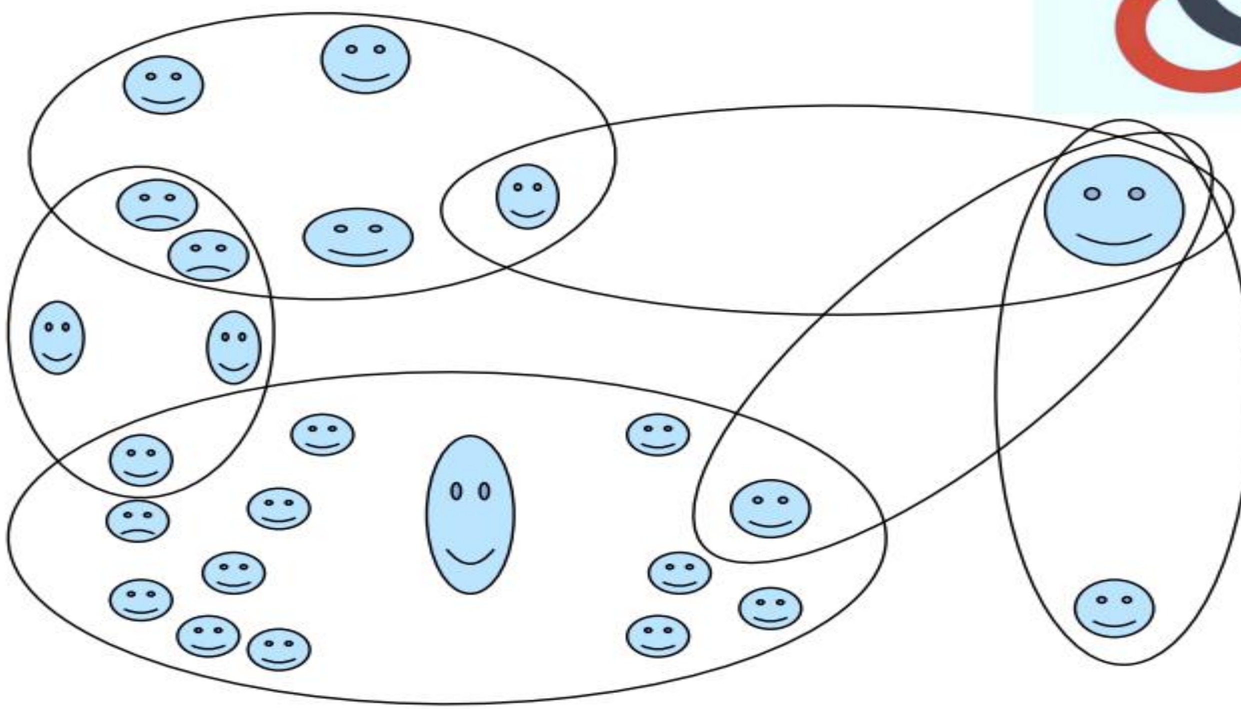
Multiparty session types and their applications in large distributed systems

Rumyana Neykova

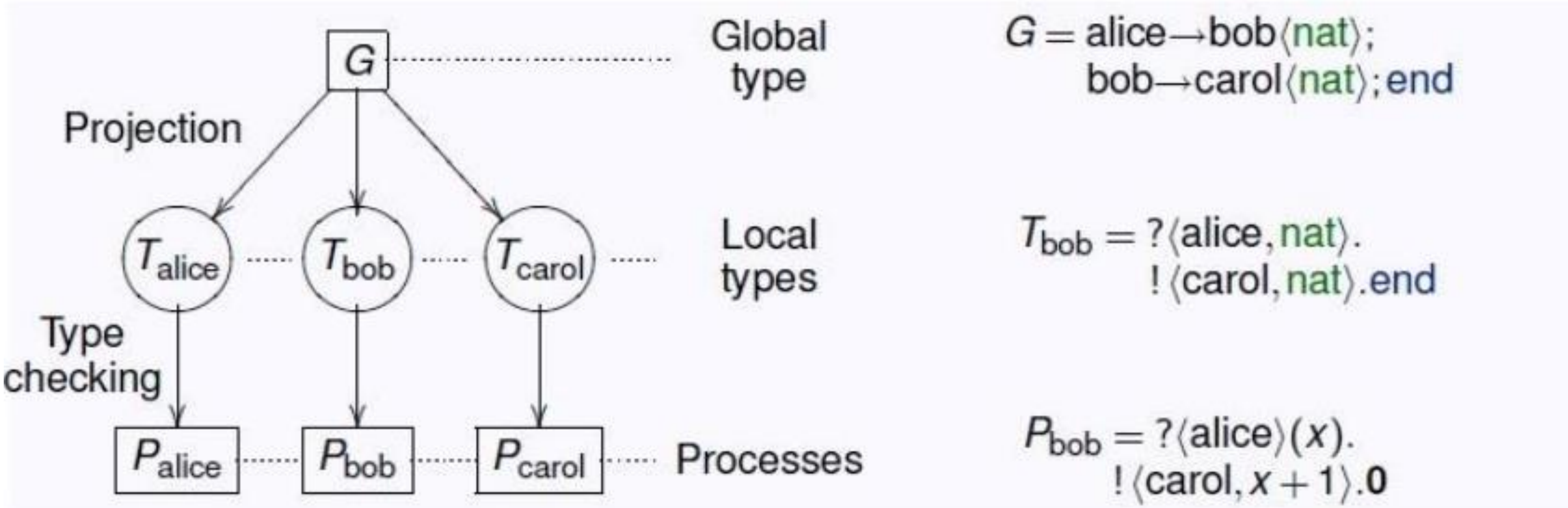
Multiparty Session Types: Concepts

- Separate the communication into conversations (sessions)



- Each process plays a role in a conversation => its type is defined by the conversation and its role

Standard Multiparty Session Types



- 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]

⬇

- Session Types with Assertions [Concur'11]

⬇

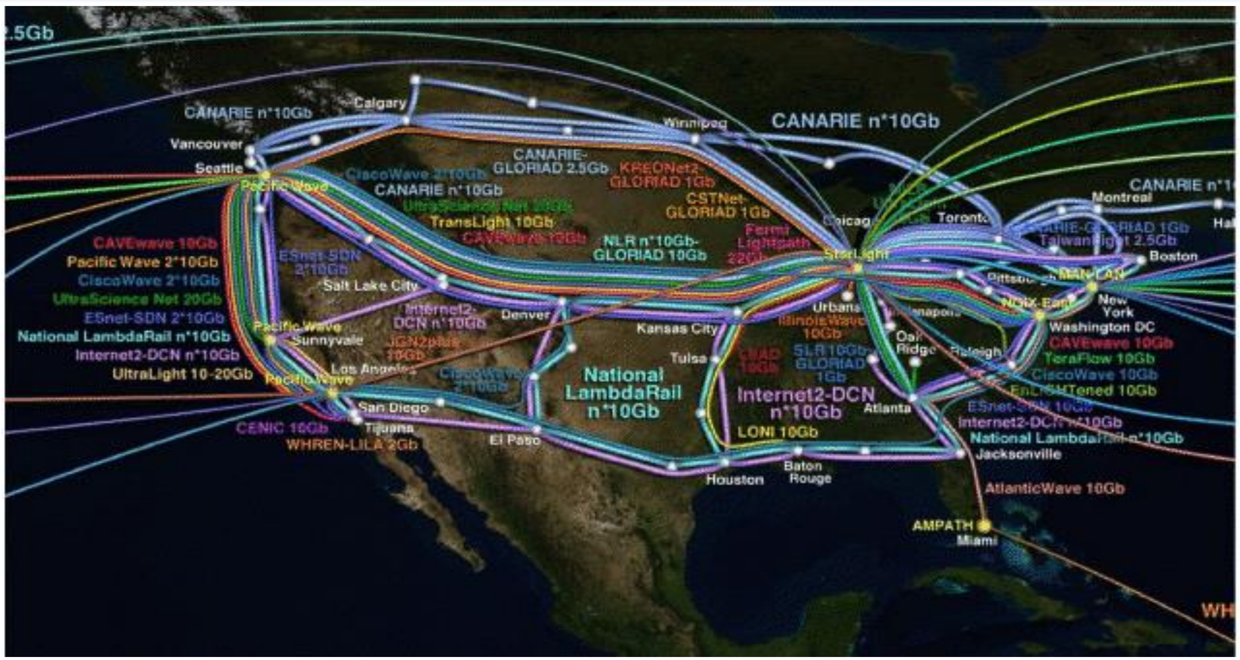
- Network Monitoring through Multiparty Session Types [FMOODS'13]

⬇

- Local Verification of Global Protocols, Practical Interruptible conversations [RV'13]

Ocean Observatory Initiative (OOI)

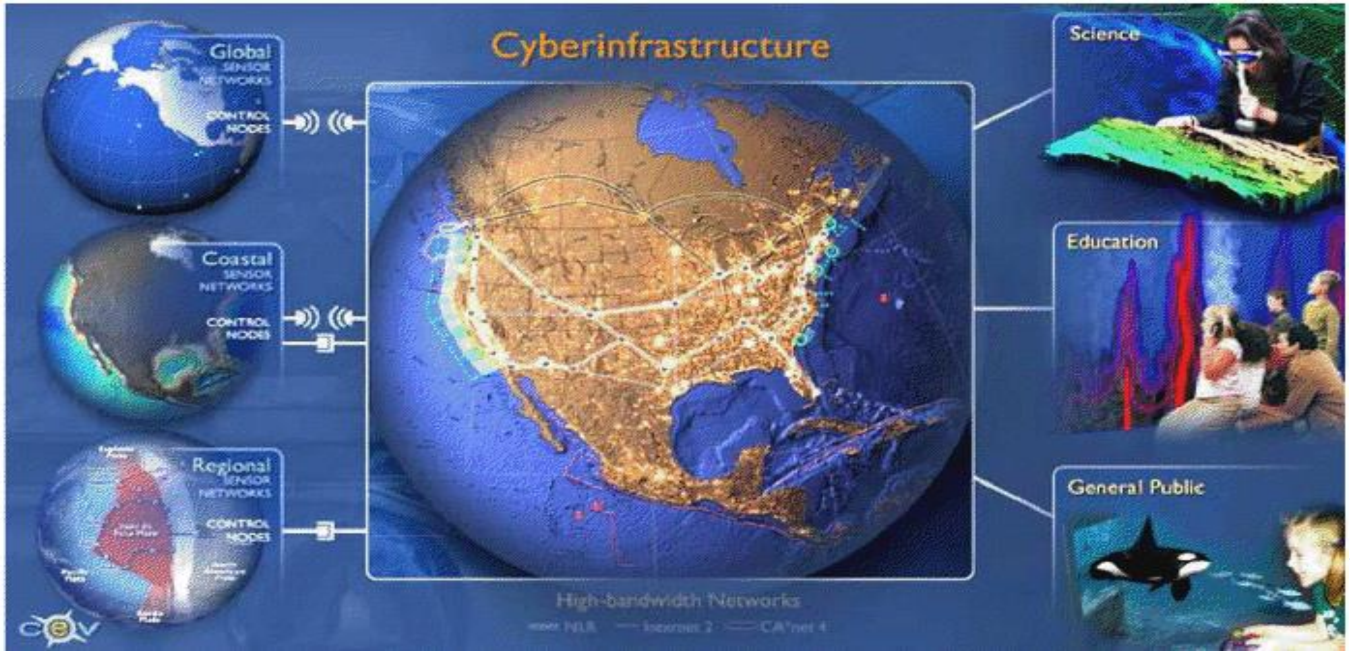
OOI aims: to deploy an infrastructure (global network) to expand the scientists' ability to remotely study the ocean



Usage: Integrate real-time data acquisition, processing and data storage for ocean research,…

Case Study: OOI

- OOI requirements
  - applications written in **different** languages, running on **heterogeneous** hardware in an **asynchronous** network.
  - different authentication domains, external **untrusted** applications
  - requires correct, safe interactions



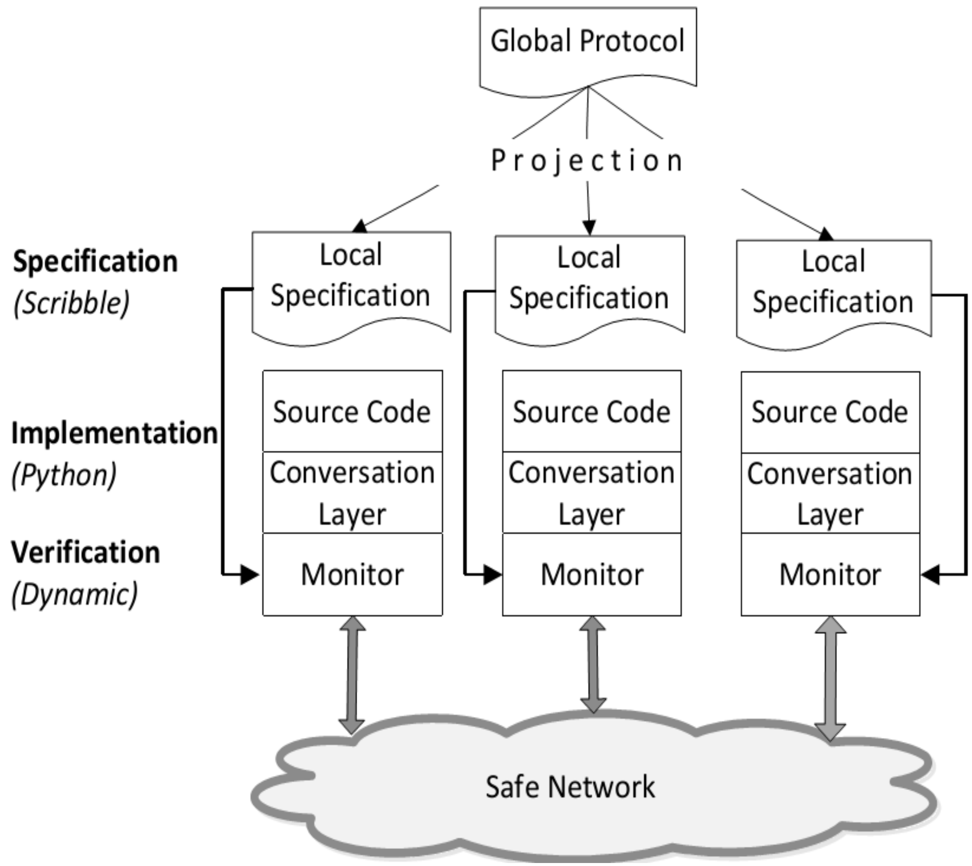
Session Types for Monitoring

- Distributed monitoring
  - attach a monitor to each application
  - the monitor checks messages w.r.t specification
  - ensures interoperablity

Session types for monitoring

- Adapting MPST theory to monitoring
- Principals
  - Developers design

  protocols in a dedicated language - Scribble
  - Well-fomedness is checked by Scribble tools
  - Protocols are projected into local types
  - Local types generate monitors

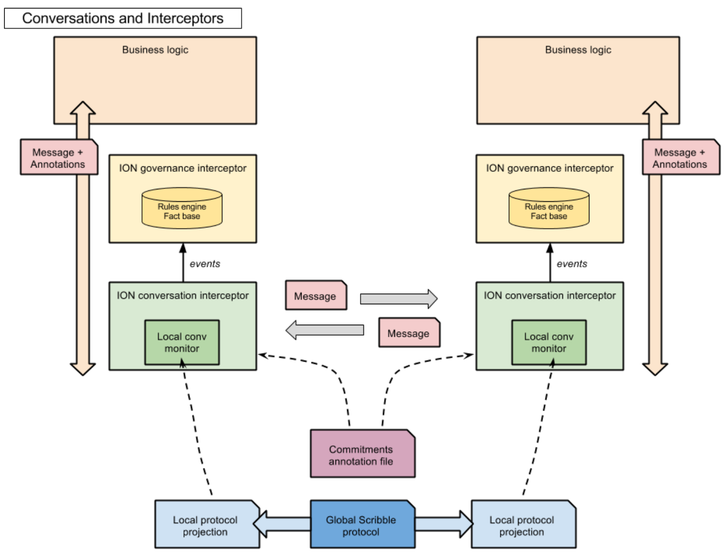


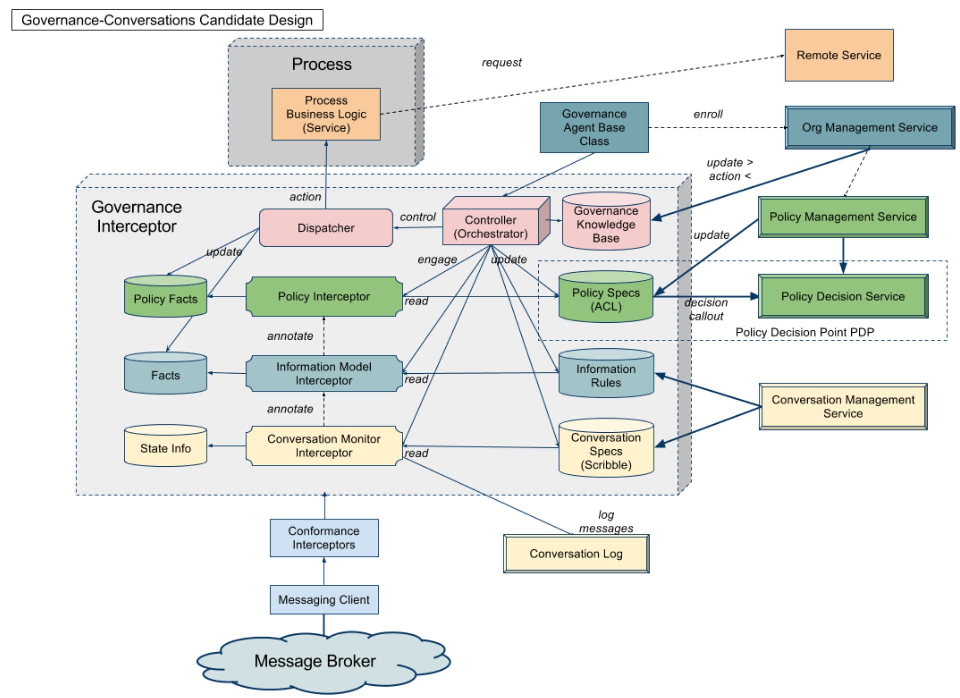
OOI Requirements - revisited

- Communication based on various protocols
  - General protocol verification **monitor**
- Heterogeneous systems
  - protocol description language - **Scribble**
- Different authentication domains
  - distributed monitoring
- Can we guarantee safety properties
  - a theory for network monitoring with soundness theorems

OOI Governance Framework Design

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





www.scribble.org

Scribble

Protocol Language

Follow me on GitHub

"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.

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 a meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send their data, or whether the other party is ready to receive a datum it is sending. In fact it is not clear what kinds of data is to be used for each interaction. It is too costly to carry out communications based on guess works and with inevitable communication mismatch (synchronisation bugs). Simply, it is not feasible as an engineering practice.

Documents

- Protocol Language Guide

Downloads

- Java Tools

Community

- Discussion Forum
- Java Tools
  - Issues
  - Wiki
- Python Tools
  - Issues
  - Wiki

Scribble Community

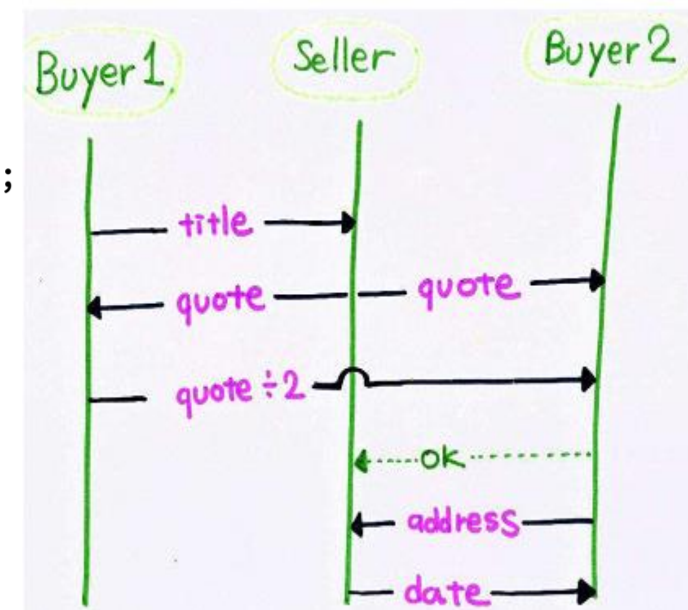
- Webpage:
  - www.scribble.org
- GitHub:
  - https://github.com/scribble
- Tutorial:
  - www.doc.ic.ac.uk/~rhu/scribble/tutorial.html
- Specification (0.3)
  - www.doc.ic.ac.uk/~rhu/scribble/langref.html

Two Buyer Protocol in Scribble

```
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;
}   }   }
```



Protocol Well-fomedness (choice)

```
global protocol Protocol1(role A, role B) {
    choice at A {
        m1() from A to B;
    } or {
        m2() from A to B; } }

global protocol Protocol2(role A, role B, role C) {
    choice at A {
        m1() from A to B;
        m1() from B to C; // Additional step
    } or {
        m2() from A to B; } }

global protocol Protocol3(role A, role B, role C) {
    choice at A {
        m1() from A to B;
        m1() from B to C;
    } or {
        m1() from A to B; // Copy-paste error
        m2() from B to C; } }
```

Buyer: A local projection

```
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;
} } }
```

The whole Picture

Global Conversation

(In Scribble)

```
protocol ListResources(client as C, registry as R){
  Request(resource_kind) from C to R;
  Response from R to C; }
```

Projection

Local Type

(In Scribble)

```
protocol ListResources at registry
                          (client as C){
  Request(resource_kind) from C;
  Response to C;}
```

```
protocol ListResources at client
              (registry as R){
  Request(resource_kind) to R;
  Response from R; }
```

Verification

Programs

(In Python)

```
p = Participant(name)
c = p.accept_invitation('ListResources',
                        'registry')
resource_kind = c.recv('client')
resource = get_resources(resource_kind)
c.send('client', resource)
```

```
p = Participant(name)
c = p.create_conversation('ListResources',
                          'client')
msg = 'Resource:asdf'
c.send('registry', msg)
resource = c.recv('registry')
```

It’s Demo time

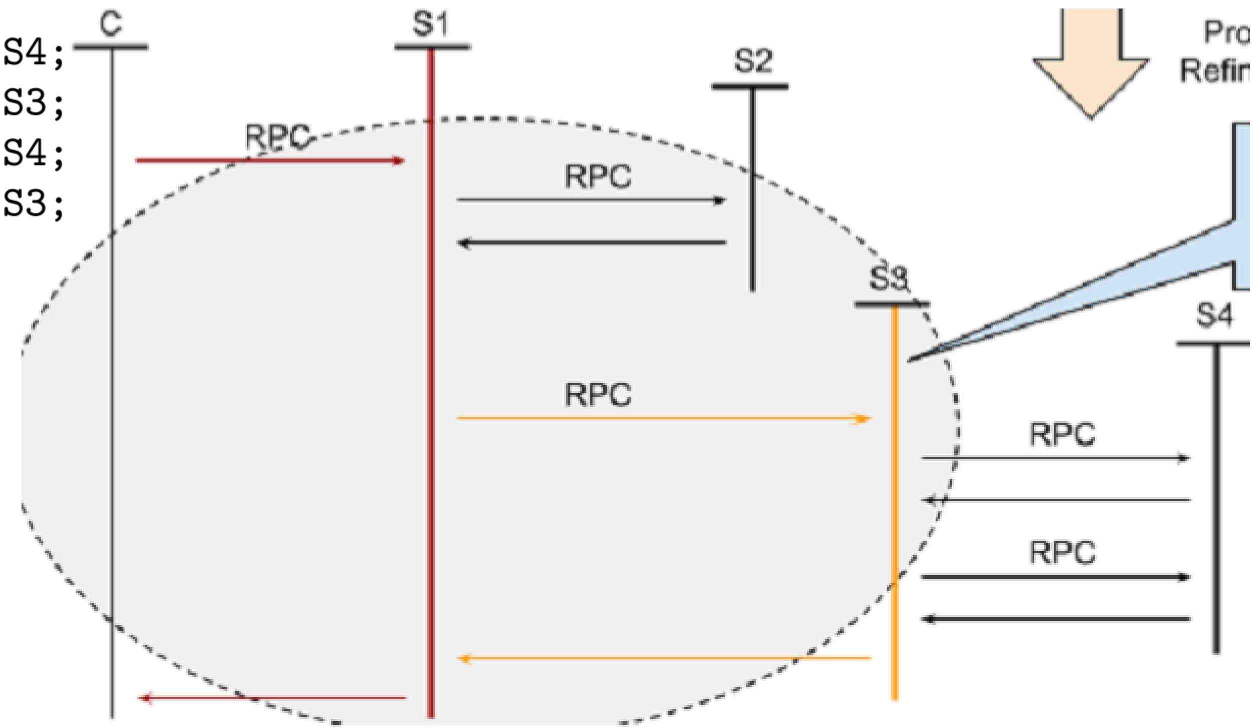
- Internal" CC Runtime component monitoring
- [DEMO]

More advanced protocols

- https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Governance+Framework
- Higher-level" application protocols
  - Composition of RPC calls
  - Negotiation protocol

Application-level service call composition

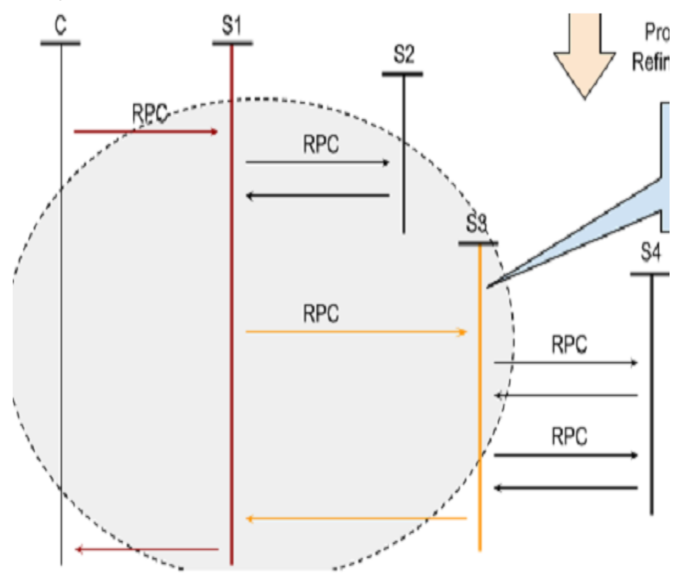
```
// Direct specification
global protocol P3(role C, role S1, role S2, role S3, role S4)
{
    () from C to S1;
        () from S1 to S2;
        () from S2 to S1;
        () from S1 to S3;
            () from S3 to S4;
            () from S4 to S3;
            () from S3 to S4;
            () from S4 to S3;
        () from S3 to S1;
    () from S1 to C;
}
```



Scoping

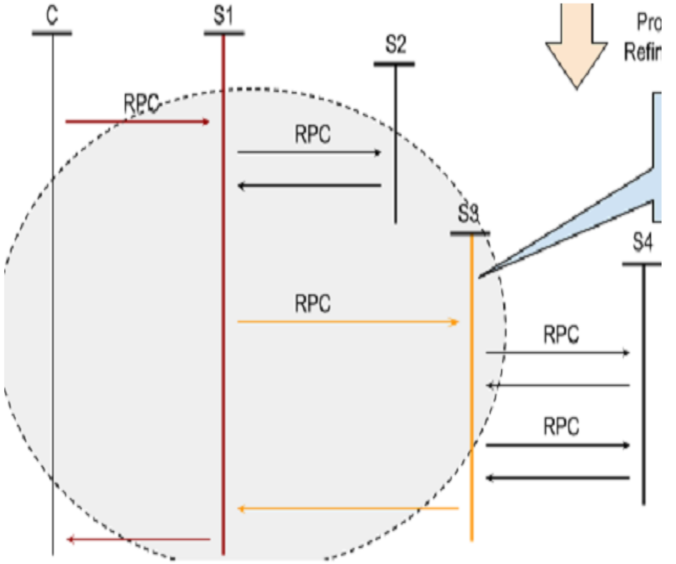
```
global protocol ServiceCall(role Client, role Service) {
    () from Client to Server;
    () from Server to Client;
}

// By composing basic ServiceCalls
global protocol P2(role C, role S1, role S2, role S3, role S4)
{
    () from C to S1;
      do ServiceCall(S1 as Client, S2 as Server);
      () from S1 to S3;
        do ServiceCall(S3 as Client, S4 as Server);
        do ServiceCall(S3 as Client, S4 as Serve
      () from S3 to S1;
    () from S1 to C;
}
```



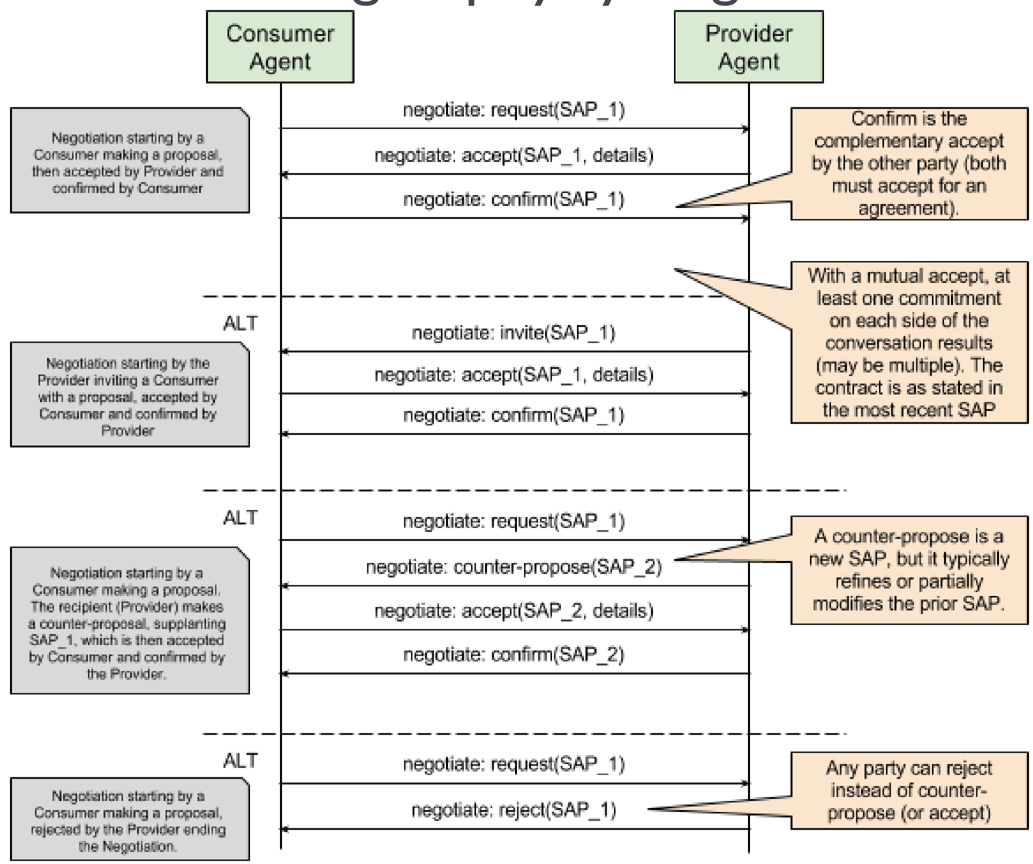
Scoping

```
// "Middleman" pattern
global protocol Middleman(
    role L, role M, role R, role S)
{
 () from L to M;
  do ServiceCall(M as Client, S as Server);
  do ServiceCall(M as Client, S as Server);
 () from M to R;
}
// By composing ServiceCall and Middleman patterns
global protocol P3(role C, role S1, role S2, role S3, role S4)
{
    () from C to S1;
        do ServiceCall(S1 as Client, S2 as Server);
        do Middleman(S1 as L, S3 as M, S4 as R);
    () from S1 to C;
}
```



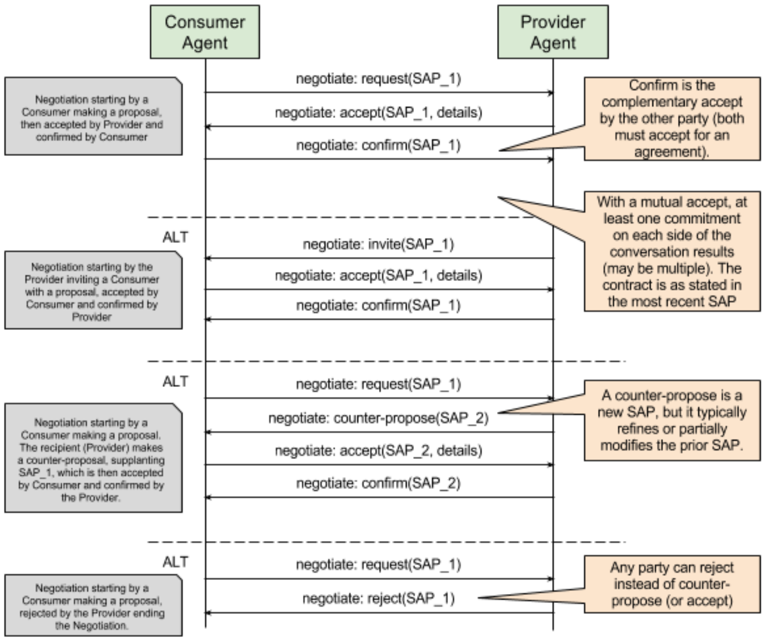
Agent Negotiation

- Provider and Consumer agents negotiate a Service Agreement Proposal
- https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Negotiate+Protocol



Negotiation protocol in Scribble

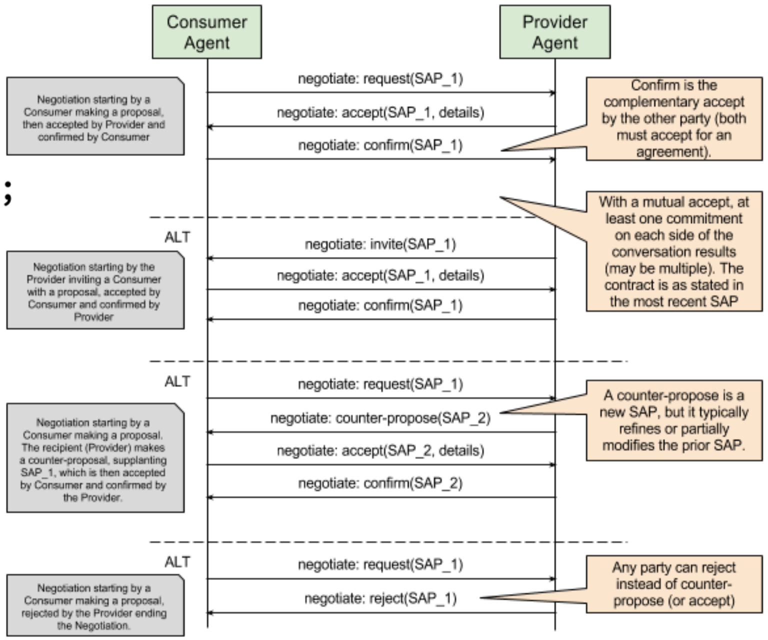
```
global protocol Negotiation1(role I, role C) {
    propose(SAP) from I to C;
    rec START {
        choice at C {
            accept() from C to I;
            confirm() from I to C;
        } or {
            propose(SAP) from C to I;
            choice at I {
                accept() from I to C;
                confirm() from C to I;
            } or {
                reject() from I to C;
            } or {
                propose(SAP) from I to C;
                continue START;
            }
        } or{
            reject() from C to I;
}   }   }
```



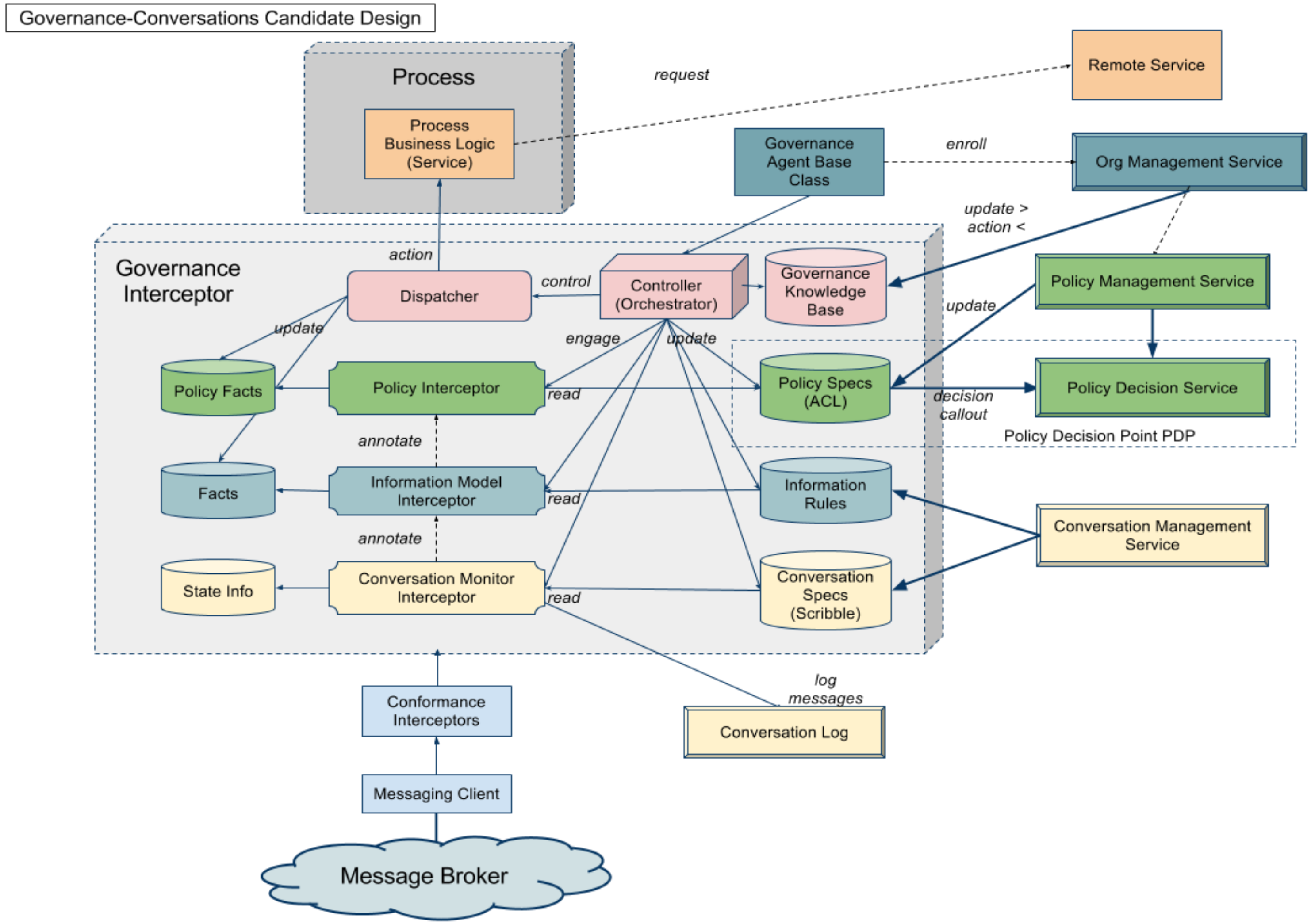
Negotiation protocol in Scribble

```
global protocol Negotiation2(role I, role C) {
    propose(SAP) from I to C;
    do NegotiationAux(I as I, C as C);
}

global protocol NegotiationAux(role I, role C) {
        choice at C {
            accept() from C to I;
            confirm() from I to C;
        } or {
            propose(SAP) from C to I;
            do NegotiationAux(C as I, I as C);
        } or{
            reject() from C to I;
        }
    }
}
```



Governance Framework



Scribble annotations

Annotations = Scribble Construct [Logic]

```
…
[Condition(payment>=1000)]
offer(payment: Integer) from C to I;
…
```

```
…
[defaultaccept]
offer(payment: Integer) from C to I;
…
```

```
…
[CreateCommitment(C, I, payment) at C]
offer(payment: Integer) from C to I;
…
```

- The monitor passes {'type':param, …} to the upper layers
- Upper layers recognize and process the annotation type or discard it

A theory for network monitoring

- Formalise MPST-monitoring and asynchronous networks.
- Introduce monitors as first-class objects in the theory
- Justify monitoring by soundness theorems.
  - Safety
    - monitors enforces specification conformance.
  - Transparency
    - monitors does not affect correct behaviours.
  - Fidelity
    - correspondence to global types is maintained.

Formalism: networks

$$N \quad ::= \quad [P]_\alpha \;\Big|\; N_1|N_2 \;\Big|\; \mathbf{0} \;\Big|\; (\nu a)N \;\Big|\; (\nu s)N \;\Big|\; \langle r \,;\, h \rangle$$

- Asynchronous networks composed of
  - processes P located at principals α
    - Abstracts local applications
  - router r
    - abstracts network routing information updated on-the-fly
  - global queue h
    - Abstracts messages in transit

Formalism: Monitor

- Specifications

$$\Sigma \ ::= \ \emptyset \ \mid \ \Sigma, \alpha : \langle \Gamma ; \Delta \rangle,$$

$$\Gamma ::= \emptyset \ \mid \ \Gamma, a :?(T[\mathtt{r}]) \mid \Gamma, a :!(T[\mathtt{r}]) \quad \Delta ::= \emptyset \ \mid \ \Delta, s[\mathtt{r}] : T,$$

Σ: spec., Δ: session env, Γ: shared env.

- Monitors

$$\mathsf{M} = \alpha : \langle \Gamma ; \Delta \rangle$$

  - Monitors are introduced as component of monitored networks

$$\frac{\mathsf{M} \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle} \mathsf{M}' \quad r(s[\mathtt{r}_2]) \neq \alpha}{[s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle]_\alpha \mid \mathsf{M} | \langle r \ ; \ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M}' | \langle r \ ; \ h \cdot s\langle \mathtt{r}_1,\mathtt{r}_2, l\langle v\rangle\rangle\rangle}$$

$$\frac{\mathsf{M} \overset{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle}{\not\longrightarrow}}{[s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle]_\alpha \mid \mathsf{M} \mid \langle r \ ; \ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M} \mid \langle r \ ; \ h\rangle}$$

Result

- Local Safety

$$\models [P]_\alpha \mid \mathsf{M} : \alpha : \langle \Gamma ; \Delta \rangle \text{ with } \mathsf{M} = \alpha : \langle \Gamma ; \Delta \rangle.$$

  - a monitored process satisfies its specification
- Global Safety

  If N is fully monitored w.r.t. Σ, then $\models \mathsf{N} : \Sigma$.

  - monitored network behaves as expected
- Session Fidelity
  - a configuration is consistent when it corresponds to a well-formed array of global types through projection
  - consistent is preserved by reduction
  - At any time, the network corresponds to a well-formed specification

Summary

- Having a context allows to control the communication
- Having granularity allows to specify constraints on the interactions
- Early error detection is much cheaper
- High-level policies on top of protocol verification
- Good abstraction means easy programming – you program with send and receive (no threads, sockets, channels)

References

- http://www.youtube.com/watch?feature=endscreen&v=mrEiwd9Buxk&NR=1
- https://confluence.oceanobservatories.org/download/attachments/18351011/OOI+CyberInfrastructure+-+Next+Generation+Oceanographic+Research-lowres.pdf?version=1&modificationDate=1246912767000

It is your turn …

```
protocol Q&A(you, me)
{
   rec Loop
   {
        Questions from you to me;
        Answers from me to you;
        Loop;
   }
}
```