P ::= \overline{a}(s[r] : T) \mid a(y[r] : T).P \mid k[r_1, r_2]!l(e) \mid k[r_1, r_2]?\{l_i(x_i) . P_i\}_{i \in I} \mid
\]
if e then P else Q \mid P \mid Q \mid 0 \mid \mu X.P \mid X \mid P ; Q \mid (\nu a) P \mid (\nu s) P
\]
\[
N ::= [P]_\alpha \mid N_1|N_2 \mid 0 \mid (\nu a)N \mid (\nu s)N \mid \langle r ; h \rangle
\]
\[
r ::= a \mapsto \alpha \mid s[r] \mapsto \alpha \quad h ::= m \cdot h \mid \emptyset \mid m ::= \overline{a}(s[r] : T) \mid s\langle r_1, r_2, l(v) \rangle
\]
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.
Language and Implementations

➤ Carrying out large-scale experiences with OOI, Pivotal, Red Hat, Congnizant, UNIFI, TrustCare

➤ JBoss SCRIBBLE [ICDCIT’10, COB’12] and SAVARA projects

➤ High-performance computing

Session Java [ECOOP’08, ECOOP’10, Coordination’11]
⇒ Session C & MPI [TOOLS’12][Hearts’12][EuroMPI’12][PDP’14]

➤ Multiparty session languages

Ocaml, Java, C, Python, Scala, Jolie

➤ Trustworthy Pervasive Healthcare Services via Multiparty Session Types [FHIES’12]

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

➤ Multiparty Session Actors [Coordination’14]
Timed Multiparty Session Types based on Communicating Timed Automata

```
1  global protocol TempAvg(
2      role M, role W, role A) {
3      [x@M: x=0; reset(x)]
4      [x@W: 400 ≤ x<800]
5      msg(task) from M to W;
6      rec Loop{
7          [x@W: x ≤ 300800; reset(x)]
8          [x@M: x==300800]
9          msg(data) from W to M
10         choice at M{
11            [x@M: x==300800]
12            [x@A: 301200 ≤ x; reset(x)]
13            more(data) from M to A;
14      } [x@W: x==800; reset(x)]
15      more(task) from M to W;
16      continue Loop;
17      or{
18          [x@M: x==300800]
19          [x@A: 301200 ≤ x; reset(x)]
20          stop(data) from M to A;
21          [x@M: x==300800; reset(x)]
22          [x@W: x==800; reset(x)]
23          stop from M to W; }
24    }
25  }
```

<table>
<thead>
<tr>
<th>TGs</th>
<th>π-calculus</th>
<th>session CTAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>timed</td>
<td>type safety (Thm 4.3), error-freedom (Thm 4.4)</td>
<td>Sound, complete characterisation, (Thm 5.6)</td>
</tr>
<tr>
<td>feasible, wait-free</td>
<td>progress (Thm 5.4, Thm 5.5)</td>
<td>progress (Thm 5.7)</td>
</tr>
</tbody>
</table>
Zero Deviation Life Cycle Platform

System Behaviour
- JVM Logs, Application Logs
- System logs, DB Logs

Adapters
- C/C++
- JAVA
- MAINFRAME
- .NET
- IBM BPM
- COBOL
- TANDEM
- ORACLE DB
- MQs

Reverse Engineering Core Module
- Configuration Matrix
- Crawler
- Generic Parser
- Analyser

Report Composer

SCRIBBLE

UML & BPMN2 Model
Synthesis of Graphical Choreographies 1/2

- Multiparty Session Types top-down approach (cf. POPL’08 & ESOP’12)
- Not applicable without *a priori knowledge* of a choreography
- Synthesise a choreography from a set of local specifications
- Concretely: from *Communicating Finite-State Machines to Global Graphs*
Synthesis of Graphical Choreographies 2/2

Alice

Bob

Carol

Dave
Graphical global specification based on Petri Nets that cannot be directly represented in the MPST linear syntax

An application of the Petri Nets token dynamics to a conformance validation
Session Nets 2/2

\[ g = \{\text{a} \mapsto \text{Planned, b} \mapsto \text{Order, c} \mapsto \text{OrderEnd, d} \mapsto \text{Checkpoint, e} \mapsto \text{Provide, f} \mapsto \text{Deliver, g} \mapsto \text{Provide, h} \mapsto \text{Update}_1, i \mapsto \text{Provide, j} \mapsto \text{Deliver, k} \mapsto \text{Provide, l} \mapsto \text{Update}_2, m \mapsto \text{PO, n} \mapsto \text{POAck, o} \mapsto \text{PO, p} \mapsto \text{Accept}_1, q \mapsto \text{Confirmation, r} \mapsto \text{Retailer, s} \mapsto \text{Accept}_2, t \mapsto \text{Finalized, u} \mapsto \text{ProvideEnd}\} \]
Session Type Projects

- EPSRC Conversation-Based Governance for Distributed Systems by Multiparty Session Types
- SADEA EPSRC Exploiting Parallelism through Type Transformations for Hybrid Manycore Systems, with Vanderbauwhede, Scholz, Gay and Luk
- Programme Grant From Data Types to Session Types: A Basis for Concurrency and Distribution, with Wadler and Gay
- EU FP7 FETOOpenX UpScale with de Boer (CWI), Clark, Wrigstad (Uppsala), Johnsen (Oslo) and Drossopoulou
- Pivotal Dynamic Assurance based on Multiparty Session Types
- Cognizant/Qualit-e EPSRC Knowledge Transfer Secondments
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.*
Multiparty Session Type Theory

- Multiparty Asynchronous Session Types [POPL’08]
- 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]
  - Buffered Communication Analysis in Distributed Multiparty Sessions [CONCUR’10]
Logics

- Design-by-Contract for Distributed Multiparty Interactions [CONCUR’10]
- Specifying Stateful Asynchronous Properties for Distributed Programs [CONCUR’12]
- Multiparty, Multi-session Logic [TGC’12]

Extensions of Multiparty Session Types

- Multiparty Symmetric Sum Types [Express’10]
- Parameterised Multiparty Session Types [FoSSaCs’10, LMCS]
- Global Escape in Multiparty Sessions [FSTTCS’10]
- Dynamic Multirole Session Types [POPL’11]
- Nested Multiparty Sessions [CONCUR’12]
Dynamic Monitoring
- Asynchronous Distributed Monitoring for Multiparty Session Enforcement [TGC’11]
- Monitoring Networks through Multiparty Sessions [FORTE’13]

Automata Theories
- Multiparty Session Automata [ESOP’12]
- Synthesis in Communicating Automata [ICALP’13]

Typed Behavioural Theories
- Governed Session Semantics [CONCUR’13]

Choreography Languages
- Compositional Choreographies [CONCUR’13]