Idioms for Interaction: Functional Types, Process Types and Distributed Systems

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

Nobuko Yoshida
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 Vladimir 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

Programs: prescription of computational behaviours based on a certain abstraction.
• 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, on structure, these operations:

  Imperative: sequential composition, if-then-else, while, procedures, modules, ....
  Functional: application, product, union, recursion, modules, ....
Var a: array[MAX] of int;

Procedure sort(l, r: int);
Var i, j, x: int;

i := l; j := r;
x := (l + r) \div 2;  // Choose a pivot.

repeat
  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 l < j then sort(i, r); end
  sort two parts.
end

Procedure swap(i, j: int)
Var w: int;
w := a[i]; a[i] := a[j]; a[j] := w
end
Quicksort in pure lambda:

Quicksort with combinators:
Quicksort in 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 Pi-calculus as a Descriptive Tool

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

\[ P ::= \Sigma \pi_1.P \mid \Pi Q \mid \omega P \mid !P \mid \emptyset. \]

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

\[ x \in \Pi \]

\[ [x]_\text{u} \equiv x(w). \]

\[ [\lambda x.M]_\text{u} \equiv u \otimes u. [M]_\text{u}. \]

\[ [(MN)]_\text{u} \equiv (\forall x)([M]_\text{y} \otimes u \ominus \{x\} \downarrow \{x=N\}) \]

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

- λ-calculus [NPW89, Milner92, Milner82, ...]
- Concurrent Object [Walker81]
- Co-ordinary term passing [Sezgin92]
- Various data structures [Milner92, ...]
- Proof Nets [Alonzo and Scott83]
- Arbitrary 'constant' interaction [HRS4]
- Strategies on Games [H095]
The Role of Types in π-calculus.

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

with operation:

\[ f : \beta \mapsto \theta \cdot e : d = f \cdot e : \beta. \]

else undefined.
When it comes to processes, composition becomes:

\[
3 + 5 = 8
\]

But some composition is dangerous!

Therefore we type processes, the connection is prohibited.

- divergence
- deadlock
- runtime error
Explosion!
Implementing ATM

1. Insert card
   - put the card in
2. Ask what to do:
   - withdraw?
   - balance?
   - deposit?
   - quit?
3. Select action:
   - withdraw, then key in the amount
   - 'Banking System'
4. Process:
   - 'Ok; take your money'
   - 'Overdraft! Try again.'
Implementing ATM

\[ \text{ATM}(cb) = \]
\[ \begin{align*}
& \text{encode}(x) = \text{encode}(\text{id}) \\
& \text{bank}(\text{id}) = \text{encode}(\text{id})
\end{align*} \]
Current: Communication is Ubiquitous

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?
Current: Communication is Ubiquitous

➤ The way to organise software is increasingly based on communications (Cloud Computing, many cores, message-passing parallel computation, ...)

➤ Question ⇒ Multiparty session type theory

➤ 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
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]

\[ \downarrow \]

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

\[ \downarrow \]

Formalisation of W3C WS-CDL [ESOP’07]

\[ \downarrow \]

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 (WS-CLs) 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
From: Robin Milner

Date: Wed, February 11, 2004 1:02 pm

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:

```java
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:

```plaintext
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 *π*\(^4\) 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
  - OK
  - address
  - date
  - quit

- Seller

branch
! String ; ? Int ; ⊕ { OK : ! String ; ? Date ; end, quit : end }
Branch

Buyer

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

Seller

P has T
Q has \( \overline{T} \)\text{ dual}
PQ typable

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

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

Buyer 1  Seller  Buyer 2

title
quote
quote ÷ 2

quote

ok
address
date
Multiparty Session Types

Buyer 1, Seller, Buyer 2

title
quote
quote / 2
quote
address
date
ok
Session Types Overview

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]
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
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.

An example

```scribble
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.

Find out more ...

Language Guide  Tools  Specification  Forum
Two Buyer Protocol in Scribble

```plaintext
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;
    } }
}
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 {
            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
Welcome to Release 2 of the Ocean Observatories Initiative Observatory (OOI). You already have access to many OOI features and real-time data. Just click on something that looks interesting on this page to start using the OOI as our guest.

For personalized services, such as setting up notifications and preserving settings for your next visit, create a free account by clicking on "Create Account" at the top of the page.

DATA LEGEND
- Temperature
- Salinity
- Oxygen
- Density
- Currents
- Sea Surface Height (SSH)
- Chlorophyll
- Turbidity
- pH
- Autonomy
- Other

RECENT UPDATES
- 03 m Oregon Coast North Salinity
- 02 m California South 160m pH
- 01 m California South Salinity
- 03 m Oregon North Turbidity
- 02 m Oregon Coast Temperature
- 01 m Oregon Coast Currents
- 01 m California South Salinity
- 01 m Oregon Coast 100m pH
- 01 m California South Turbidity
- 01 m California North Temperature
- 02 m Oregon Coast Salinity

Note: Please check the website for more details and updates.
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]
  - 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]

Timed Multiparty Session Types [CONCUR’14]
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]

Petri Nets
- Multiparty Session Nets [TGC’14]

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

Choreography Languages
- Compositional Choreographies [CONCUR’13]
**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]
http://www.zdlc.co/faq/

WHAT DOES ZDLC DO?

Professor Steve Ross-Talbot
Managing Director, ZDLC BU
Cognizant Technical Services
Zero Deviation Life Cycle Platform
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 representations of choreographies involving interactions and states with transitions such as 'A→C:cwin', 'A→B:bwin', 'C→B:blose', 'B→C:close', 'B→A:sig', and 'A→C:score'.
Session Nets 1/2

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

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