Multiparty Session Types and their Applications

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

Nobuko Yoshida, Rumyana Neykova
Imperial College London
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?
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
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:

```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 π⁴ Technology

↓

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

Binary Session Types \textit{[PARL’94, ESOP’98]}

\quad \Downarrow

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

\quad \Downarrow

Formalisation of W3C WS-CDL \textit{[ESOP’07]}

\quad \Downarrow

Scribble at $\pi^4$ Technology

\quad \Downarrow

Multiparty Session Types \textit{[POPL’08]}
Binary Session Types: Buyer-Seller Protocol

Buyer

quote

OK

address

date

quit

Seller
! String ; ? Int ; ⊕ { OK : ! String ; ? Date ; end, quit : end }
```
! 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 ➔ ok ➔ date
Multiparty Session Types

Buyer 1 → Seller → Buyer 2

- title
- quote
- quote ÷ 2
- quote
- 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]

Projection

Global Type

Local Type

Program Alice

Local Type

Program Bob

Local Type

Program Carol
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;
    } } }
OOI agent negotiation 1/5

```
Consumer Agent

- Negotiation starting by a Consumer making a proposal, then accepted by Provider and confirmed by Consumer
  - negotiate: request(SAP_1)
  - negotiate: accept(SAP_1, details)
  - negotiate: confirm(SAP_1)

- Negotiation starting by the Provider inviting a Consumer with a proposal, accepted by Consumer and confirmed by Provider
  - negotiate: invite(SAP_1)
  - negotiate: accept(SAP_1, details)
  - negotiate: confirm(SAP_1)

- Negotiation starting by a Consumer making a proposal. The recipient (Provider) makes a counter-proposal, supplanting SAP_1, which is then accepted by Consumer and confirmed by the Provider.
  - negotiate: request(SAP_1)
  - negotiate: counter-propose(SAP_2)
  - negotiate: accept(SAP_2, details)
  - negotiate: confirm(SAP_2)

- Negotiation starting by a Consumer making a proposal, rejected by the Provider ending the Negotiation.
  - negotiate: request(SAP_1)
  - negotiate: reject(SAP_1)

Provider Agent

- Confirm is the complementary accept by the other party (both must accept for an agreement).

- With a mutual accept, at least one commitment on each side of the conversation results (may be multiple). The contract is as stated in the most recent SAP.

- A counter-propose is a new SAP, but it typically refines or partially modifies the prior SAP.

- Any party can reject instead of counter-propose (or accept)

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;
  }

  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;
  }

  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;
        }
    }
}
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

ZDLC: Process Modeling

Protocol Verification

MPI code generations
Dynamic Monitoring

[RV’13, COORDINATION’14, FMSD’15]

Global Type

Projection

Local Type

Dynamic Monitoring

Program Alice

Local Type

Dynamic Monitoring

Program Bob

Local Type

Dynamic Monitoring

Program Carol
Type Checking \cite{OOPSLA'15, POPL'16}

- **Global Type**
  - **Projection**
    - **Local Type**
      - Program Alice
      - Type Checking
    - **Local Type**
      - Program Bob
      - Type Checking
    - **Local Type**
      - Program Carol
      - Type Checking
Code Generation [CC’15, FASE’16]

Global Type

Projection

Local Type

Generation

Local Type

Generation

Local Type

Generation

Program Alice

Program Bob

Program Carol
Synthesis

[ICALP'13, POPL'15, CONCUR'15, TACAS'16]
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/

WHAT DOES ZDLC DO?

Professor Steve Ross-Talbot
Managing Director, ZDLC BU
Cognizant Technical Services
Zero Deviation Life Cycle Platform
Message Passing Programming
[CC’15, OOPSLA’15]

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
Example: Simple search engine

Scatter-Gather protocol

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

```pi
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 Reduce
  bufReduce1_s = pabble_sendq_dequeue();
  MPI_Gather( ..., MPI_T, Worker_RANK(1) ... );
  free(bufReduce1_s);
  pabble_recvq_enqueue(Reduce, bufReduce1_r);
#pragma pabble Finish

typedef char *S; MPI_Datatype MPI_S = MPI_CHAR;

typedef char *T; MPI_Datatype MPI_T = MPI_CHAR;

load_data();
bufMap0_r = calloc(meta.buflen(Map), sizeof(S));
distribute_data();
bufMap0_s = pabble_sendq_dequeue();
MPI_Scatter( ..., MPI_S, Worker_RANK(1), ... );
free(bufMap0_s);
  pabble_recvq_enqueue(Map, bufMap0_r);
distribute_data();
bufReduce1_r = calloc(meta.buflen(Reduce)*meta.nprocs, sizeof(T));
collect_records();
bufReduce1_s = pabble_sendq_dequeue();
MPI_Gather( ..., MPI_T, Worker_RANK(1) ... );
free(bufReduce1_s);
  pabble_recvq_enqueue(Reduce, bufReduce1_r);
collect_records();
display_cleanup();
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>

http://mrg.doc.ic.ac.uk
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></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>15</td>
<td>154</td>
<td>335</td>
<td>0.69</td>
</tr>
<tr>
<td>nbody</td>
<td>15</td>
<td>93</td>
<td>228</td>
<td>0.71</td>
</tr>
<tr>
<td>wordcount</td>
<td>8</td>
<td>76</td>
<td>176</td>
<td>0.70</td>
</tr>
<tr>
<td>adpredictor</td>
<td>8</td>
<td>76</td>
<td>182</td>
<td>0.71</td>
</tr>
<tr>
<td>montecarlo</td>
<td>8</td>
<td>76</td>
<td>70</td>
<td>0.48</td>
</tr>
<tr>
<td>montecarlo-mw</td>
<td>10</td>
<td>82</td>
<td>70</td>
<td>0.46</td>
</tr>
<tr>
<td>LEsolver</td>
<td>15</td>
<td>132</td>
<td>208</td>
<td>0.66</td>
</tr>
<tr>
<td>matvec</td>
<td>15</td>
<td>130</td>
<td>117</td>
<td>0.41</td>
</tr>
<tr>
<td>fft64</td>
<td>11</td>
<td>64</td>
<td>134</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Effort ratio
\[ \pi = \text{LOC savings} \]
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 Optimisation 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.*