Session TYPES AS A Descriptive Tool for Distributed PROTOCOL

Nobuko Yoshida & Raymond Hu

25th Oct 2017
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 Pawliska
2013 winner (Undergraduate) source: QMUL

Mr. Valmir Negacevshi
2013 winner (Postgraduate) source: QMUL
Programming languages are tools which offer frameworks of abstraction for such activities – promoting or limiting them

- Imperative
- Functional
- Logical
• The most fundamental element of a PL in this context is a set of operations it is based on:
  Imperative: assignment, jump.
  Functional: β-reduction.
  Logical: unification.

• Another element is how we can combine, on structure, these operations:
  Imperative: sequential composition, if-then-else, while, procedures, modules, ... 
  Functional: application, product, union, recursion, modules, ...
Var $a$: array[1..MAX] of int;

Procedure sort($l, r$: int);

Var $i$, $j$, $x$: int;

$i := l$; $j := r$;

$x := \lfloor (l+r)/2 \rfloor$; \hspace{1cm} * Choose a pivot.*

repeat

while $a[i] < x$ do $i := i+1$ end \hspace{1cm} * Partition into two parts.*

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 $l < j$ then sort($l, j$); \hspace{1cm} * Recursively sort two parts.*

if $l > r$ then sort($i, r$);

end

Procedure swap($i, j$: int)

Var $w$: int;

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

Quicksort with combinators:
Quicksort in ML:

```ml
fun qs nil:int list = nil
  | qs (x::r) = let val small = filter (fn y => y<x) r
     and large = filter (fn y => y>=x) r
     in qs small@[x]@[qs large] end

fun filter p nil = nil
  | filter p (x::r) = if p x then x::filter p r
                        else filter p r
```
The π-calculus as a Descriptive Tool

\[ M ::= x \mid xx.M \mid MN. \]

\[ P ::= \Sigma_{x}.P \mid PIQ \mid \omega P \mid !P \mid \emptyset. \]

with \[ \Pi ::= x(\emptyset) \mid x(\emptyset). \]

\[ \boxed{\lambda \in \Pi} \]

\[ [x]_u = x(u). \]

\[ [x.M]_u = u \cdot u. [M]_u. \]

\[ ([M]_u) = (vF) (\exists x) v. [M]_v \mid (x = N) \]

\[ \text{with } [x = N] \equiv !x(u). [N]_u. \]
Examples of Representable Computation

- λ-calculus [NPW89, Milner89, Milner82, ...]
- Concurrent Object [Walker81]
- Co-order term passing [Sangi92]
- Various data structures [Milner92, ...]
- Proof Nets [Horn and Scott 85]
- Arbitrary "context" interaction [Hi1984]
- Strategies on Games [H095]
The Role of Types in TrCalculus

- (classification) How can we classify name-passing interactive behaviours, i.e. behaviours representable in TrCalculus? 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

\[
\begin{align*}
\text{Nat} \Rightarrow \text{Nat} & \quad \text{Succ, Div, Id, ...} \\
\text{Nat} \Rightarrow \text{Bool} & \quad \text{Iszero, ...} \\
\text{Nat} \Rightarrow \text{Nat} \Rightarrow \text{Nat} & \quad \text{Add, Sub, Mult, Div, Exp, ...} \\
\text{Bool} \Rightarrow \text{Bool} & \quad \text{And, Or, Nor, ...} \\
\text{Bool} & \quad \text{True, False}
\end{align*}
\]

with operation:

\[
\begin{align*}
\text{if} \beta \in \Theta \quad \text{then } \text{e;d} = f \cdot \beta \cdot \beta \\
\text{else undefined}
\end{align*}
\]
• When it comes to processes, composition becomes:

\[ 3 + 5 \]

\[ \Rightarrow \]

\[ 8 \]


• But some composition is dangerous!

• Therefore we type processes,

- divergence
- deadlock
- run-time error

The connection is prohibited.
Implementing ATM

- Put the card in.
- Ask what you want.
- Select, for example, withdraw, then key in the amount.
- Banking system.
- Ok, take your money.
- Overdraft! Try again.
Implementing ATM

ENCODING

\[ \text{ATM(c,b)} = \]

\[ x \cdot (\text{cb}) : = ? \times \text{id} \; | \; [? \text{wd}; ? \times] \]

\[ \begin{array}{l}
\text{b} \cdot (\text{cb}) : = \text{bu} \; \text{at} ; \; \text{id} ; \; ? \times \text{cb}
\end{array} \]

\[ \begin{array}{l}
[? \text{ok}:
\text{b} \cdot \text{cb} ; \; ? \times ; \; \text{ATM} \cdot \text{cb}
\end{array} \]

\[ \begin{array}{l}
? \text{ordr} :: = +
\text{b} \cdot \text{ordr} ; \; \text{ATM} \cdot \text{cb}
\end{array} \]

\[ \begin{array}{l}
? \text{bcl} :: = \text{bu} \; \text{tbcl} ; \; ? \times
\end{array} \]

\[ \begin{array}{l}
\text{b} \cdot (\text{cb}) : = \text{bu} \; \text{tbcl} ; \; ? \times
\end{array} \]

\[ \begin{array}{l}
\text{b} \cdot \text{cb} ; \; ? \times ; \; \text{ATM} \cdot \text{cb}
\end{array} \]
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
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:

```python
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 $\pi^4$ Technology

↓

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

- Buyer
  - title
  - quote
  - address
  - data
  - retry

- Seller
  - ok
  - repeat

(choice)
Binary Session Types: Buyer - Seller Protocol

Buyer

- title
- quote

Seller

- ok
- address

choice

- data
- retry

\(mt! \text{Title; } ? \text{Quote}; !\{ \text{ok; !Add; } ? \text{Date}, \text{retry: t} \}\)
Buyer

choice

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>quote</td>
</tr>
<tr>
<td>ok</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>retry</td>
</tr>
</tbody>
</table>

Seller

P has T
Q has T

P | Q typable

mut! Title; ? Quote; ![ ok: ! Add; ? Date, retry: t ]
mut? Title; ! Quote; ? { ok: ? Add; ! Date, retry: t }
Multiparty Session Types

Buyer 1: title, quote, quote ÷ 2
Seller: quote
Buyer 2: address, date

Acknowledgment: ok
Multiparty Session Types

Buyer 1

Seller

Buyer 2

title

quote

quote \div 2

ok

title

quote

address

date
If you use binary Session Types ....
Multiparty Session Types

[Honda, Yoshida, Carbone 2008]

\[
B_1 \rightarrow S \quad \text{Int.} \\
S \rightarrow B_2 \quad \text{Char}
\]

STEP 1
Write Global Type
Multiparty Session Types

[B1 → S] Int.

[S → B2] Char

STEP 1
Write Global Type

Bi!Int. B2! Char

STEP 2
Project to Local Types

T1, T2, T3

project
Multiparty Session Types [Honda, Yoshida, Carbone 2008]

**STEP 1**
Write Global Type

**STEP 2**
Project to Local Type

**STEP 3**
- Static Check
- Generate Code
- Run-time check
Alice: $AB!a; CA?c$

Bob: $AB?a; BC!b$

Carol: $BC?b; CA!c$

NO Deadlock

Global Type

Projection

LOCAL TYPES
Selected Publications 2016/2017

- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornella 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.
Relationship with Communicating Automata

ESOP 12
ICAALP 13
CONCUR 15

Characterisation
Synthesis
Timed Automata

POPL 15
TACAS 16
FOSSACS 17

Graphical
Synthesis
Subtyping
Model-checking
Undecidability
mut! Title; ? Quote; ![ok!: !Add; ? Date, retry: t]

mut? Title; ! Quote; ?{ [ok: ?Add; ! Date, retry: t]
Communicating Automata [1980s]
1. Deterministic
2. No-Mixed State
3. Compatible

[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.
Typed Semantics in \( \pi \) 1991 →

IO-subtyping, Linear types, Secure Information Flow, …

\[ \Gamma \vdash P \preceq Q \]

- Correctness of Encoding
- Limit environment
  \[ \Rightarrow \text{Equate more processes} \]
- Compositional
GLOBALLY GOVERNMED
SESSION SEMANTICS

CONCUR 15
LMCS

Dimitrios Kousapas
Glasgow

Nobuko Yoshida
Imperial College London
Use case: UC.R2.13 “Acquire Data from Instrument” from Ocean Observatories Initiative (OOI)

Instrument  Agent  User

I  A1  A2  U

I  A1  A2  U

A1 → U: <pd>.
A2 → U: <pd>.
end
Use case: UC.R2.13 "Acquire Data from Instrument" from Ocean Observatories Initiative (OOI)

Instrument → Agent → User

A1 → U: ⟨pd⟩.
A2 → U: ⟨pd⟩. end

I A1 A2 U

I A1 A2 U
HOTπ and TYPES

CONCUR 16 Characteristic Bisimulations
ESOP 16 Expressiveness
ACTA INFORMATICA
Interconnectability of Session Logical Processes

Multi Party

Linear Logical Session

Interconnectability of Session Logical Processes

Multi Party

CONCUR’15
Multiparty Session Linear Logic
Acta Informatica

Linear Logical Session

Cf. [1995] Specification Structures and Proposition as Type
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.

**Describe 📝**
Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

**Verify 👍**
Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

**Project 🎨**
Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

**Implement 🌐**
Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

**Monitor 📈**
Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.
module examples;

// protocol

// hello() from Me to World;

// choice at World {
//   goodMorning1() from World to Me;
// }
// or {
//   goodMorning1() from World to Me;
// }

• **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
Dynamic Monitoring
[RV’13, COORDINATION’14, FMSD’15, LMCS’17, CC’17]
Type Checking

[OOPSLA’15, ECOOP’16, ECOOP’17, COORDINATION’17]
Code Generation
[CC’15, FASE’16, FASE’17]
Synthesis

[ICALP’13, POPL’15, CONCUR’15, TACAS’16, CC’16]
Session Type based Tools

OOI Governance

ZDLC: Process Modeling

Actor Verification

MPI code generations
Interactions with Industries

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

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

Nobuko Yoshida
Imperial College, London

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

Static deadlock detector
Tool developed based on our research:
github.com/programminghustler
- Static (compile-time) detection of deadlock
- Help prevent deadlocks
- Ongoing research
- Analyzed common concurrency patterns & open source projects

@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...

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)
Java API Generation [FASE’16]
Scribble – Proving a distributed design

1. All design work takes place in ABACUS, DCC’s enterprise architecture tool. This can export standard XMI files (an open standard for UML5)

2. XMI is converted into OpenTracing format for consumption by managed service

3. OpenTracing files are combined to build a model in Scribble

4. Model holds types rather than instances to understand behaviour

5. Scribble compiler identifies inconsistency, change & design flaws

6. Issues highlighted graphically in Eclipse

7. Generate exception report and send back to DCC