Open Problems and State-of-Art of Session Types

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

Nobuko Yoshida
Imperial College London
Questions on Mobile Processes

➤ How to apply mobile process theories to real distributed and concurrent applications and programming languages?

Common Questions on Session (Protocol) Types

➤ I wish to extend (Multiparty) Session Types to XXX.
➤ I wish to learn (Multiparty) Session Types. What is the best paper?
➤ (Multiparty) Session Types can verify/specify XXX?
➤ What is a relationship between (Multiparty) Session Types and MSCs/Petri Nets/State Machines/...?
Usage on Multiparty Session Types

➤ Static Type Checking via End Point Projection
   ⇒ Java, C, Haskell, Ocaml

➤ Dynamic Type Checking
   ⇒ Runtime Monitoring, Python, Erlang, Java, ...

➤ Synthesis
   ⇒ Generating BPMN Choreographies and Legacy Code Analysis

➤ Code Generation
   ⇒ MPI Parallel Programming
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 \cite{PARL'94, ESOP'98}
\[\downarrow\]
Milner, Honda and Yoshida joined W3C WS-CDL (2002)
\[\downarrow\]
Formalisation of W3C WS-CDL \cite{ESOP’07}
\[\downarrow\]
Scribble at $\pi^4$ Technology
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:

```prolog
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
- 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 $T$

dual

PIQ typable

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

dual ? String ; ! Int ; 8 { ok : ? String ; ! Date ; end, quit : end }
Multiparty Session Types

Buyer 1

- title
- quote
- quote ÷ 2

Seller

- quote
- ok
- address
- date

Buyer 2
Multiparty Session Types

Buyer 1

Buyer 2

Seller

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]

- Distributed Runtime Verification with Session Types and Python [FMSD’15]
- Multiparty Session Actors [COORDINATION’14]
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.

Find out more ...

Language Guide  Tools  Specification  Forum

An example

```plaintext
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.
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) {

type 
> "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;
        } }
} }

// 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;
        } }
} }

// 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;
        } }
} }

// 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;
        } }
} }

// 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;
        } }
} }

// 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;
        } }
} }

// Local protocol projection (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

Scribble online checker: scribble.doc.ic.ac.uk
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.
http://www.zdlc.co/faq/
Zero Deviation Life Cycle Platform
From Communicating Machines to Graphical Choreographies [POPL’15]

[ESOP’10, ESOP’12, CONCUR’12, CONCUR’14]
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
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>Repository</th>
<th>Berkeley HPC Dwarfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>heateq</td>
<td>stencil*</td>
</tr>
<tr>
<td>nbody</td>
<td>ring*</td>
</tr>
<tr>
<td>wordcount</td>
<td>scatter-gather*</td>
</tr>
<tr>
<td>adpredictor</td>
<td>scatter-gather*</td>
</tr>
<tr>
<td>montecarlo</td>
<td>scatter-gather*</td>
</tr>
<tr>
<td>montecarlo-mw</td>
<td>master-worker*</td>
</tr>
<tr>
<td>LEsolver</td>
<td>wraparound mesh</td>
</tr>
<tr>
<td>matvec</td>
<td>custom</td>
</tr>
<tr>
<td>fft64</td>
<td>6-step butterfly</td>
</tr>
</tbody>
</table>

π
Evaluation

Productivity: Effort

Protocols in repository
- Use backbone directly
- Write kernel
- Effort = $K / B + K$

Custom protocols
- Write Pabble protocol
- Tool generate backbone
- Write kernel
- Effort = $P + K / B + K$

<table>
<thead>
<tr>
<th>Program</th>
<th>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
$\pi$ LOC savings
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.