PRINCIPLES AND PRACTICE OF SESSION TYPES

Vasco T. Vasconcelos, University of Lisbon
and
Raymond Hu, Imperial College London

A tutorial at POPL
20 January 2014

OUTLINE

- Part I _ Fundamentals of session types, by Vasco
- Part II _ Specification and verification of distributed applications using multiparty session types, by Ray
- Grab these slides from http://www.doc.ic.ac.uk/~rhu/popl14tutorial.pdf

MOTIVATION _ ITERATOR

- Met java.util.Iterator?

```
interface Iterator {
    boolean hasNext ();
    Object next ();
    void remove ();
}
```

COMMON MISTAKES

```
void commaSeparatedList (Iterator it) {
   System.out.print(it.next());
   while (it.hasNext())
      System.out.print(", " + it.next()); }
```

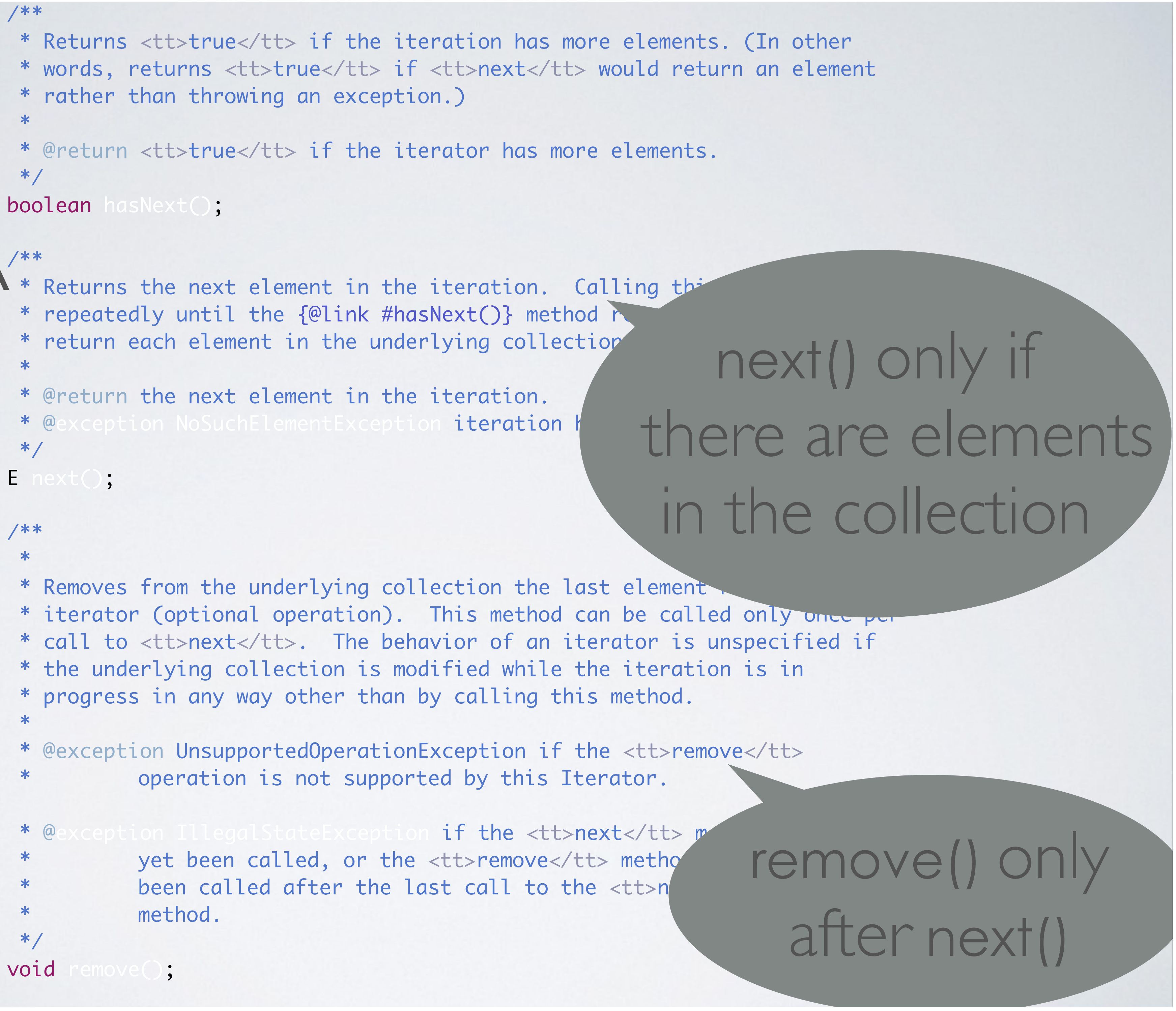
```
void filter (Iterator it, Object o) {
   while (it.hasNext())
      if (it.next().equals(o))
         System.out.print(it.next()); }
```

```
void removeFirst (Iterator it) {
   if (it.hasNext())
      it.remove(); }
```

COMPILE AND RUN

- This code compiles...
- ... and sometimes even runs
- To “correctly” use the iterator one must read the documentation

THE ITERATOR DOCUMENTATION



```
    /**
     * Returns <tt>true</tt> if the iteration has more elements. (In other
     * words, returns <tt>true</tt> if <tt>next</tt> would return an element
     * rather than throwing an exception.)
     *
     * @return <tt>true</tt> if the iterator has more elements.
     */
    boolean hasNext();

    /**
     * Returns the next element in the iteration.  Calling thi
     * repeatedly until the {@link #hasNext()} method r
     * return each element in the underlying collection
     *
     * @return the next element in the iteration.
     * @exception NoSuchElementException iteration h
     */
    E next();

    /**
     *
     * Removes from the underlying collection the last element r
     * iterator (optional operation).  This method can be called only once per
     * call to <tt>next</tt>.  The behavior of an iterator is unspecified if
     * the underlying collection is modified while the iteration is in
     * progress in any way other than by calling this method.
     *
     * @exception UnsupportedOperationException if the <tt>remove</tt>
     *            operation is not supported by this Iterator.

     * @exception IllegalStateException if the <tt>next</tt> m
     *            yet been called, or the <tt>remove</tt> metho
     *            been called after the last call to the <tt>n
     *            method.
     */
    void remove();
}
```

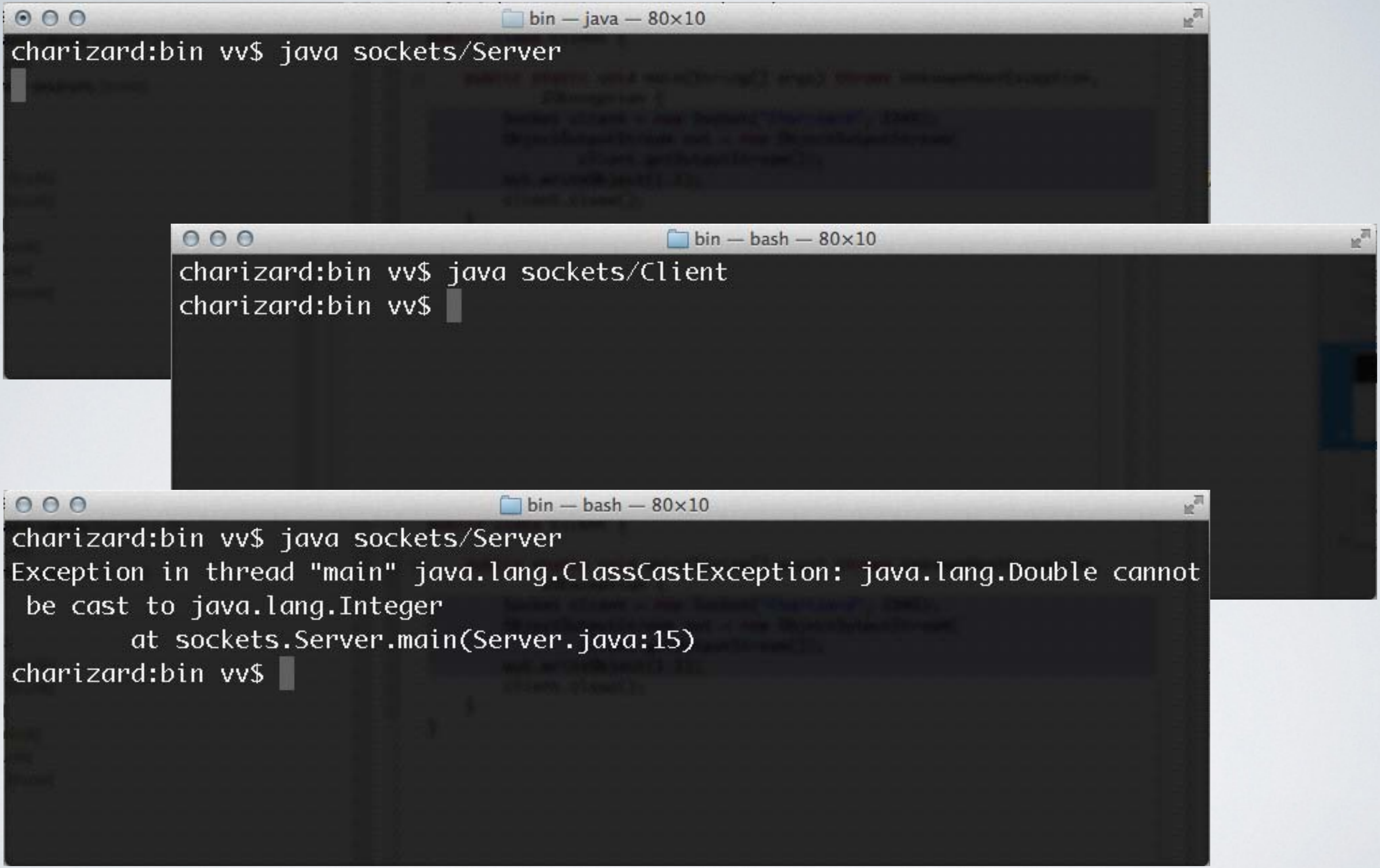
next() only if there are elements in the collection

remove() only after next()

SOCKET COMMUNICATION

```
Socket client = new Socket("Charizard", 2345);
ObjectOutputStream out = new ObjectOutputStream(
        client.getOutputStream());
out.writeObject(1.1);
```

```
ServerSocket serverSocket = new ServerSocket(2345);
Socket server = serverSocket.accept();
ObjectInputStream in = new
    ObjectInputStream(server.getInputStream());
Integer i = (Integer) in.readObject();
```



```
charizard:bin vv$ java sockets/Server
```

```
charizard:bin vv$ java sockets/Client
charizard:bin vv$ 
```

```
charizard:bin vv$ java sockets/Server
Exception in thread "main" java.lang.ClassCastException: java.lang.Double cannot
 be cast to java.lang.Integer
        at sockets.Server.main(Server.java:15)
charizard:bin vv$ 
```

WOULDN'T IT BE NICE …

- … to program in a language that makes NoSuchElementException, IllegalStateException, ClassCastException unnecessary?
- We need more expressive types...

WHAT WE REALLY NEED

- Abstractions that allow to talk about continuous interactions

- Languages and compilers that make sure code follows the abstractions

SESSION TYPES TO THE RESCUE

- Introduced by Kohei Honda *et alia* in 1994-98 (see further reading)

- Abstract series of continuous interactions; abstract communication protocols

- Originally associated to the pi-calculus; later transposed to functional and OO languages

RUNNING EXAMPLE _ AN ONLINE DONATION SERVICE

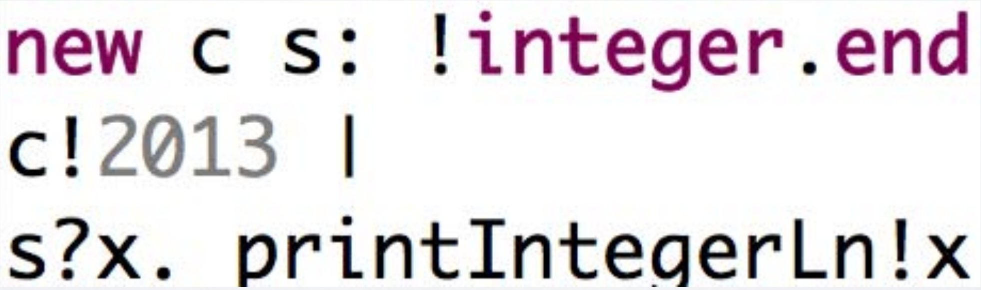
Three sorts of participants: server, clients, and benefactors

- Clients create donation campaigns and send the campaign link to benefactors
- Benefactors donate by providing a credit card number and an amount to be charged to the card
- The server provides for the creation of campaigns and forwards the donations to the bank

DEMO

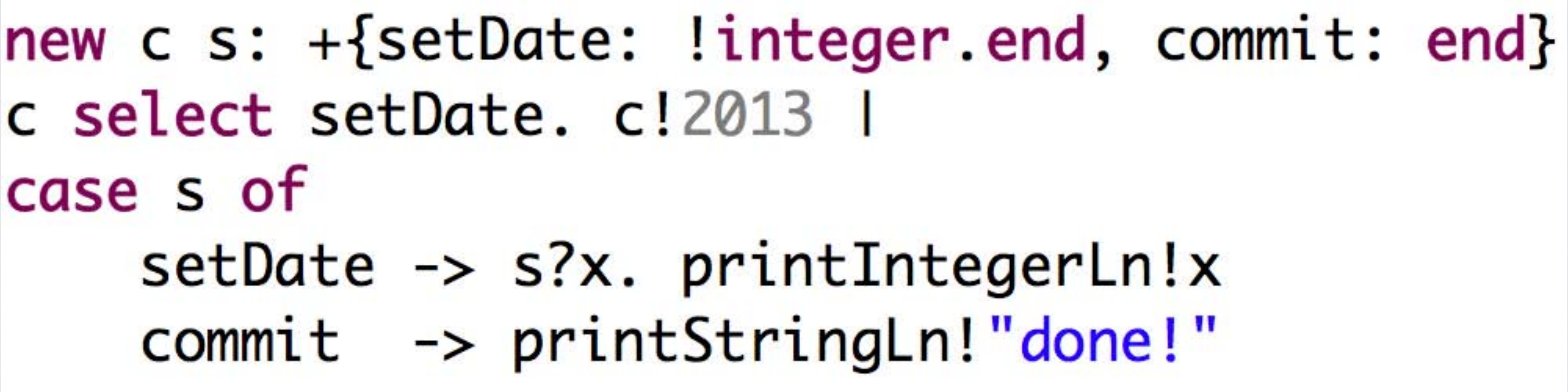
- Based on SePi, Sessions on Pi, http://gloss.di.fc.ul.pt/sepi/
- A pi-calculus based language with (linearly refined) session types
- We introduce the various basic type and process constructors

V1 _ CHANNEL CREATION, INPUT, OUTPUT, PARALLEL COMPOSITION



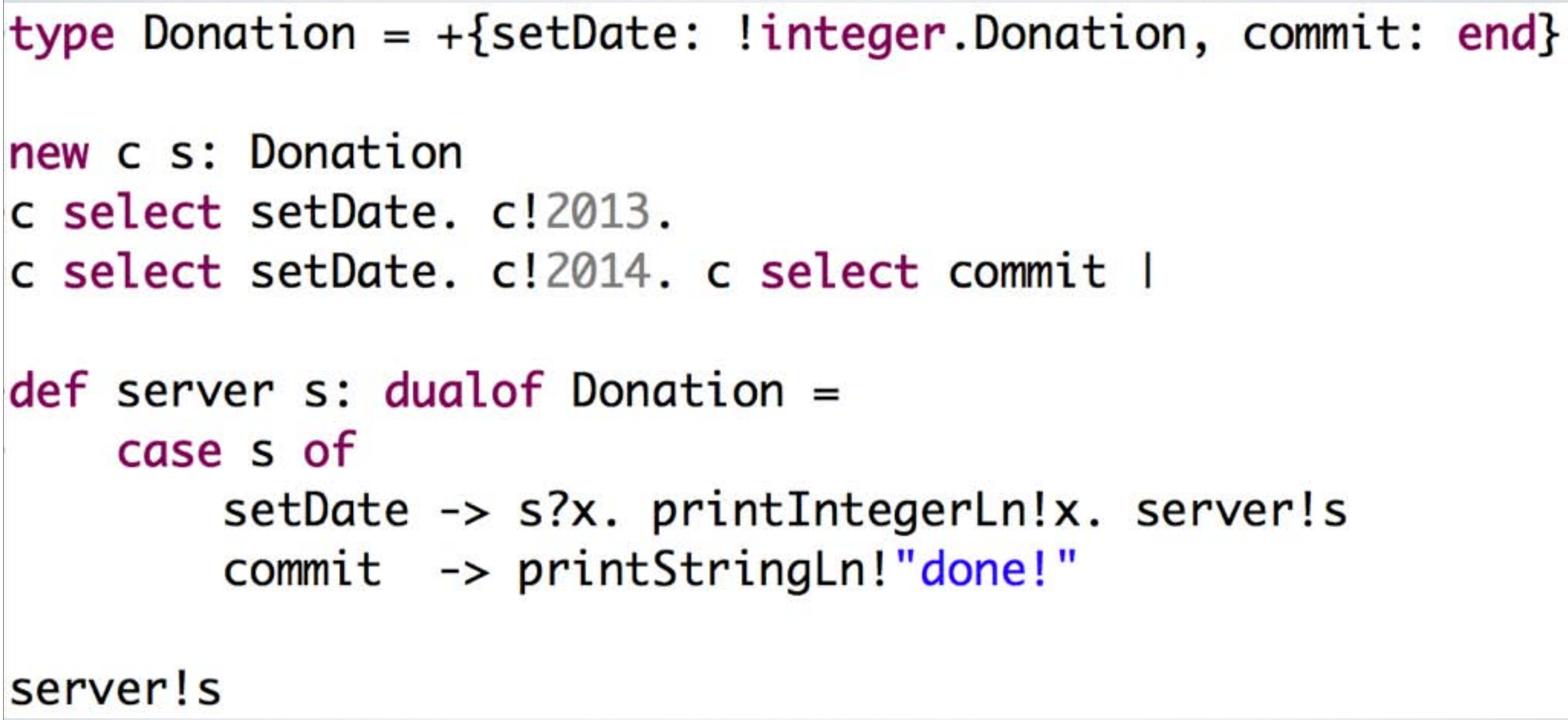
```
new c s: !integer.end
c!2013 |
s?x. printIntegerLn!x
```

V2 _ CHOICE



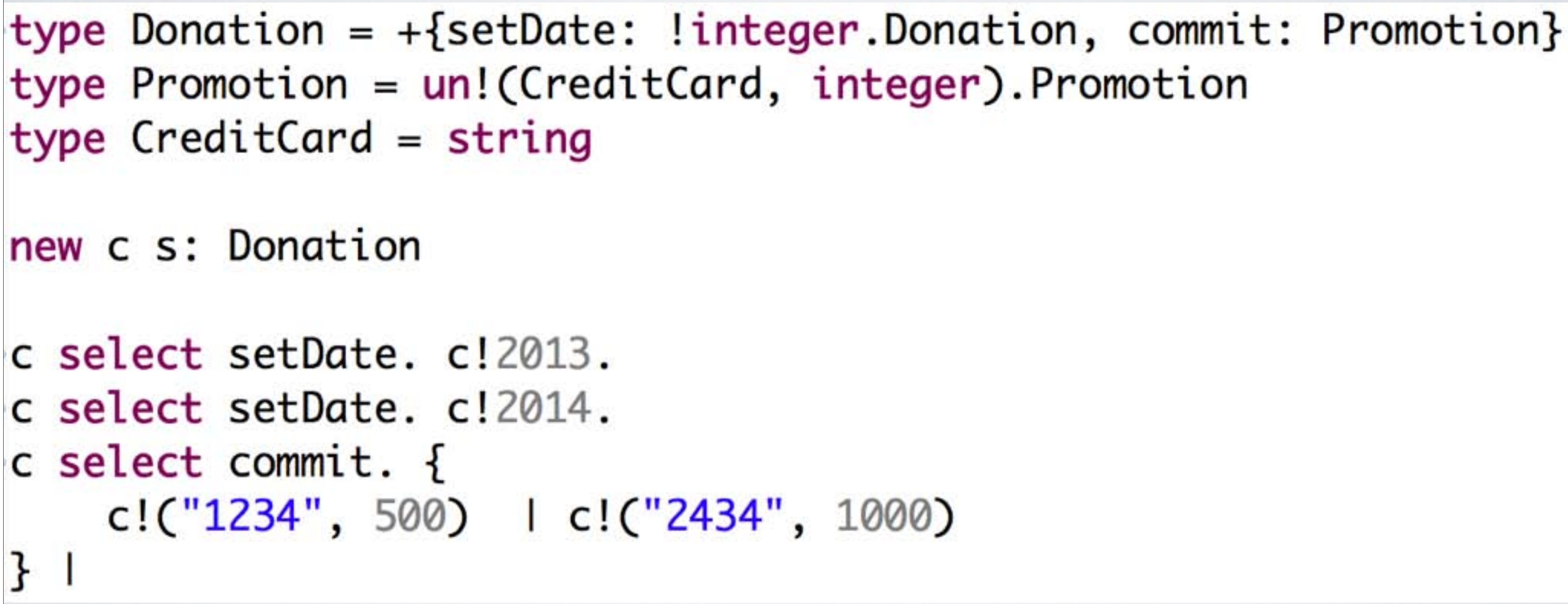
```
new c s: +{setDate: !integer.end, commit: end}
c select setDate. c!2013 |
case s of
    setDate -> s?x. printIntegerLn!x
    commit  -> printStringLn!"done!"
```

V3 _ RECURSIVE TYPES AND PROCESS DEFINITIONS



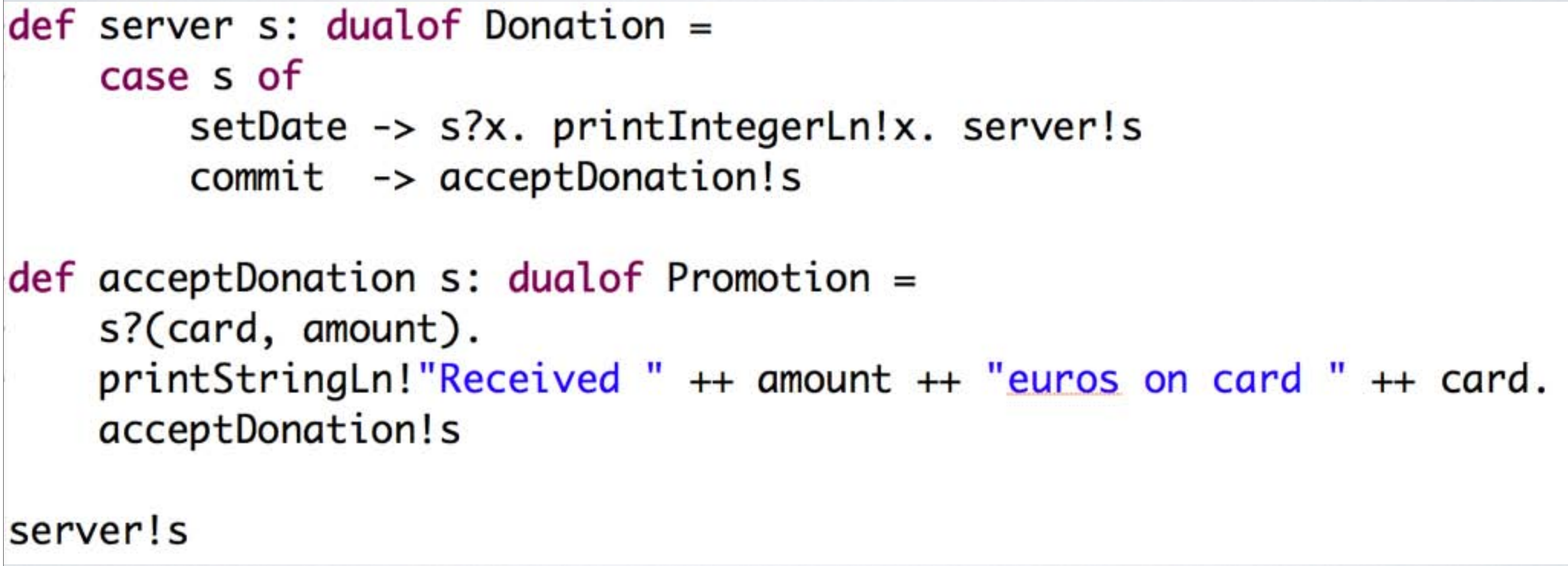
```
type Donation = +{setDate: !integer.Donation, commit: end}

new c s: Donation
c select setDate. c!2013.
c select setDate. c!2014. c select commit |

def server s: dualof Donation =
    case s of
        setDate -> s?x. printIntegerLn!x. server!s
        commit  -> printStringLn!"done!"

server!s
```

V4 _ LINEAR CHANNELS THAT BECOME UNRESTRICTED (I/II)



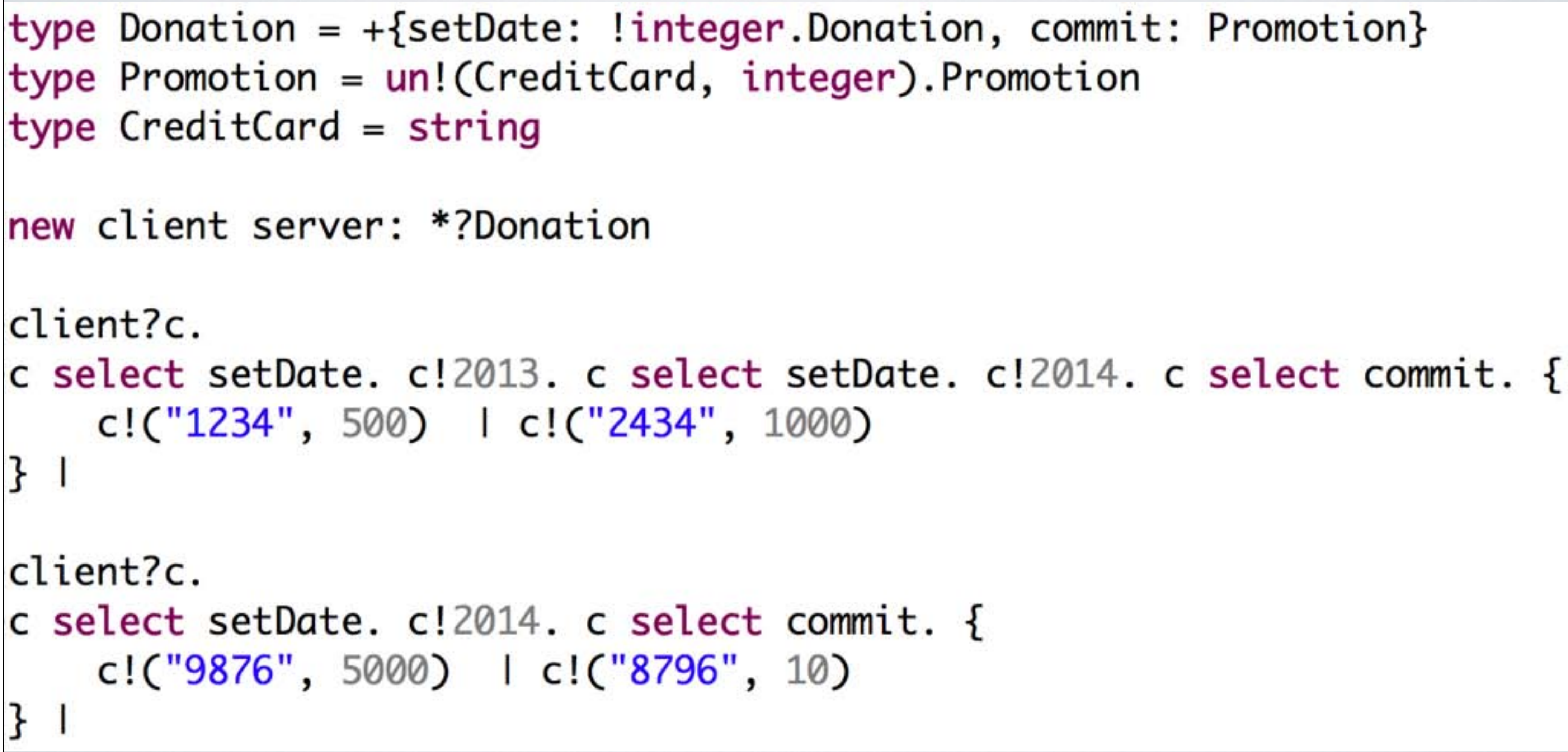
```
type Donation = +{setDate: !integer.Donation, commit: Promotion}
type Promotion = un!(CreditCard, integer).Promotion
type CreditCard = string

new c s: Donation

c select setDate. c!2013.
c select setDate. c!2014.
c select commit. {
   c!("1234", 500)  | c!("2434", 1000)
} |
```

V4 _ LINEAR CHANNELS THAT BECOME UNRESTRICTED (II/II)



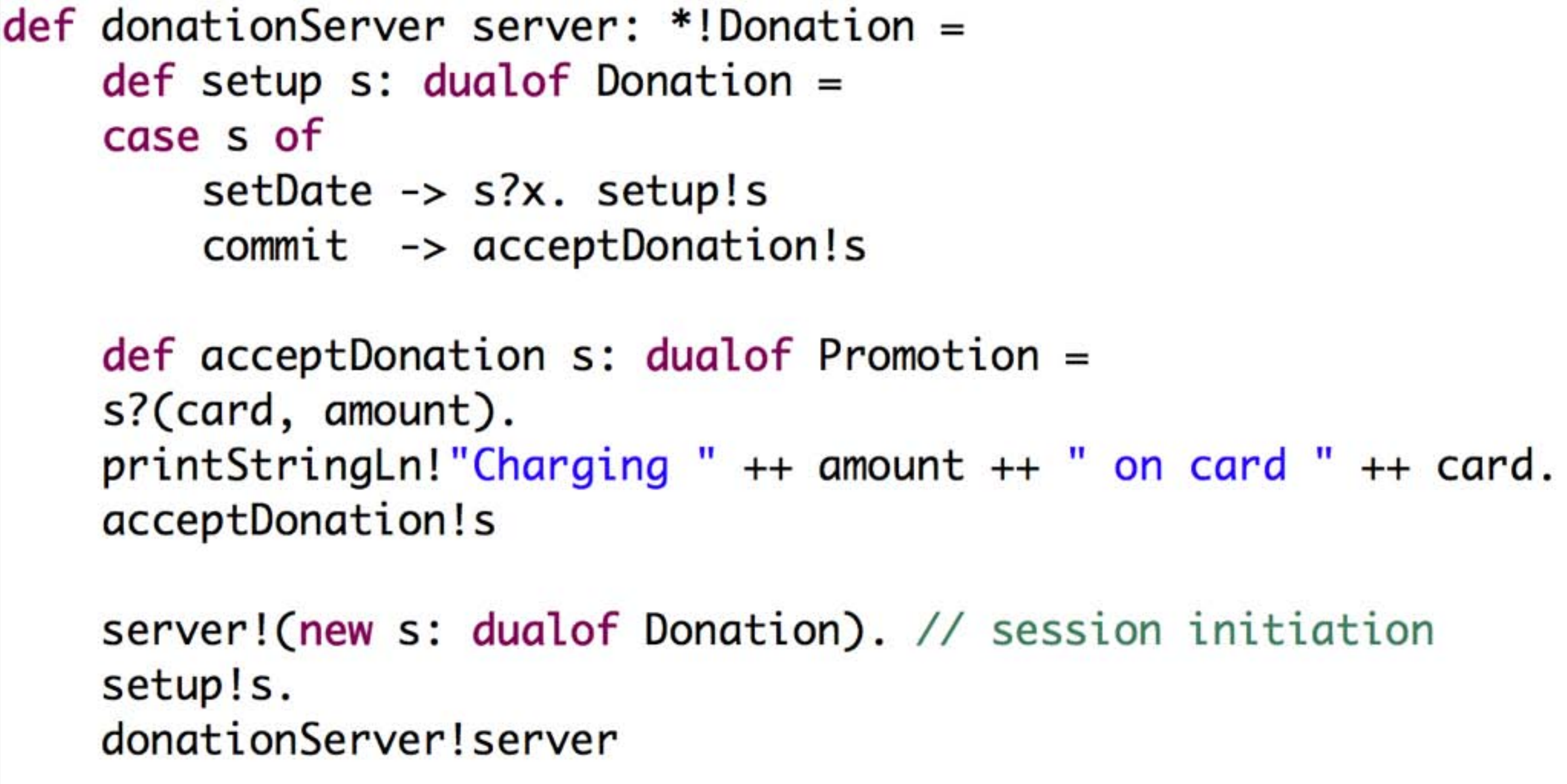
```
def server s: dualof Donation =
    case s of
        setDate -> s?x. printIntegerLn!x. server!s
        commit  -> acceptDonation!s

def acceptDonation s: dualof Promotion =
    s?(card, amount).
    printStringLn!"Received " ++ amount ++ "euros on card " ++ card.
    acceptDonation!s

server!s
```

V5_ MULTIPLE CLIENTS (I/II)



```
type Donation = +{setDate: !integer.Donation, commit: Promotion}
type Promotion = un!(CreditCard, integer).Promotion
type CreditCard = string

new client server: *?Donation

client?c.
c select setDate. c!2013. c select setDate. c!2014. c select commit. {
    c!("1234", 500)  | c!("2434", 1000)
} |

client?c.
c select setDate. c!2014. c select commit. {
    c!("9876", 5000)  | c!("8796", 10)
} |
```

V5_ MULTIPLE CLIENTS (II/II)



```
def donationServer server: *!Donation =
    def setup s: dualof Donation =
    case s of
        setDate -> s?x. setup!s
        commit  -> acceptDonation!s

    def acceptDonation s: dualof Promotion =
    s?(card, amount).
    printStringLn!"Charging " ++ amount ++ " on card " ++ card.
    acceptDonation!s

    server!(new s: dualof Donation). // session initiation
    setup!s.
    donationServer!server
```

CONCLUSION _ FUNDAMENTALS OF SESSION TYPES

- Session types describe continuous interaction, provide for protocol description
- Work well with imperative, functional and OO languages
- When incorporated in programming languages session types prevent a series of runtime errors
- May also be used to monitor communication on applications built with untyped (or non session typed) languages

NEXT

- Part II _ Specification and verification of distributed applications using multiparty session types

The Scribble Protocol Language

Specification and verification of distributed applications using multiparty session types

Raymond Hu (Imperial College London, Cognizant)

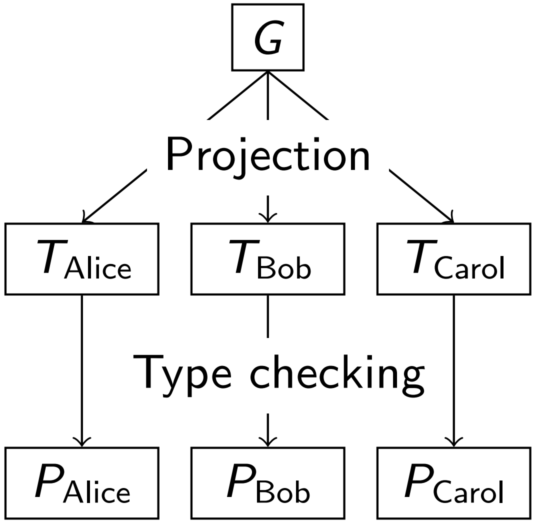
and the Scribble team

`http://www.doc.ic.ac.uk/~rhu/popl14tutorial.pdf`

Outline

- Background:
  - Multiparty session types (MPST)
  - The Scribble protocol language
  - Active use case project: Ocean Observatories Initiative
- Scribble by examples
  - Global protocol specification
  - Multiparty protocol validation (well-formedness)
  - Dynamic MPST verification by runtime monitoring of conversation endpoints
- `http://www.doc.ic.ac.uk/~rhu/popl14tutorial.pdf`

Background: Multiparty Session Types (MPST) 1/2



- Global session type
  - $G = A \to B : m_1;\ B \to C : m_2;\ C \to A : m_3$
- Local session types
  - Slice of global protocol relevant to each role
  - Mechanically derived from global protocol
  - $T_A = B!m_1.\, C?m_3$
- Process language
  - Execution model of I/O actions by session participants
  - $P_A = s(x).\, s!B(m1).\, s?C(x)$
- (Static) type checking for *communication safety*

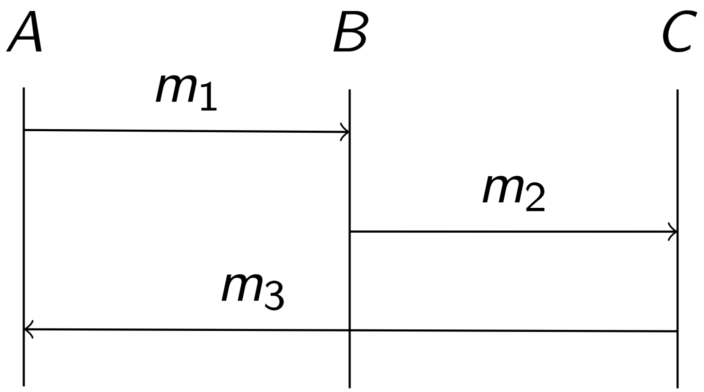
[POPL08] *Multiparty asynchronous session types.* Honda et al.

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions.* Bettini et al.

Background: Multiparty Session Types (MPST) 2/2

- Specifying protocols involving more than two parties!

$$G = A \to B : m_1;\ B \to C : m_2;\ C \to A : m_3$$



- Stronger safety than separate binary session types:

$$\begin{array}{ll}
P_A = s_{AC}?x.s_{AB}!m_1 & T_{AB} = B!m_1,\ T_{AC} = C?m_3 \\
P_B = s_{BA}?y.s_{BC}!m_2 & T_{BA} = A?m_1,\ T_{BC} = C!m_2 \\
P_C = s_{CB}?z.s_{C}!m_3 & T_{CB} = B?m_2,\ T_{CA} = C!m_3
\end{array}$$

× deadlock (due to lost causality between inter- (binary) session actions)

The Scribble protocol language

- Scribble: adapts and extends MPST as an engineering language for describing multiparty message passing protocols
  - Communication model: asynch., reliable, role-to-role ordering

```
global protocol MyProtocol(role A, role B, role C) {
  m1(int) from A to B;
  rec X {
    choice at B {
      m2(String) from B to C;
      continue X;
    } or {
      m3() from B to C;
} } }
```

- Global and local protocol definitions
  - Other features: parallel protocols, subprotocol composition, parameterised protocol declarations, interruptible conversations

[COB12] *Structuring communication with session types*. Honda et al.

[ICDCIT11] *Scribbling interactions with a formal foundation*. Honda et al.

Industry collaborations

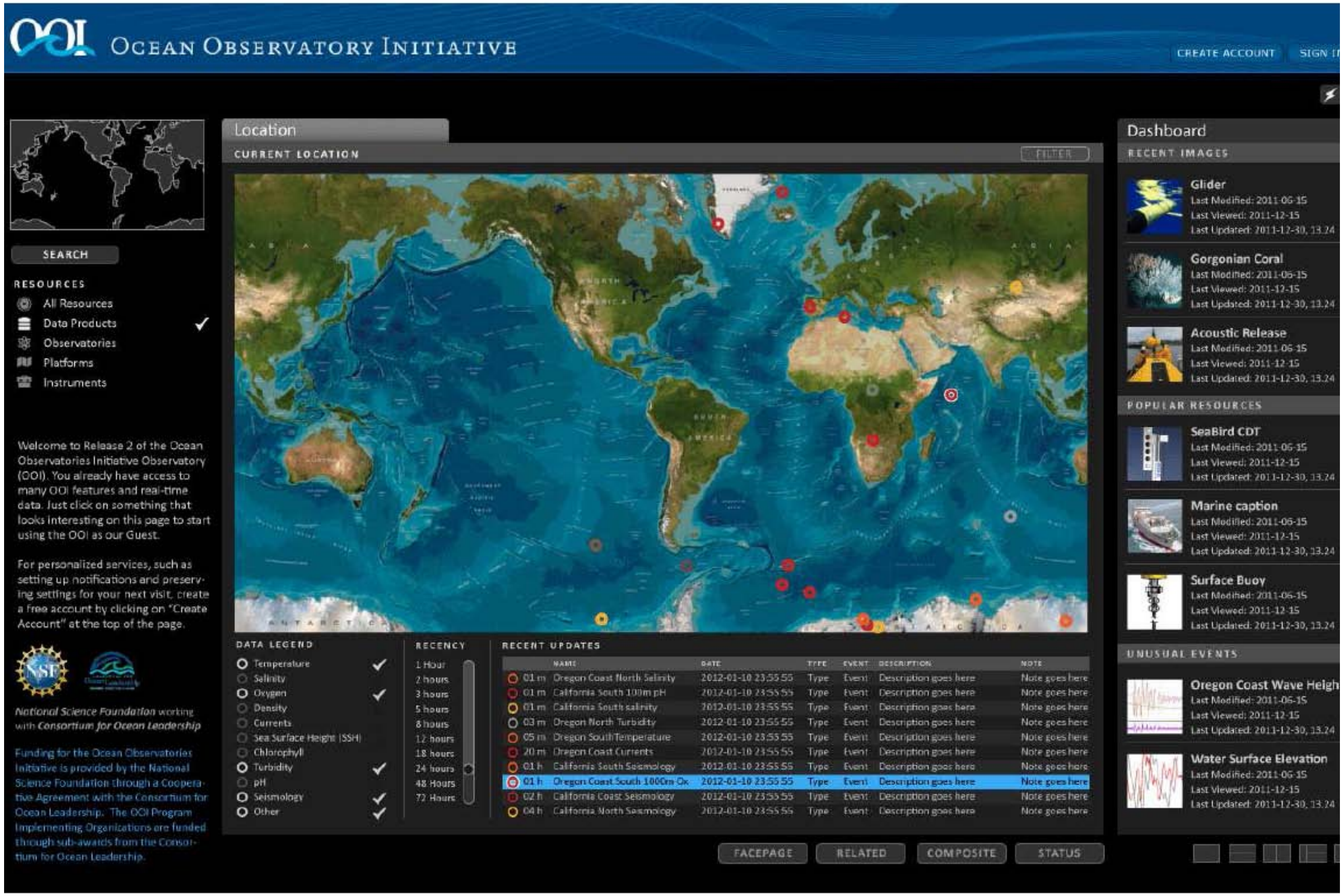
- JBoss Savara: Tool support for Testable Architecture frameworks (Red Hat, Cognizant)
  - Scribble: intermediate protocol language underneath BPMN2/WS-CDL user interface
  - Tooling: global-to-local projection, protocol/system simulations:
    - Requirements model (e.g. sequence diagram traces) against service specification
    - System outputs (e.g. log files) against requirements/service model

[JBOSS] http://www.jboss.org/savara
http://www.jboss.org/scribble

[TA] http://www.cognizant.com/InsightsWhitepapers/SOA_Manifesto_WP1.2010.pdf

Ocean Observatories Initiative (OOI) 1/2

- NSF project ($400M, 5 years) to build a cyberinfrastruture for the remote acquisition and delivery of oceanography data



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.

Funding for the Ocean Observatories Initiative is provided by the National Science Foundation through a Cooperative Agreement with the Consortium for Ocean Leadership. The OOI Program Implementing Organizations are funded through sub-awards from the Consortium for Ocean Leadership.

RECENT UPDATES

	NAME	DATE	TYPE	EVENT	DESCRIPTION	NOTE
○ 01 m	Oregon Coast North Salinity	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 01 m	California South 100m pH	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 01 m	California South salinity	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 03 m	Oregon North Turbidity	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 05 m	Oregon SouthTemperature	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 20 m	Oregon Coast Currents	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 01 h	California South Seismology	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 01 h	Oregon Coast South 1000m Ox	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 02 h	California Coast Seismology	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
○ 04 h	California North Seismology	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here

Glider
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Gorgonian Coral
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Acoustic Release
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

SeaBird CDT
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Marine caption
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Surface Buoy
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Oregon Coast Wave Heigh
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Water Surface Elevation
Last Modified: 2011-06-15
Last Viewed: 2011-12-15
Last Updated: 2011-12-30, 13.24

Ocean Observatories Initiative (OOI) 2/2

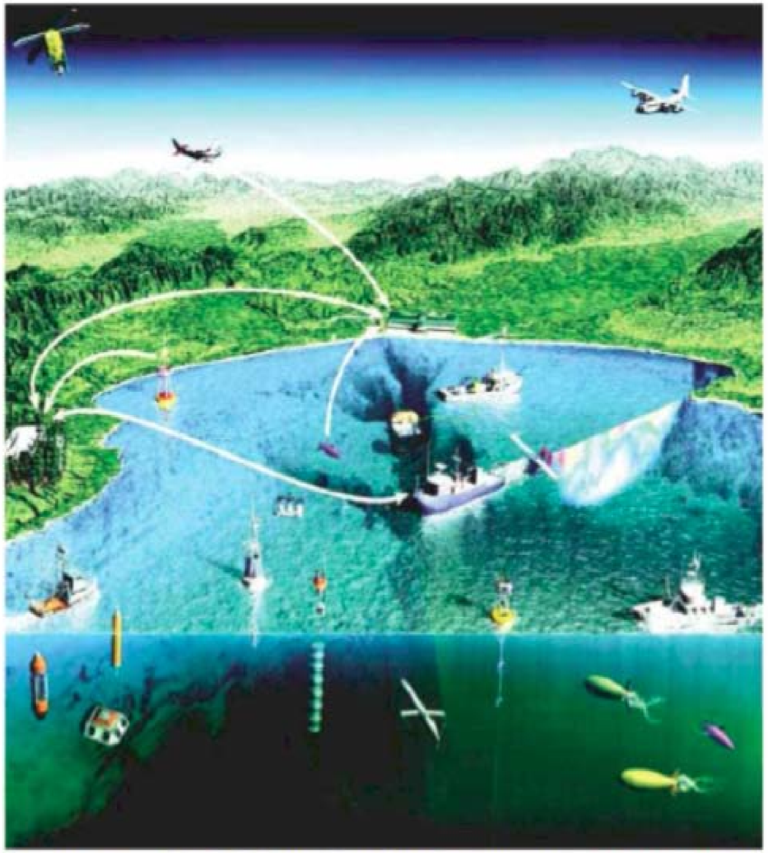
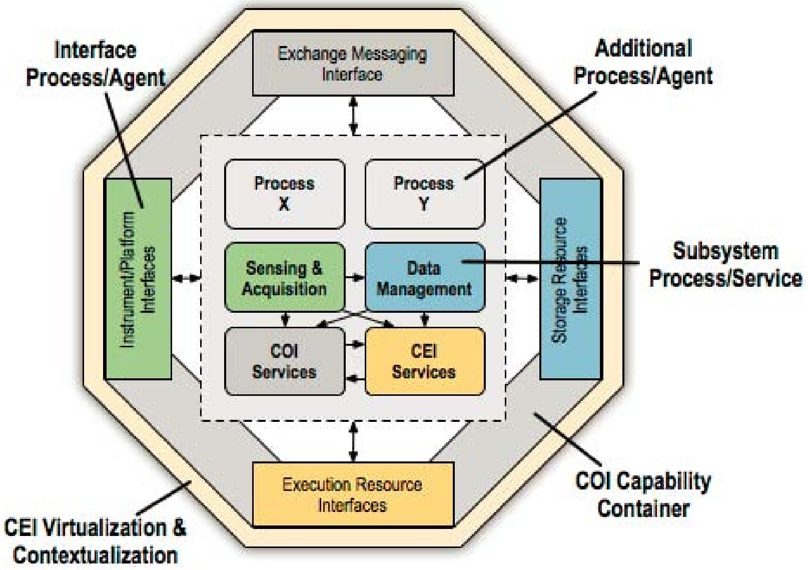


Figure 3: Observatory comprised of ships, aircraft and autonomous vehicles linked to assimilation modeling capabilities on shore

- COI: Python-based endpoint platforms (Capability Containers), AMQP-based messaging network



Interface Process/Agent
Exchange Messaging Interface
Additional Process/Agent
Process X
Process Y
Instrument/Platform Interfaces
Sensing & Acquisition
Data Management
Storage Resource Interfaces
Subsystem Process/Service
COI Services
CEI Services
Execution Resource Interfaces
COI Capability Container
CEI Virtualization & Contextualization

Capability Container

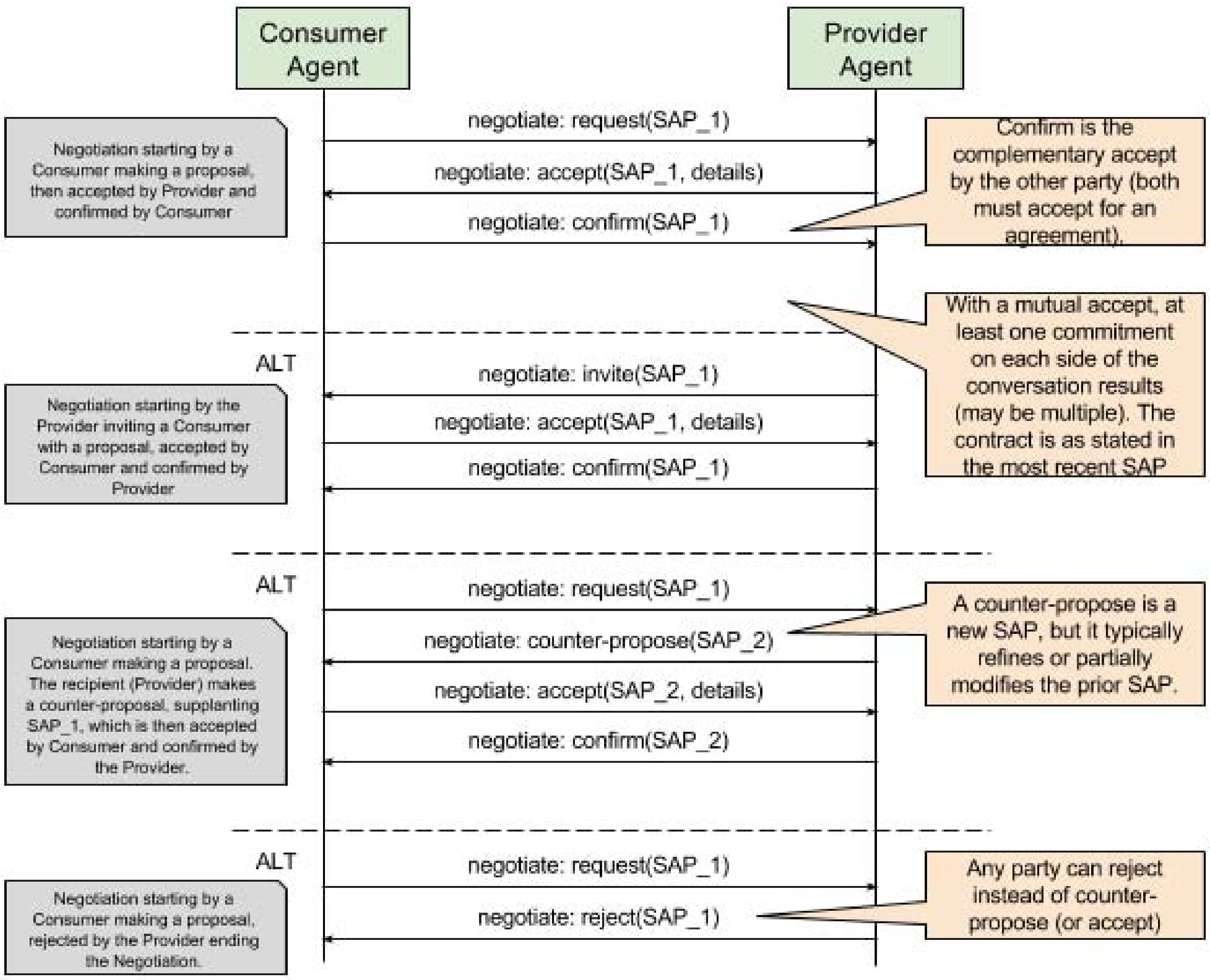
Scribble people

Matthew Arrott	UCSD, Ocean Observatories Initative
Laura Bocchi	Imperial College London
Gary Brown	Red Hat
Tzu-Chun Chen	L'Università di Torino
Romain Demangeon	Université Pierre et Marie Curie
Pierre-Malo Deniélou	Royal Holloway, University of London
Kohei Honda	Queen Mary, University of London
Raymond Hu	Imperial College London
Rumyana Neykova	Imperial College London
Nicholas Ng	Imperial College London
Nobuko Yoshida	Imperial College London

Scribble examples

- Basic scribble (OOI agent negotiation)
  - Applied MPST framework:
    Global well-formedness; local projection; FSM generation
- Parameterised protocols and subprotocols
  - OOI RPC service composition
  - Agent negotiation refactored
- Interruptible conversations: (OOI resource usage control)
- OOI endpoint code and runtime monitoring
- We demo the current status of Scribble
  - The work on Scribble and the OOI integration (and other applications of MPST) is ongoing

OOI agent negotiation 1/5


Consumer Agent
Provider Agent
negotiate: request(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: invite(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: request(SAP_1)
negotiate: counter-propose(SAP_2)
negotiate: accept(SAP_2, details)
negotiate: confirm(SAP_2)
ALT
negotiate: request(SAP_1)
negotiate: reject(SAP_1)

Negotiation starting by a Consumer making a proposal, then accepted by Provider and confirmed by Consumer

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

Negotiation starting by the Provider inviting a Consumer with a proposal, accepted by Consumer and confirmed by Provider

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.

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

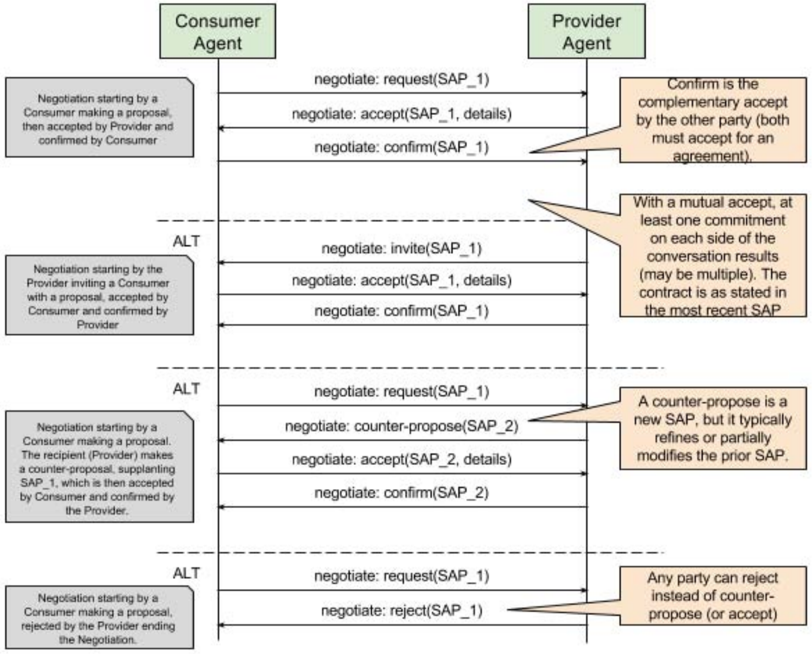
Negotiation starting by a Consumer making a proposal, rejected by the Provider ending the Negotiation.

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

- `https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Negotiate+Protocol`

OOI agent negotiation 2/5

```
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
```



Consumer Agent

Provider 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)

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

ALT

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)

ALT

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)

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

ALT

Negotiation starting by a Consumer making a proposal, rejected by the Provider ending the Negotiation.

negotiate: request(SAP_1)

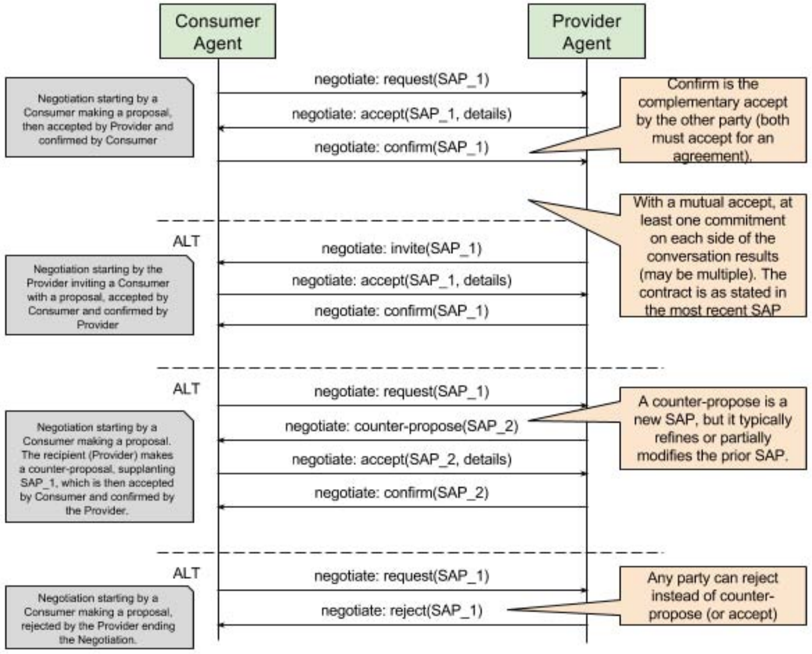
negotiate: reject(SAP_1)

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

```
}
```

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;

} }
```


Consumer Agent
Provider Agent
negotiate: request(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: invite(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: request(SAP_1)
negotiate: counter-propose(SAP_2)
negotiate: accept(SAP_2, details)
negotiate: confirm(SAP_2)
ALT
negotiate: request(SAP_1)
negotiate: reject(SAP_1)

Negotiation starting by a Consumer making a proposal, then accepted by Provider and confirmed by Consumer

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

Negotiation starting by the Provider inviting a Consumer with a proposal, accepted by Consumer and confirmed by Provider

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.

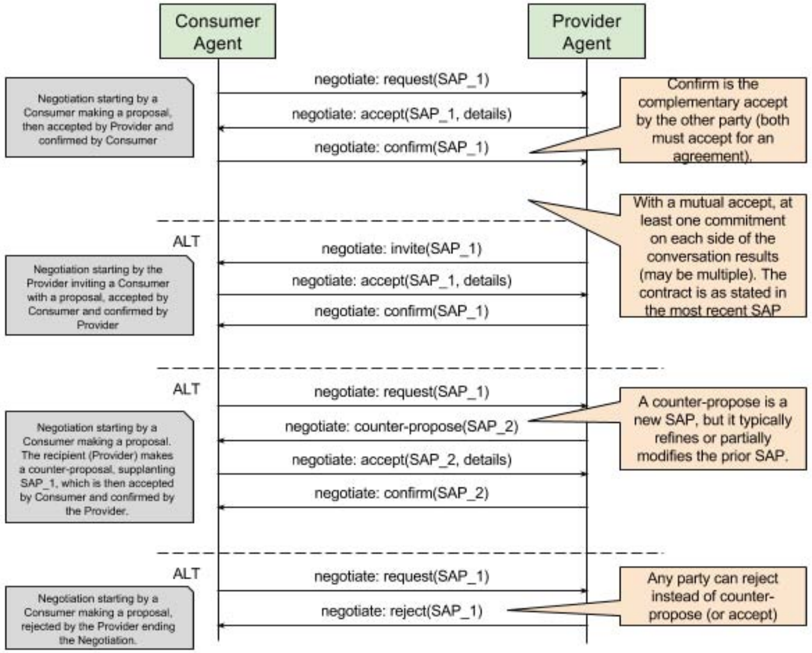
A counter-propose is a new SAP, but it typically refines or partially modifies the prior SAP.

Negotiation starting by a Consumer making a proposal, rejected by the Provider ending the Negotiation.

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

OOI agent negotiation 4/5

```
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;

} } }
```



Consumer Agent
Provider Agent
negotiate: request(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: invite(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: request(SAP_1)
negotiate: counter-propose(SAP_2)
negotiate: accept(SAP_2, details)
negotiate: confirm(SAP_2)
ALT
negotiate: request(SAP_1)
negotiate: reject(SAP_1)

Negotiation starting by a Consumer making a proposal, then accepted by Provider and confirmed by Consumer

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

Negotiation starting by the Provider inviting a Consumer with a proposal, accepted by Consumer and confirmed by Provider

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.

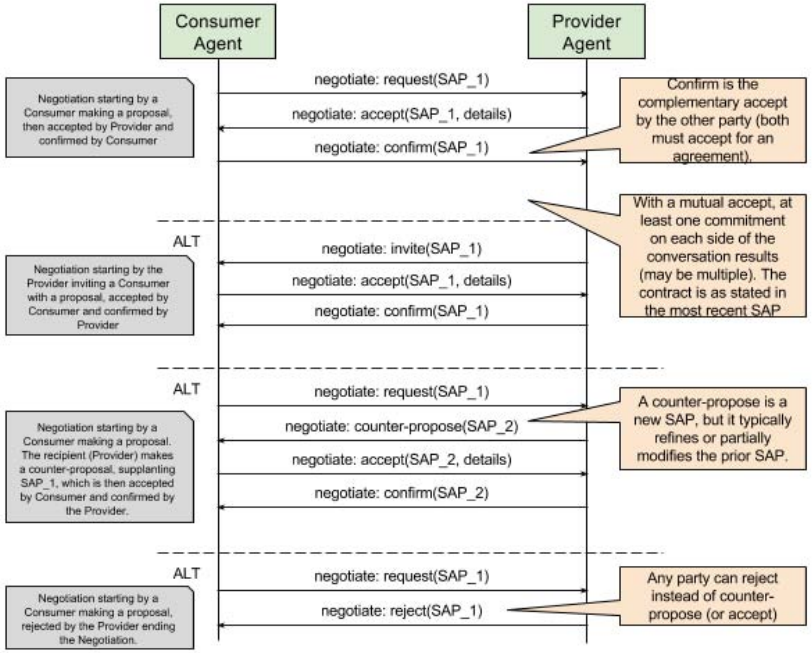
A counter-propose is a new SAP, but it typically refines or partially modifies the prior SAP.

Negotiation starting by a Consumer making a proposal, rejected by the Provider ending the Negotiation.

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

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;
      }
    }
```


Consumer Agent
Provider Agent
negotiate: request(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: invite(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(SAP_1)
ALT
negotiate: request(SAP_1)
negotiate: counter-propose(SAP_2)
negotiate: accept(SAP_2, details)
negotiate: confirm(SAP_2)
ALT
negotiate: request(SAP_1)
negotiate: reject(SAP_1)

Negotiation starting by a Consumer making a proposal, then accepted by Provider and confirmed by Consumer

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

Negotiation starting by the Provider inviting a Consumer with a proposal, accepted by Consumer and confirmed by Provider

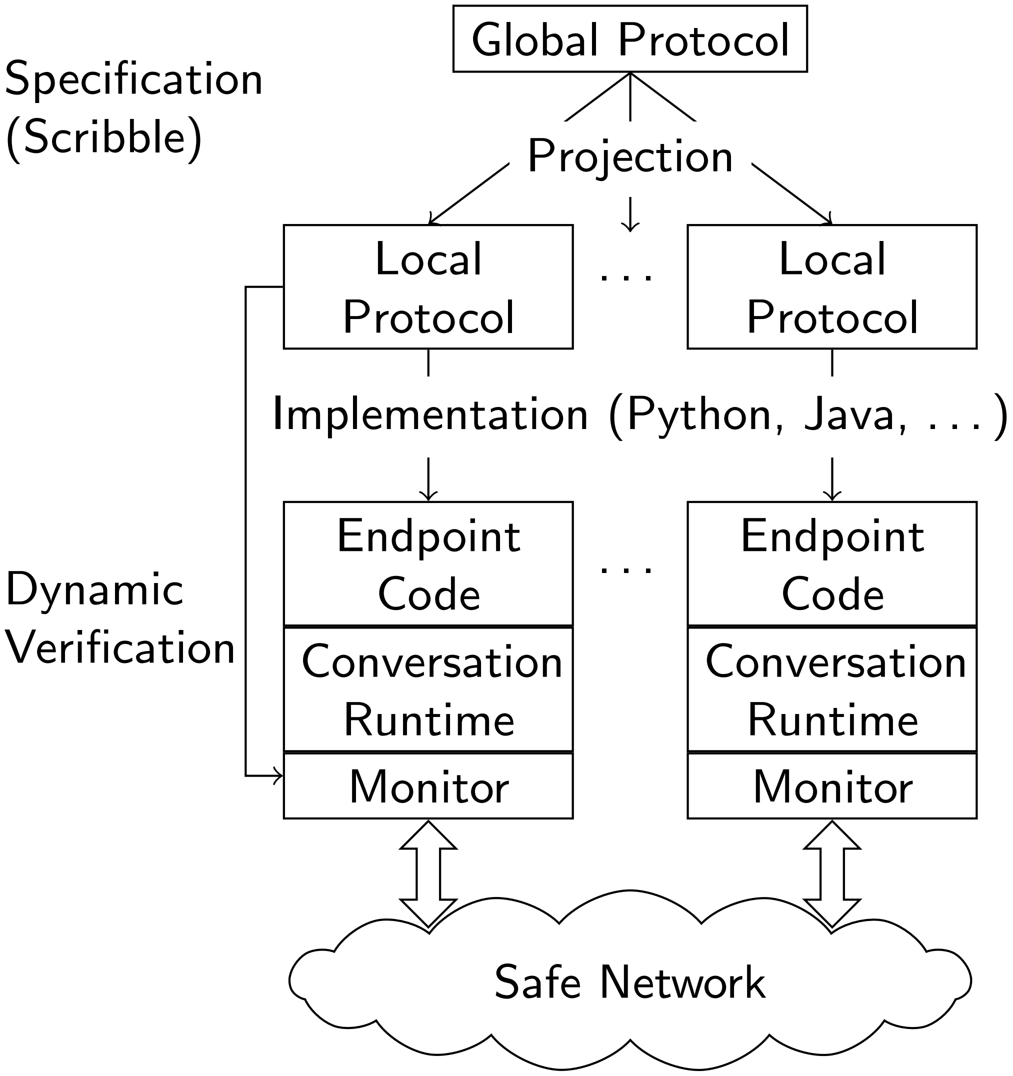
Negotiation starting by a Consumer making a proposal. The recipient (Provider) makes a counter-proposal, supplanting SAP_1, which is then accepted by Consumer and confirmed by the Provider.

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

Negotiation starting by a Consumer making a proposal, rejected by the Provider ending the Negotiation.

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

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

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 well-formedness 2/2

```
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;
} }
```

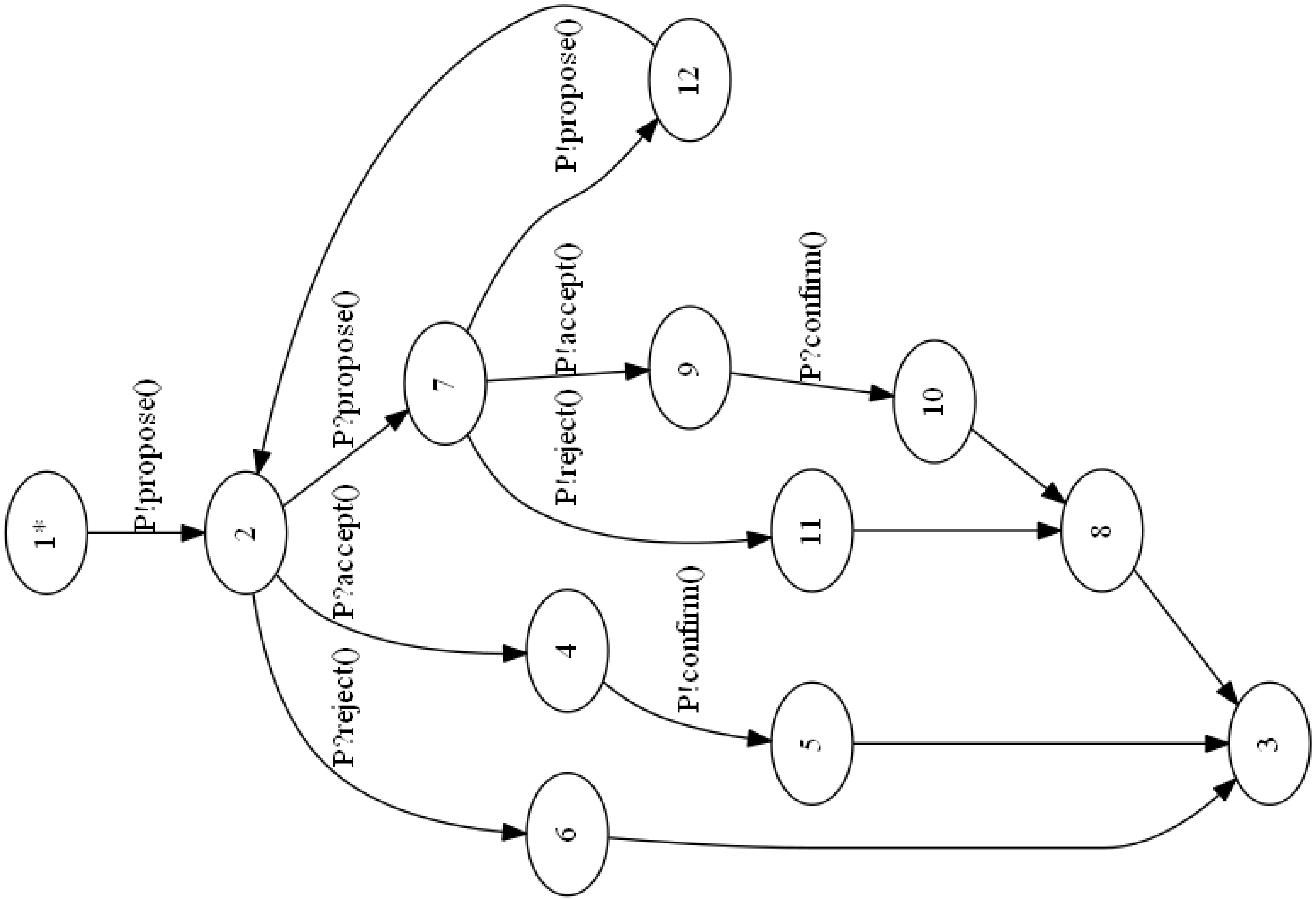
```
global protocol RecursionNoExit(role A, role B, role C, role D) {
  rec X {
    m1() from A to B;
    continue X;
  }
  m2() from A to B; // Unreachable for A, B
  m3() from C to D;
}
```

Local protocol projection (Negotiation Consumer)

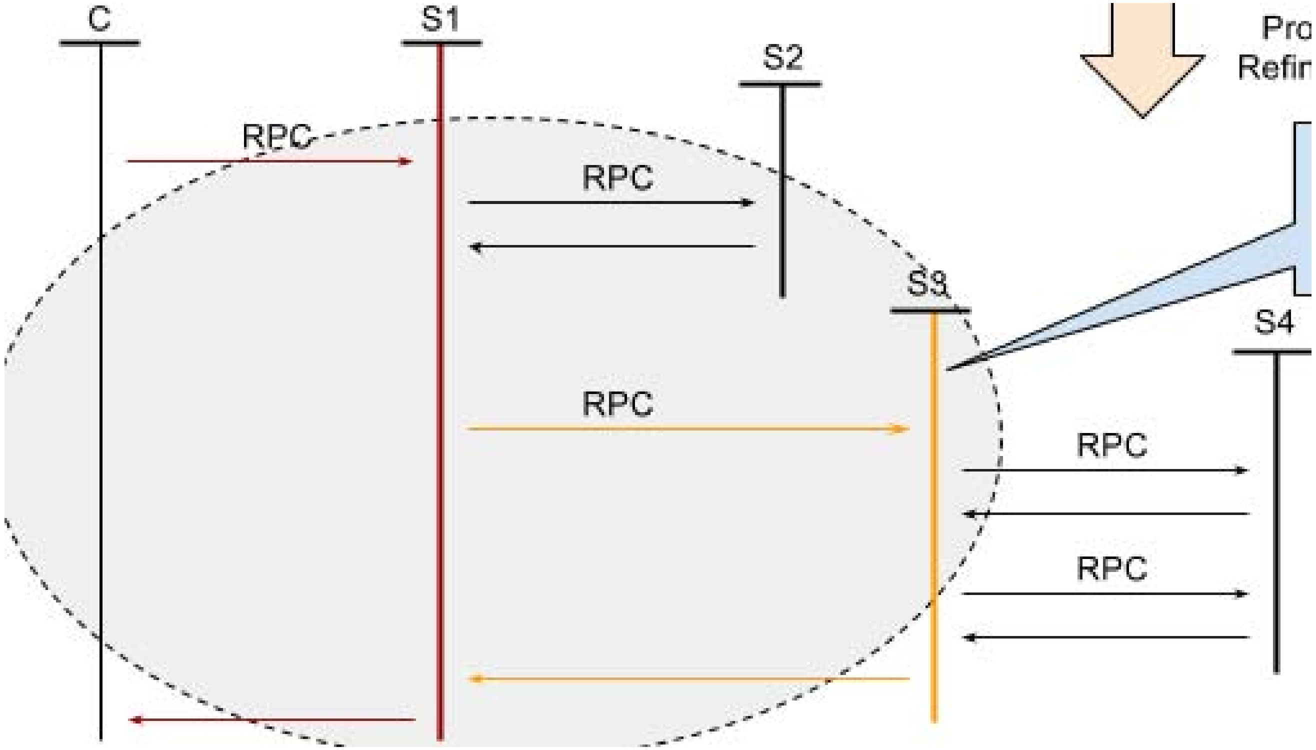
```
// Global
propose(SAP) from C to P;
rec START {
  choice at P {
    accept() from P to C;
    confirm() from C to P;
  } or {
    reject() from P to C;
  } or {
    propose(SAP) from P to C;
    choice at C {
      accept() from C to P;
      confirm() from P to C;
    } or {
      reject() from 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)



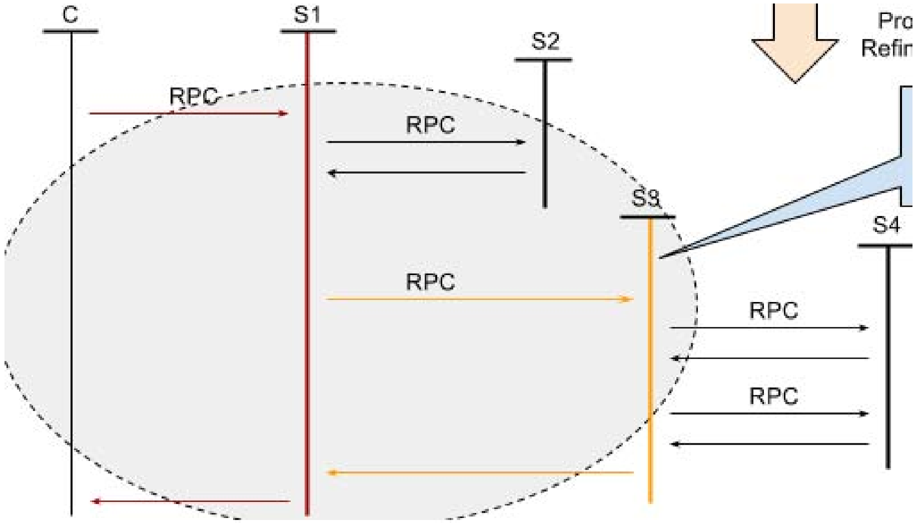
RPC composition 1/4



- https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Conversation+Management

RPC composition 2/4

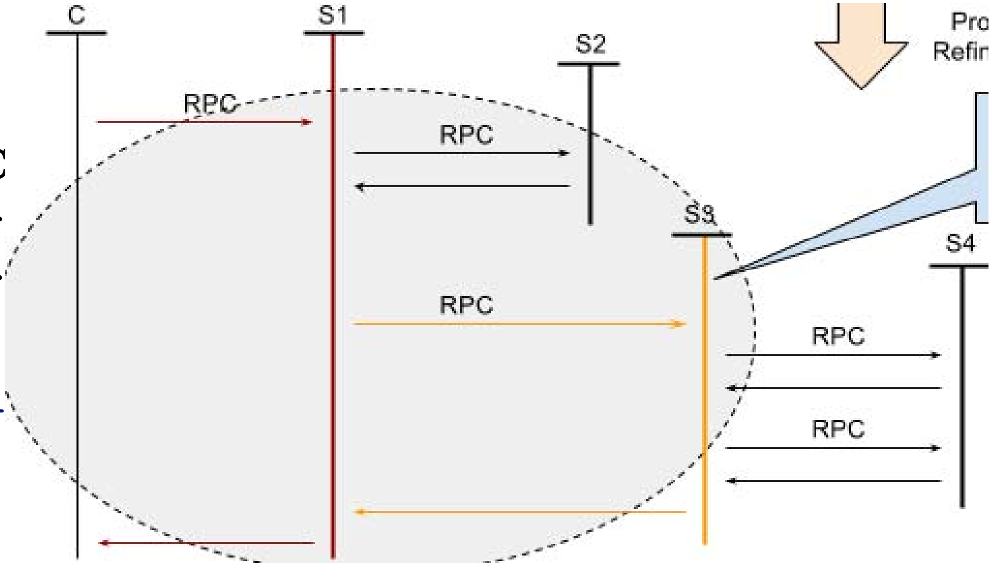
```
global protocol Comp1(role Client as C,
                      role Service1 as S1, role Service2 as S2,
                      role Service3 as S3, role Service4 as S4) {
  m1() from C to S1;
    m2() from S1 to S2;
    m2a() from S2 to S1;
    m3() from S1 to S3;
      m4() from S3 to S4;
      m4a() from S4 to S3;
      m5() from S3 to S4;
      m5a() from S4 to S3;
    m3a() from S3 to S1;
  m1a() from S1 to C;
}
```



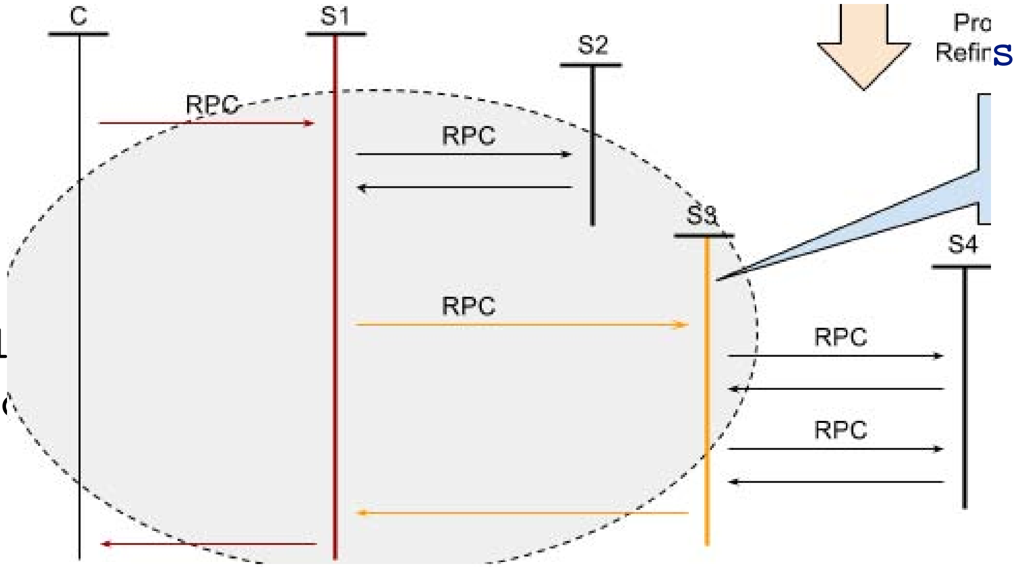
- https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Conversation+Management

RPC composition 3/4 (parameterised subprotocols)

```
global protocol RPC<sig M1, sig M2>(role Client as C, role Server as S)
  M1 from C to S;
  M2 from S to C;
}

global protocol Relay<sig M1, sig M2>(
    role First as F, role Middle as M, role Last as L) {
  M1 from F to M;
  M2 from M to L;
}

global protocol Comp3(role C
                     role Ser
                     role Ser
  do Relay<m1(), m2()>(C as
  do Relay<m2a(), m3()>(S2 a
  do RPC<m4(), m4a()>(S3 as
  do RPC<m5(), m5a()>(S3 as
  do Relay<m3a(), m1a()>(S2 as First, S1 as Middle, C as Last);
}
```



RPC composition 4/4



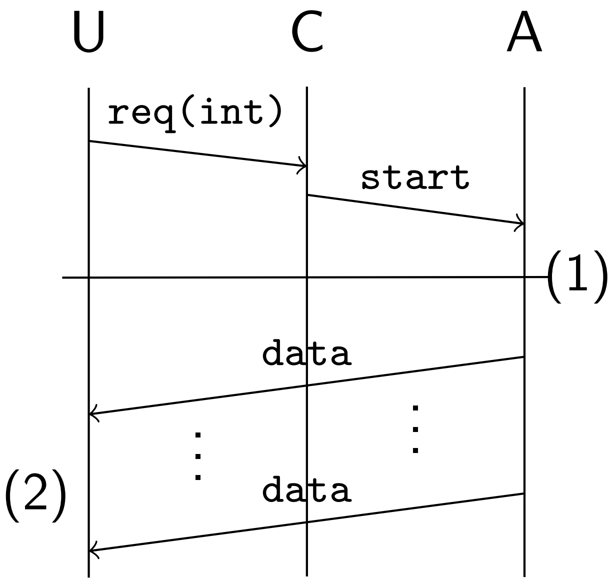
```
global protocol RPC<sig M1,                                         S)
  M1 from C to S;
  M2 from S to C;
}

global protocol Relay<sig M1
    role First as F, role Mi
  M1 from F to M;
  M2 from M to L;
}

global protocol Comp3(role Client as C,
                      role Service1 as S1, role Service2 as S2,
                      role Service3 as S3, role Service4 as S4) {
  do Relay<m1(), m2()>(C as First, S1 as Middle, S2 as Last);
  do Relay<m2a(), m3()>(S2 as First, S1 as Middle, S3 as Last);
  do RPC<m4(), m4a()>(S3 as Client, S4 as Server);
  do RPC<m5(), m5a()>(S3 as Client, S4 as Server);
  do Relay<m3a(), m1a()>(S2 as First, S1 as Middle, C as Last);
}
```

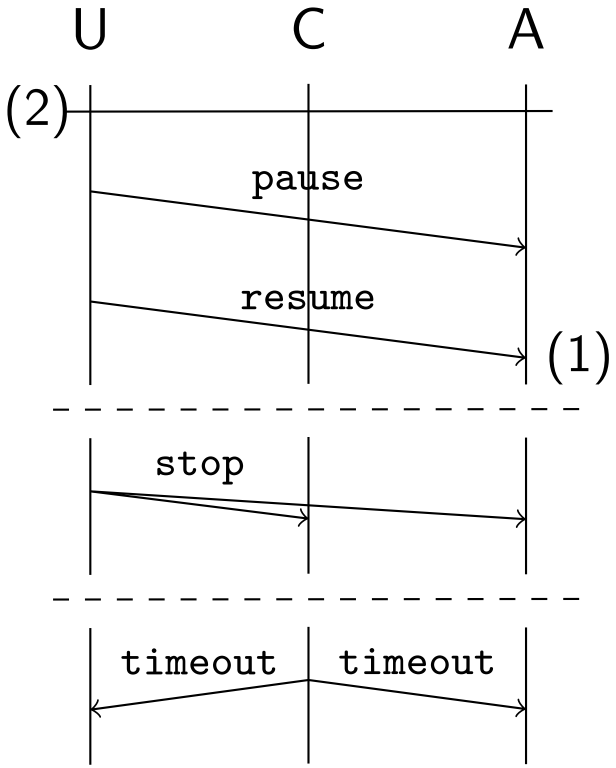
Agent negotiation (refactored)

```
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer, role Producer) {
  propose(SAP) from Consumer to Producer;
  do NegotiateAux(Consumer as Proposer, Producer as CounterParty);
}

global protocol NegotiateAux(
    role Proposer as A, role CounterParty as B) {
  choice at B {
    accept() from B to A;
    confirm() from A to B;
  } or {
    reject() from B to A;
  } or {
    propose(SAP) from B to A;
    do NegotiateAux(B as Proposer, A as CounterParty);
} }
```

Resource Usage Control (interruptible)

- **U**ser, Resource **C**ontroller, Instrument **A**gent
- **U** registers with **C** to use a resource (instrument) via **A** for a specified duration (or another metric)



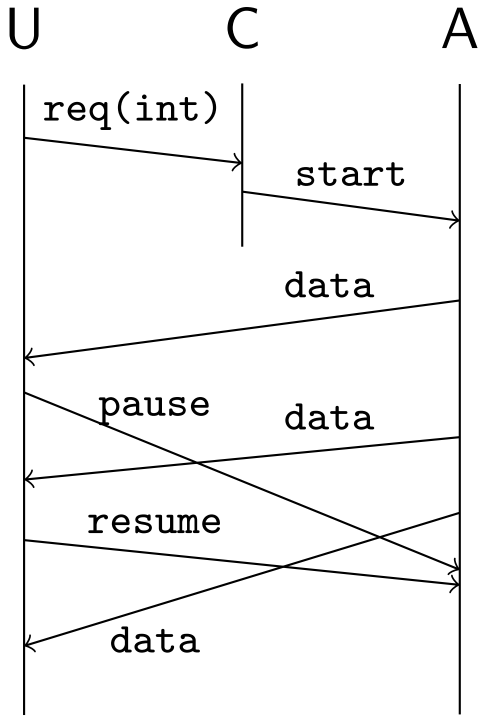


- `https://confluence.oceanobservatories.org/display/CIDev/Resource+Control+in+Scribble`

Extending MPST with interruptible conversations

- Well-formed global types traditionally rule out any ambiguities between roles in conversation instances
  - Sent messages are expected and vice versa
  - No messages lost or redundant

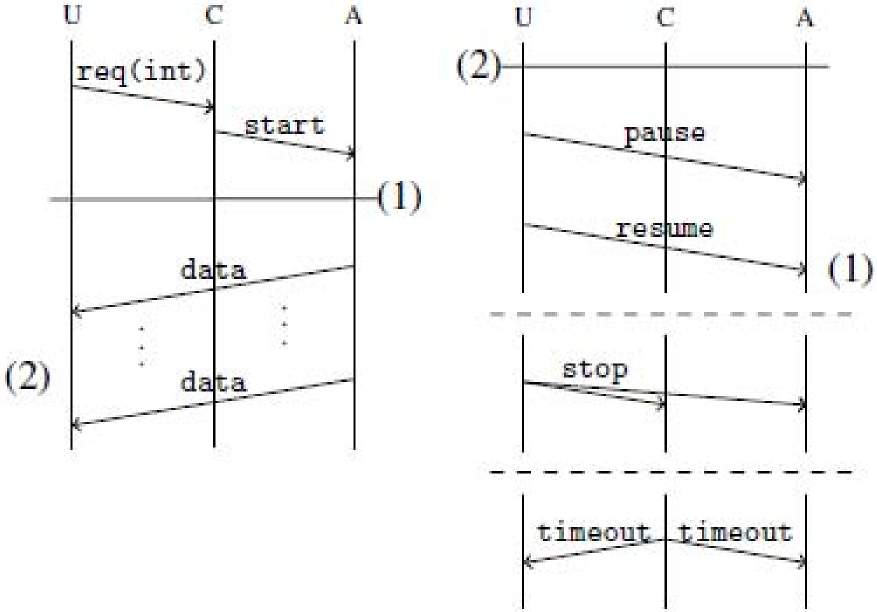
- Asynchronous interrupts: inherent "communication races"
  - Interruptible is a mixed choice, also completely optional
  - Concurrent and nested interrupts
  - Asynchronous entry/exit of interruptible blocks by roles



A valid trace

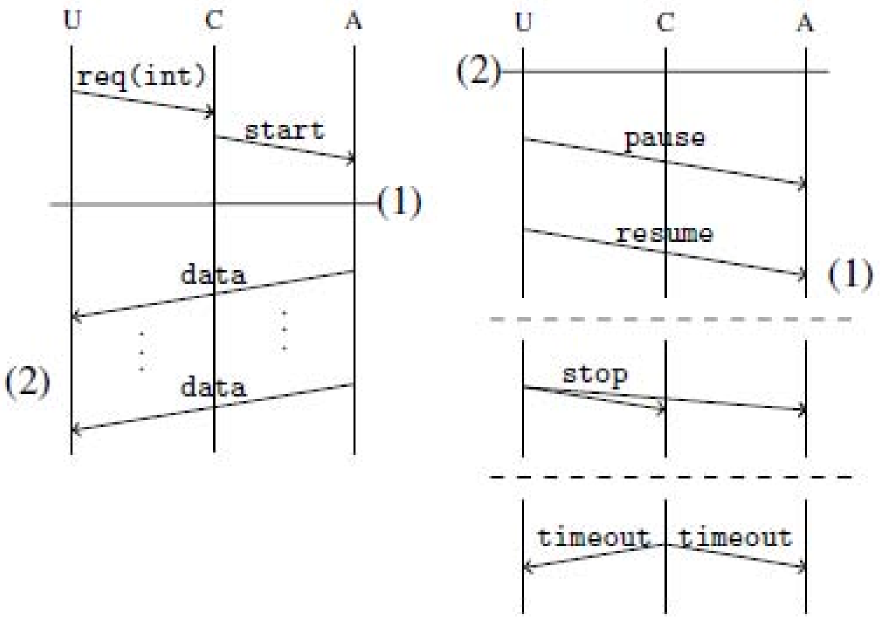
RUC Scribble 1/5 (streaming)

```
global protocol RUC(
    role User as U, role Controller as C, role Agent as A) {




        rec Y {
          data() from A to U;
          continue Y;
        }





}
```



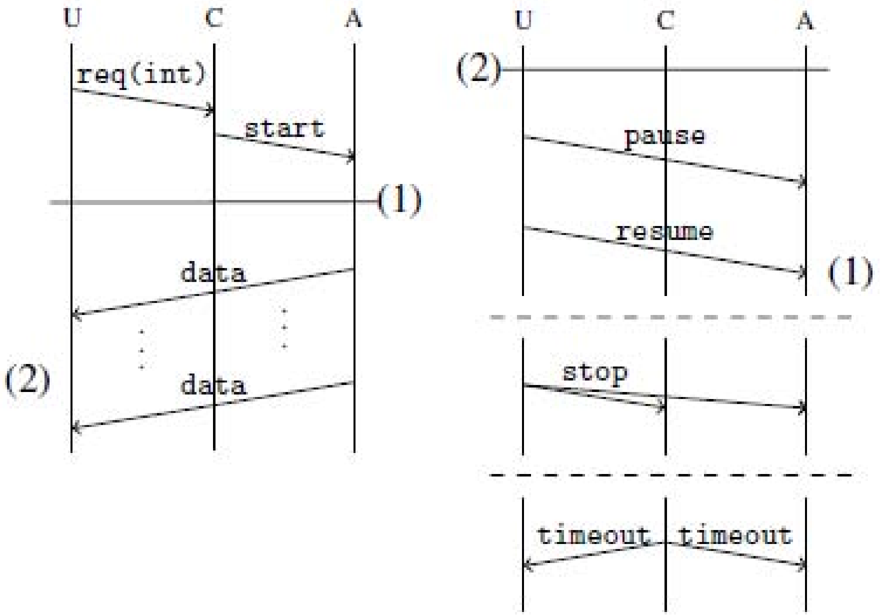
RUC Scribble 2/5 (interruptible stream)

```
global protocol RUC(
    role User as U, role Controller as C, role Agent as A) {



      interruptible {
        rec Y {
          data() from A to U;
          continue Y;
      } }
      with {
        pause() by U;
      }
      resume() from U to A;




}
```



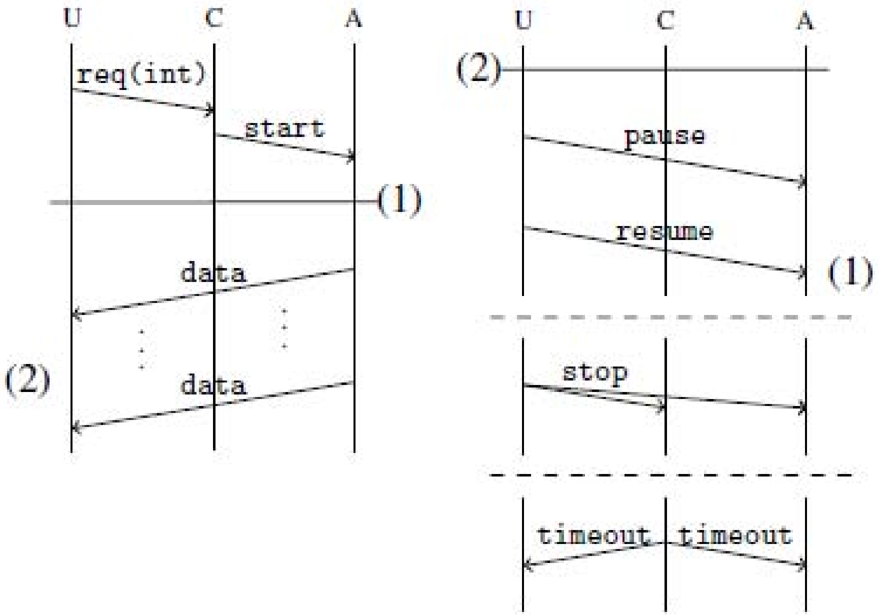
RUC Scribble 3/5

```
global protocol RUC(
    role User as U, role Controller as C, role Agent as A) {


  interruptible {
    rec X {
      interruptible {
        rec Y {
          data() from A to U;
          continue Y;
      } }
      with {
        pause() by U;
      }
      resume() from U to A;
      continue X;
  } }
  with {
    stop() by U;
    timeout() by C;
} }
```



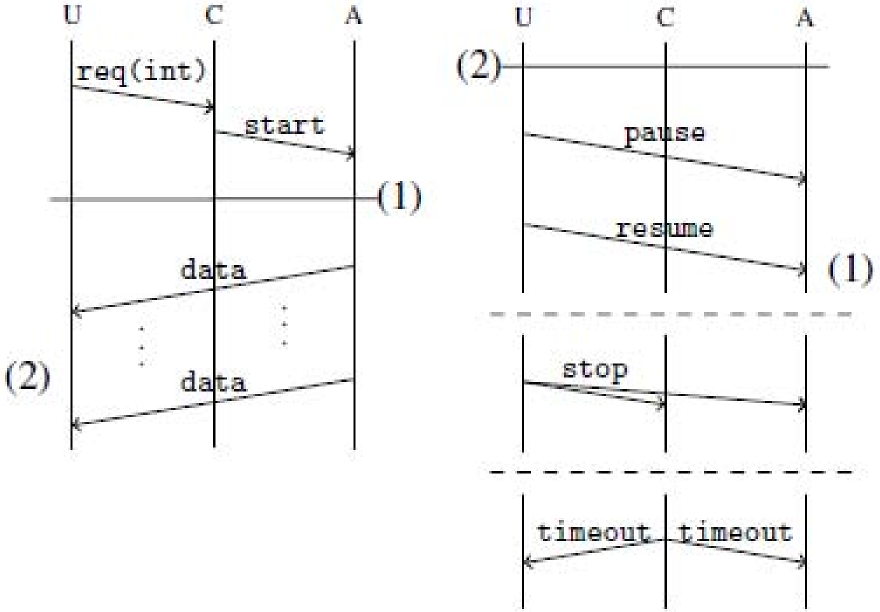
RUC Scribble 4/5

```
global protocol RUC(
  role User as U, role Controller as C, role Agent as A) {
 req(int) from U to C;
 start() from C to A;
 interruptible {
  rec X {
   interruptible {
    rec Y {
     data() from A to U;
     continue Y;
   } }
   with {
    pause() by U;
   }
   resume() from U to A;
   continue X;
 } }
 with {
  stop() by U;
  timeout() by C;
} }
```



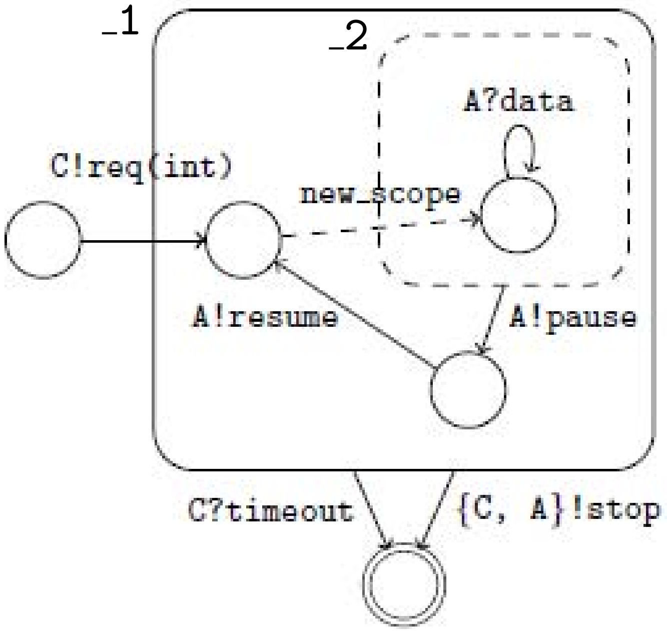
RUC Scribble 5/5 (conversation scopes)

```
global protocol RUC(
    role User as U, role Controller as C, role Agent as A) {
  req(int) from U to C;
  start() from C to A;
  interruptible _1 {
    rec X {
      interruptible _2 {
        rec Y {
          data() from A to U;
          continue Y;
      } }
      with {
        pause() by U;
      }
      resume() from U to A;
      continue X;
  } }
  with {
    stop() by U;
    timeout() by C;
} }
```



MPST-based distributed protocol monitoring 1/3

```
req(int) to C;
interruptible _1 {
  rec X {
    interruptible _2 {
      rec Y {
        data() from A;
        continue Y;
    } } with {
      throws pause() to A;
    }
    resume() to A;
    continue X;
} } with {
  throws stop() to A, C;
  catches timeout() from C;
}
```



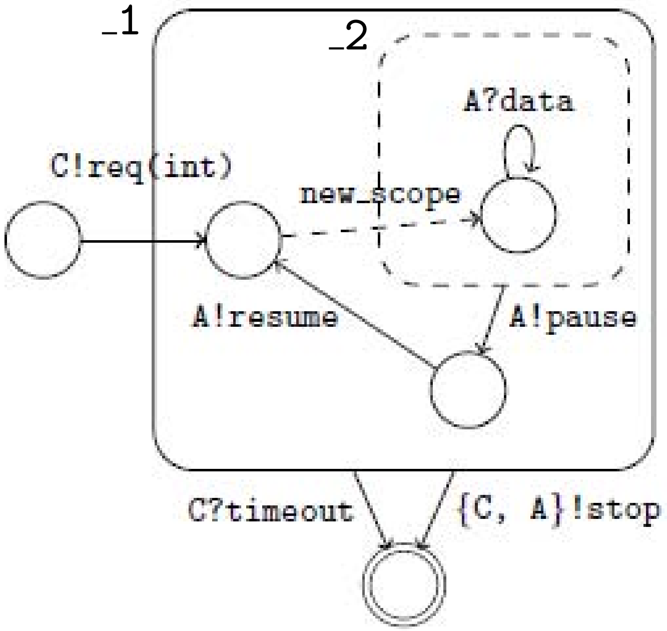
Projection and FSM for U

- Builds on formal MPST-FSM encoding
  - Interruptible scopes modelled by dynamically nested FSMs

[ESOP12] *Multiparty Session Types Meet Communicating Automata*. Deniélou and Yoshida.

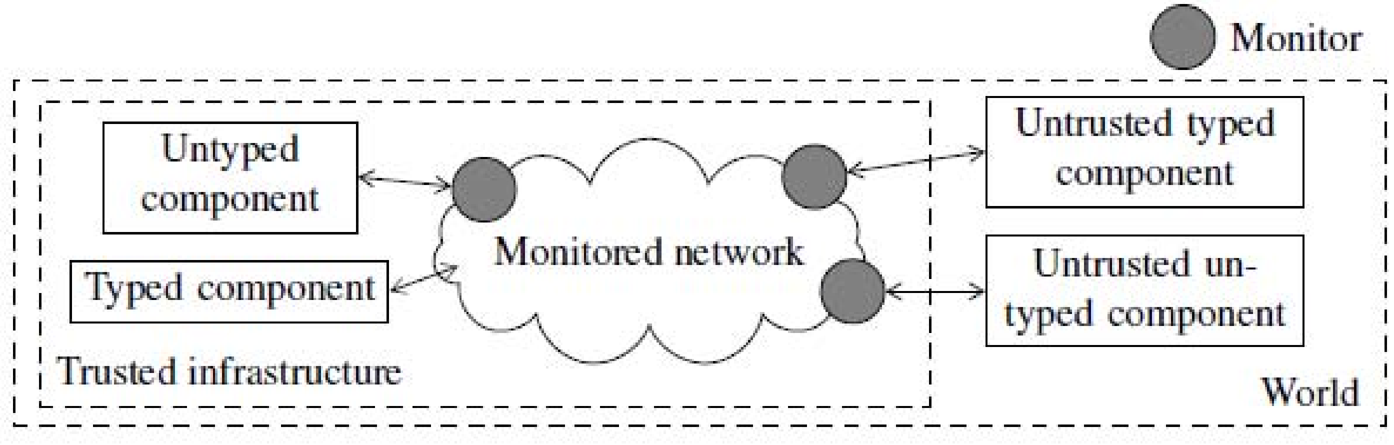
MPST-based distributed protocol monitoring 2/3

```
with conv.join(’user’) as c:
  c.send(controller, ’req’, 100)
  with c.scope(’timeout’ ’stop’) as c1:
    while not self.enough_data():
      with c1.scope(’timeout’, ’stop’) as c2:
        while not batch.full():
          next = c2.recv(agent, ’data’)
          batch.append(next)
        c2.interrupt(’pause’)
        process_data(batch)
      c1.send(agent, ’resume’)
    c1.interrupt(’stop’)
```



- MPST monitoring requirements: complete mediation, Scribble metadata (embedded in payload: msg. operator, source/dest.)
- Errors detected: non-conformance to protocol
  - Local actions: bad I/O, bad operator, bad source role, . . .
  - Remote: firewall exepected messages (operator, role)

MPST-based distributed protocol monitoring 3/3



- Local monitoring of endpoint and environment conversation actions
  - Dynamic verification of MPST communication safety

[RV13] *Practical Interruptible Conversations – Distributed Dynamic Verification with Session Types and Python.* Hu et al.

[FMOODS13] *Monitoring networks through multiparty session types.* Bocchi et al.

[TGC11] *Asynchronous distributed monitoring for multiparty session enforcement.* Chun et al.

Dynamic verification of MPST (with interruptible)

- MPST motivations:
  - MPST type systems typically designed for languages with first-class communication and concurrency features
- Distributed systems motivations:
  - Heterogenous languages, runtime platforms, implementation techniques, . . .
  - Unavailable source code
- OOI use case motivations:
  - Python (untyped languages)
  - OOI governance stack
- Interruptible:
  - Dynamic creation of nested FSMs for fresh scope generation

OOI Demo

Static session type checking

- Session typing checks endpoint code against projections
  - Built for a target language (extension) or API
  - Mapping of protocol “constants” to program entities
  - Conformance of control flow to protocol structure

```
session *s;
role *B, *Seller;
session_init(&argc, &argv, &s, "TwoBuyers_A.scr");
send_string(str_title, B, TITLE);
recv_int(&quote, Seller, QUOTE);
while (true) {
  probe_label(&label, B);
  if (has_label(label, "accept")) {
    vsend_string(result_str, 2, B, Seller);
    break;
  } else if (has_label(&label, "retry")) { continue;
  } else if (has_label(&label, "quit")) { break;
} }
```

- C [TOOLS’12], OCaml [CSF’09], Java [COORD’10], others...

Conclusion

- Scribble adapts MPST to practical distributed application development
  - Global protocol specification and validation
  - Local projection and FSM generation
  - Conversation API and runtime endpoint monitoring
- Many future directions
  - Extending Scribble/MPST to capture additional forms of interaction
  - Integrating Scribble with other specification/programming techniques
  - Driven by use cases
- Reference list (from p18): `http://mrg.doc.ic.ac.uk/presentations/tgc13/August13.pdf`
- `https://github.com/scribble` (demo'd tools not fully available just yet but soon)

Binary Session Types Reading

- Honda, Vasconcelos and Kubo. *Language Primitives and Type Discipline for Structured Communication-Based Programming.* In European Symposium on Computing, volume 1381 of LNCS, pages 122–138. Springer, 1998.
- Gay and Hole. *Subtyping for session types in the pi calculus.* Acta Informatica, 42(2/3):191–225, 2005.
- Vasconcelos. *Fundamentals of Session Types. Information and Computation.* Elsevier, 217:52–70, 2012.
- SePi, A pi-calculus based language with linearly refined session types, `http://gloss.di.fc.ul.pt/sepi/`
- Caires, Pfenning and Toninho. *Linear logic propositions as session types.* Mathematical Structures in Computer Science, 2013. To appear.

MPST Reading

- *Multiparty asynchronous session types*. Honda, Yoshida and Carbone. POPL 2008
- *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, De Luca, Dezani-Ciancaglini and Yoshida. CONCUR 2008
- *Scribbling interactions with a formal foundation*. Honda, Mukhamedov, Brown, Chen and Yoshida. ICDCIT 2011
- *Asynchronous distributed monitoring for multiparty session enforcement*. Chen, Bocchi, Denilou, Honda and Yoshida. TGC 2011
- *Structuring communication with session types*. Honda, Hu, Neykova, Chen, Demangeon, Denilou and Yoshida. COB 2012

MPST Reading

- *Multiparty Session Types Meet Communicating Automata*. Deniélou and Yoshida. ESOP 2012
- *Monitoring networks through multiparty session types*. Bocchi, Chen, Demangeon, Honda and Yoshida. FMOODS 2013
- *Practical Interruptible Conversations – Distributed Dynamic Verification with Session Types and Python.* Hu, Neykova, Yoshida and Demangeon. RV 2013

- More references (from p18):
`http://mrg.doc.ic.ac.uk/presentations/tgc13/August13.pdf`