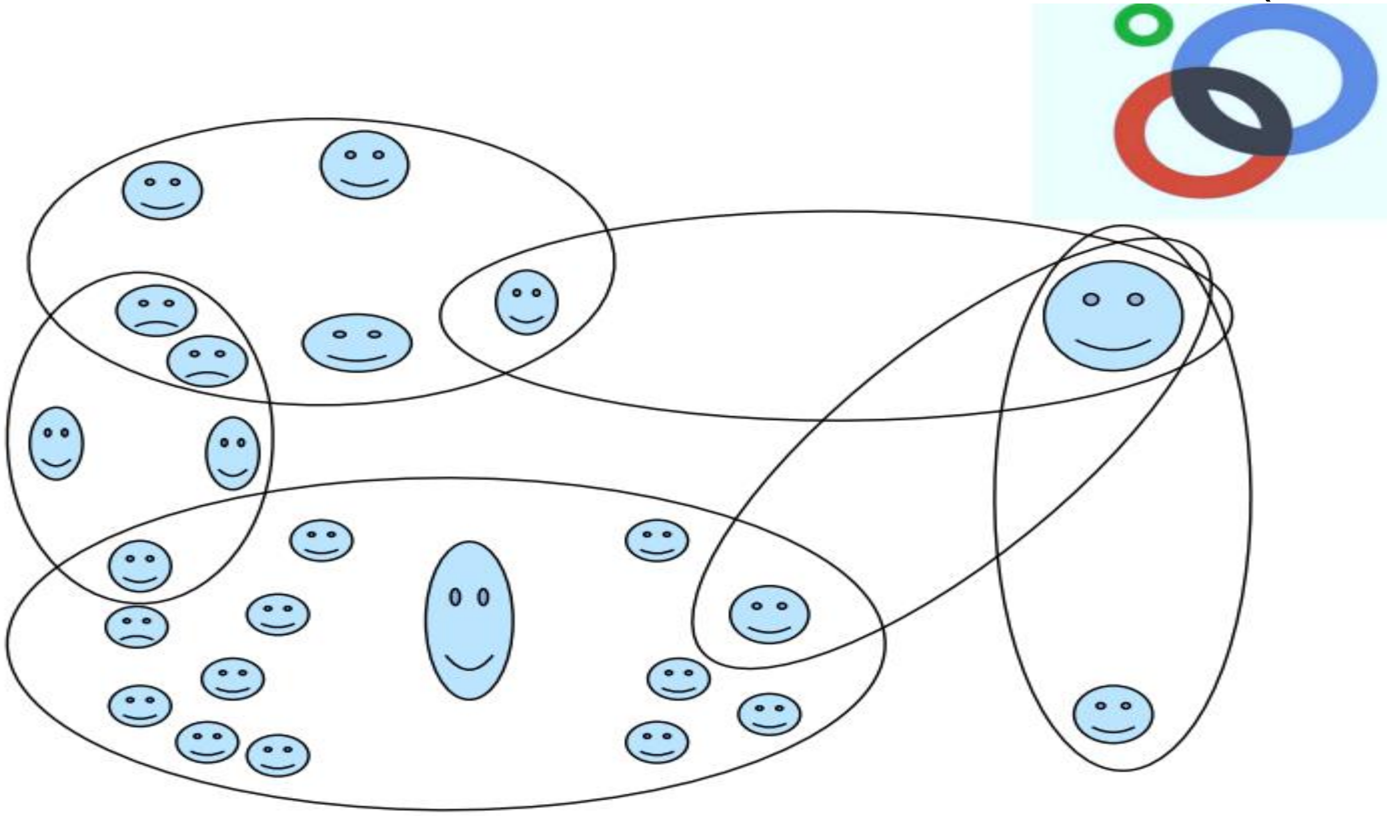
Multiparty Session Types: Concepts

- Separate the communication into conversations (sessions)



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

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]

↓

- Monitoring Networks through Multiparty Session Types [TGC'12]

↓

- Multiparty Session Types Meet Communicating Automata [ESOP'12, ICALP'13]

↓

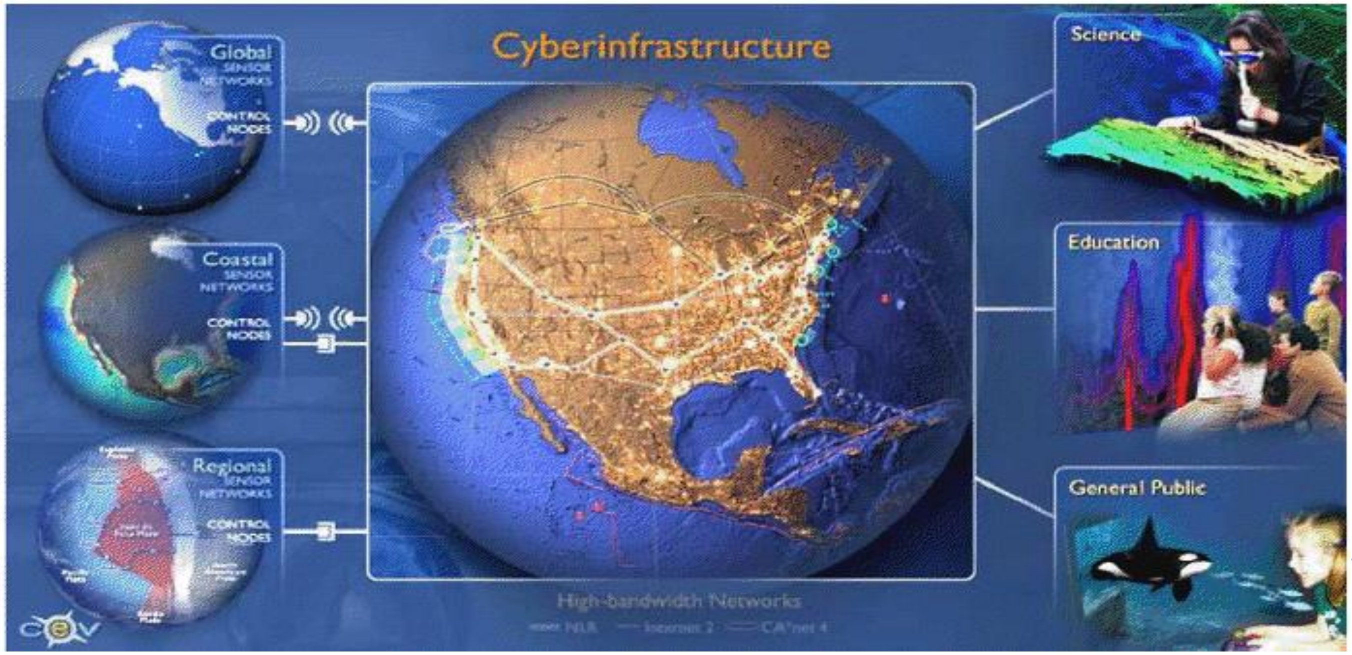
- Network Monitoring through Multiparty Session Types [FMOODS'13]

↓

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

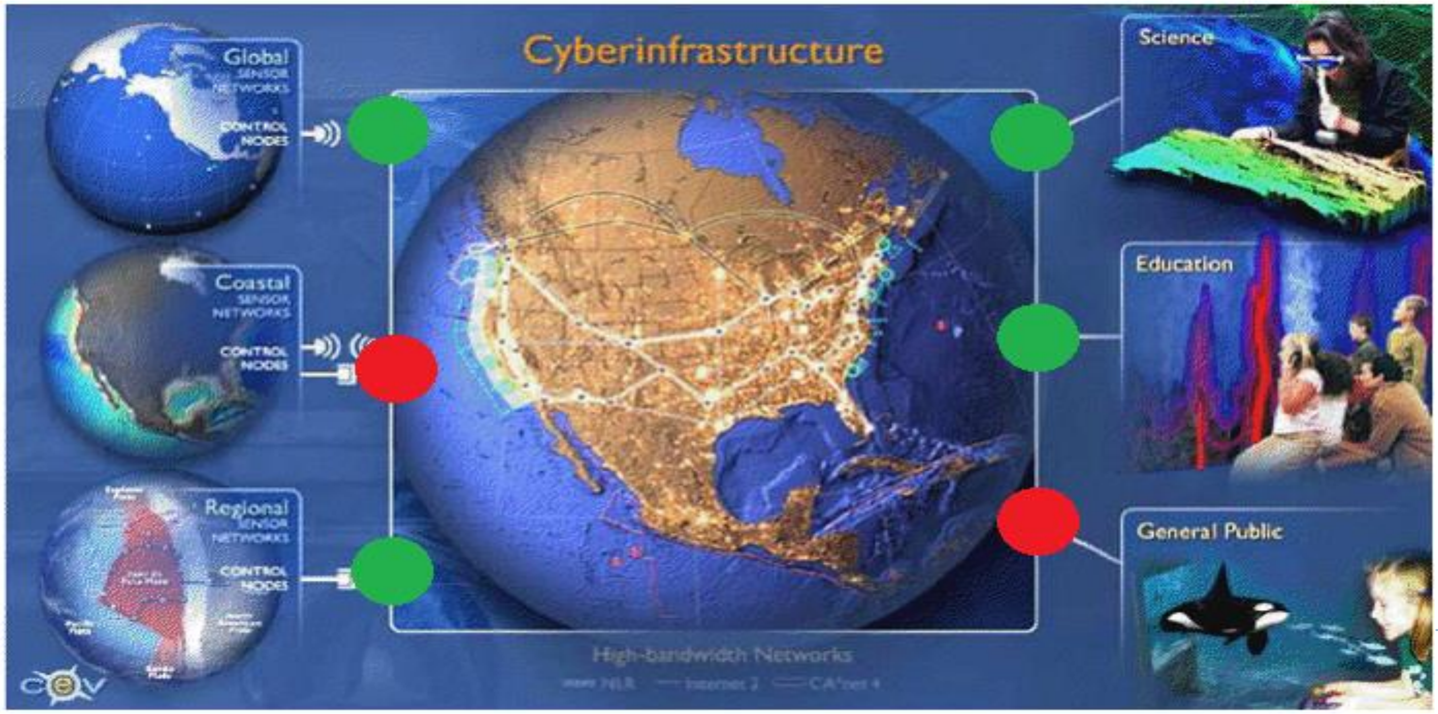
Case Study: OOI

- 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



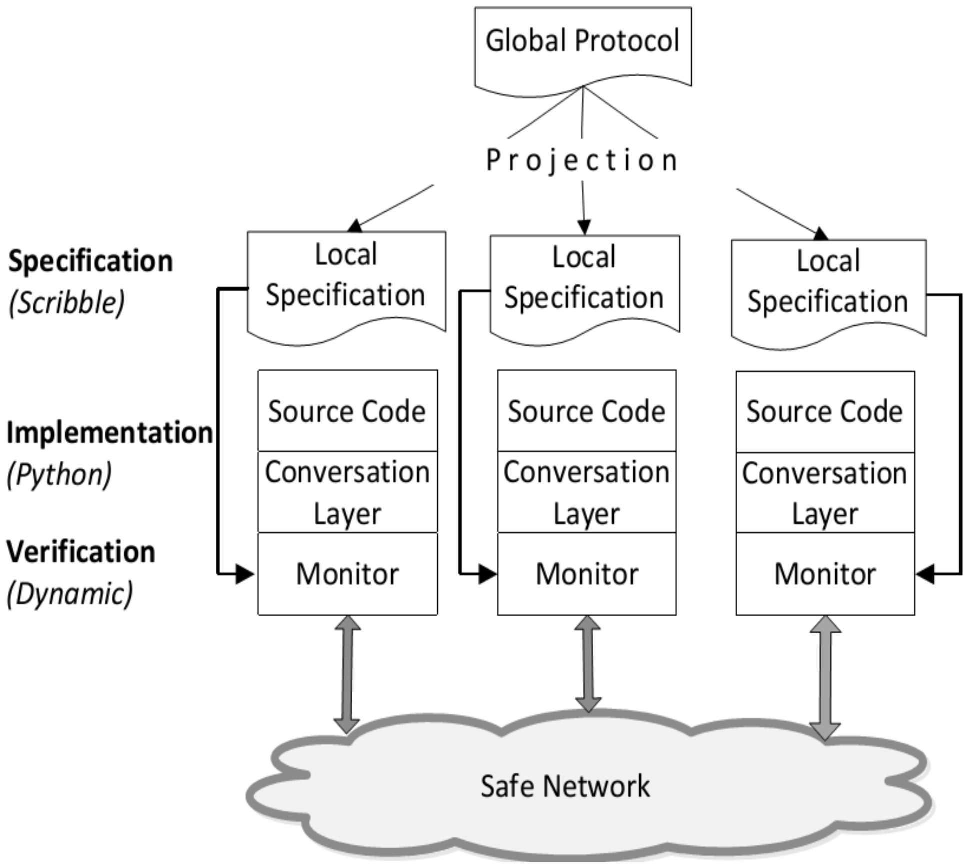
Session Types for Monitoring

- 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

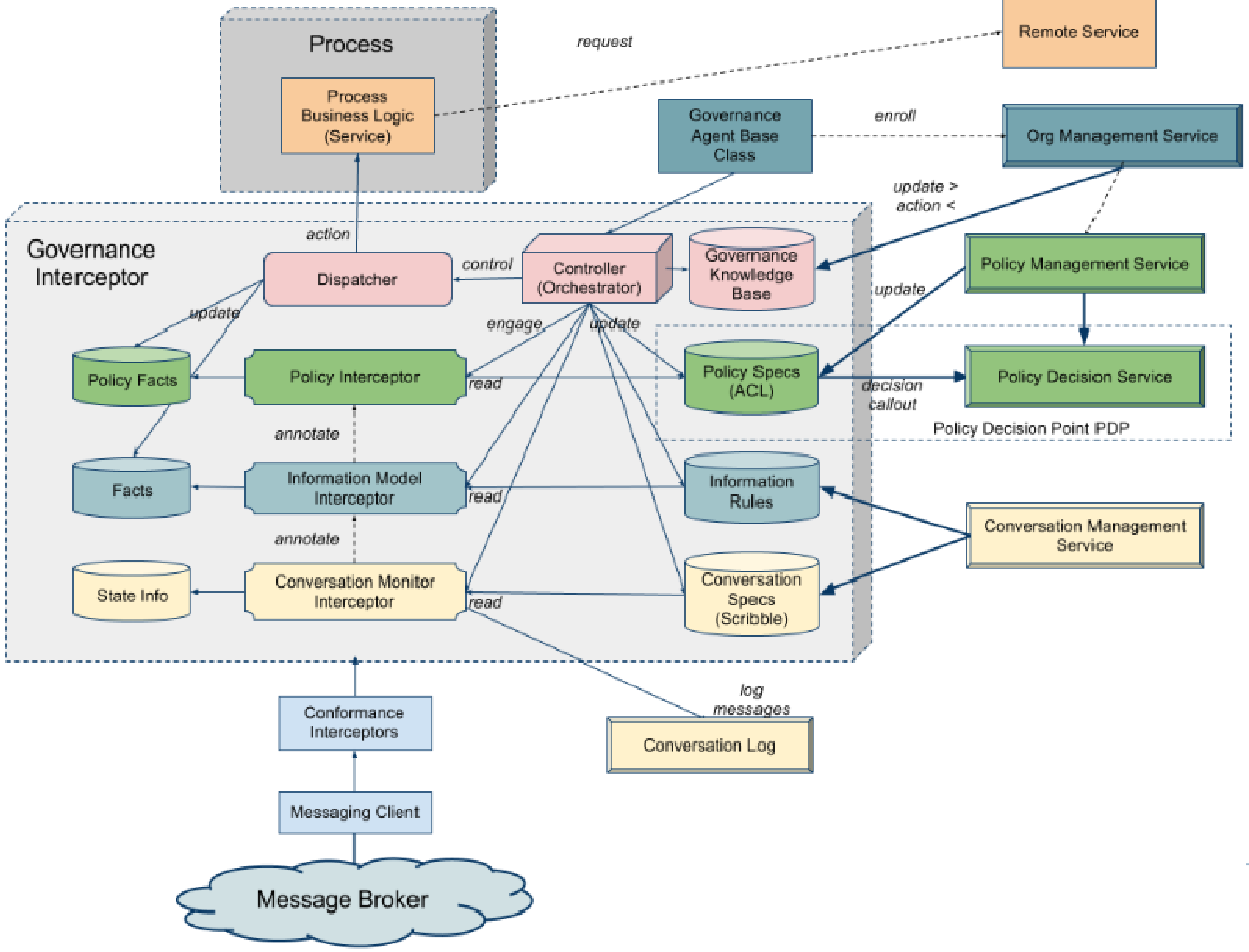
CIAD COI OV Governance Framework

5 Added by Michael Meisinger, last edited by Michael Meisinger on Oct 26, 2012 (view change)

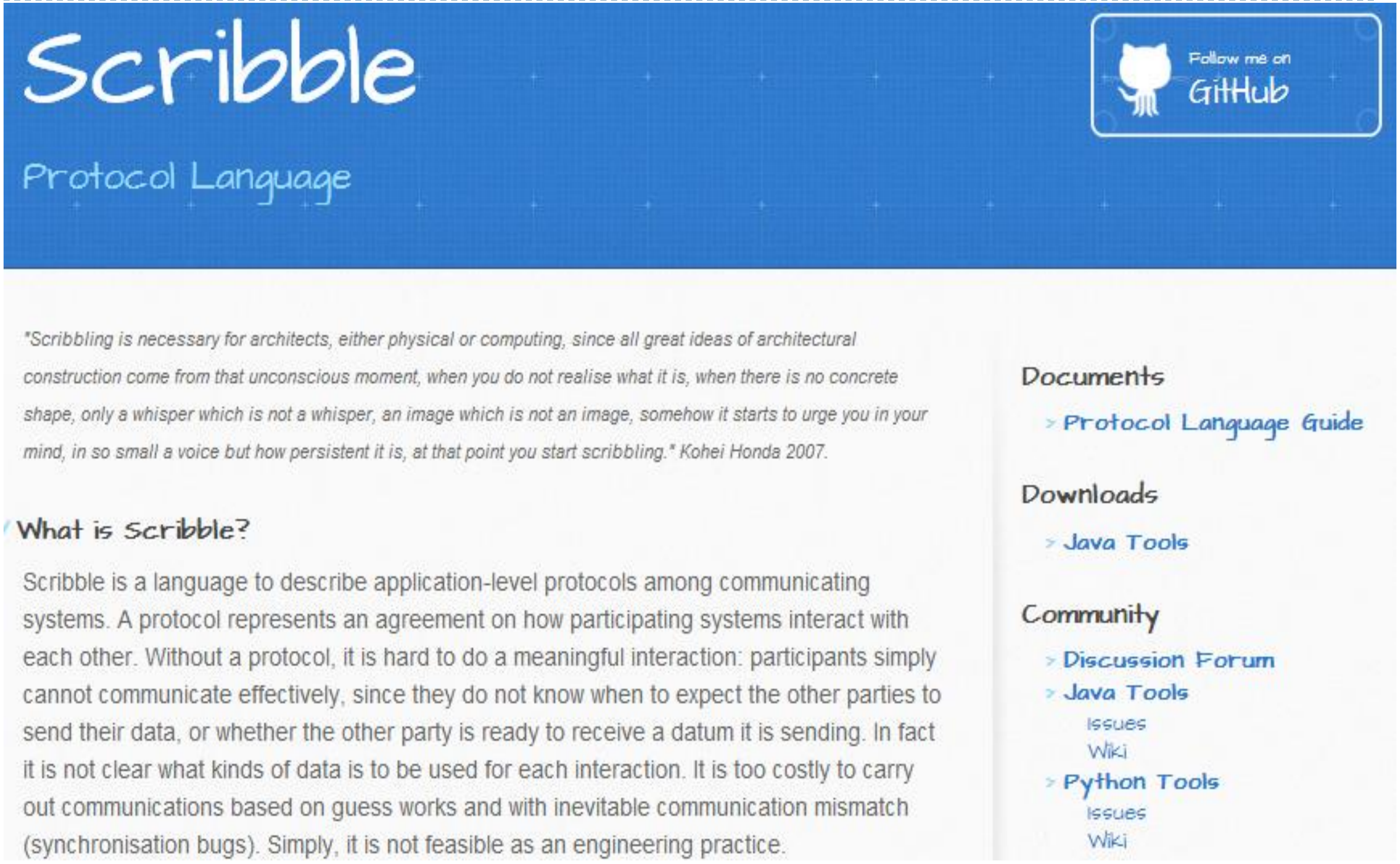
Tools

Governance is the term describing the ION capabilities of electronically tracking the relationships and commitments between the ION users, their agents and their resources. This knowledge is applied to enforce consistent system policy, track resource access and verify system interaction correctness.

Governance-Conversations Candidate Design



www.scribble.org



"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.

Scribble Community

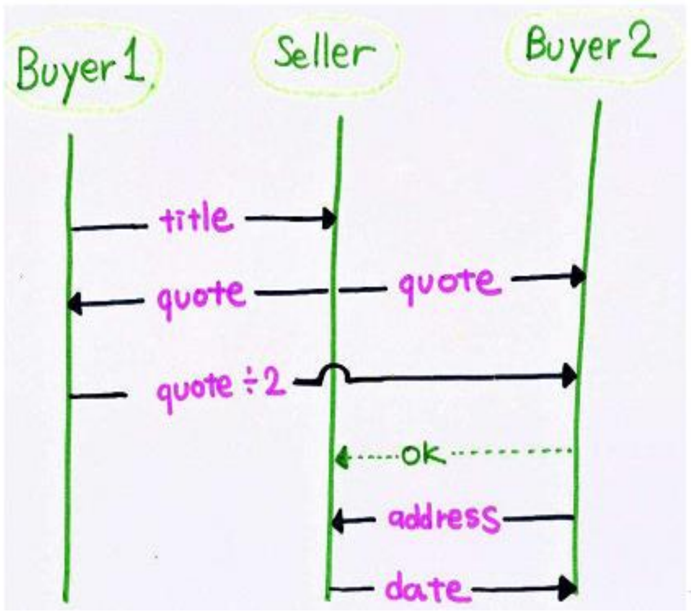
- Webpage:
  - www.scribble.org
- GitHub:
  - https://github.com/scribble
- Tutorial:
  - www.doc.ic.ac.uk/~rhu/scribble/tutorial.html
- Specification (0.3)
  - www.doc.ic.ac.uk/~rhu/scribble/langref.html

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; } }
```

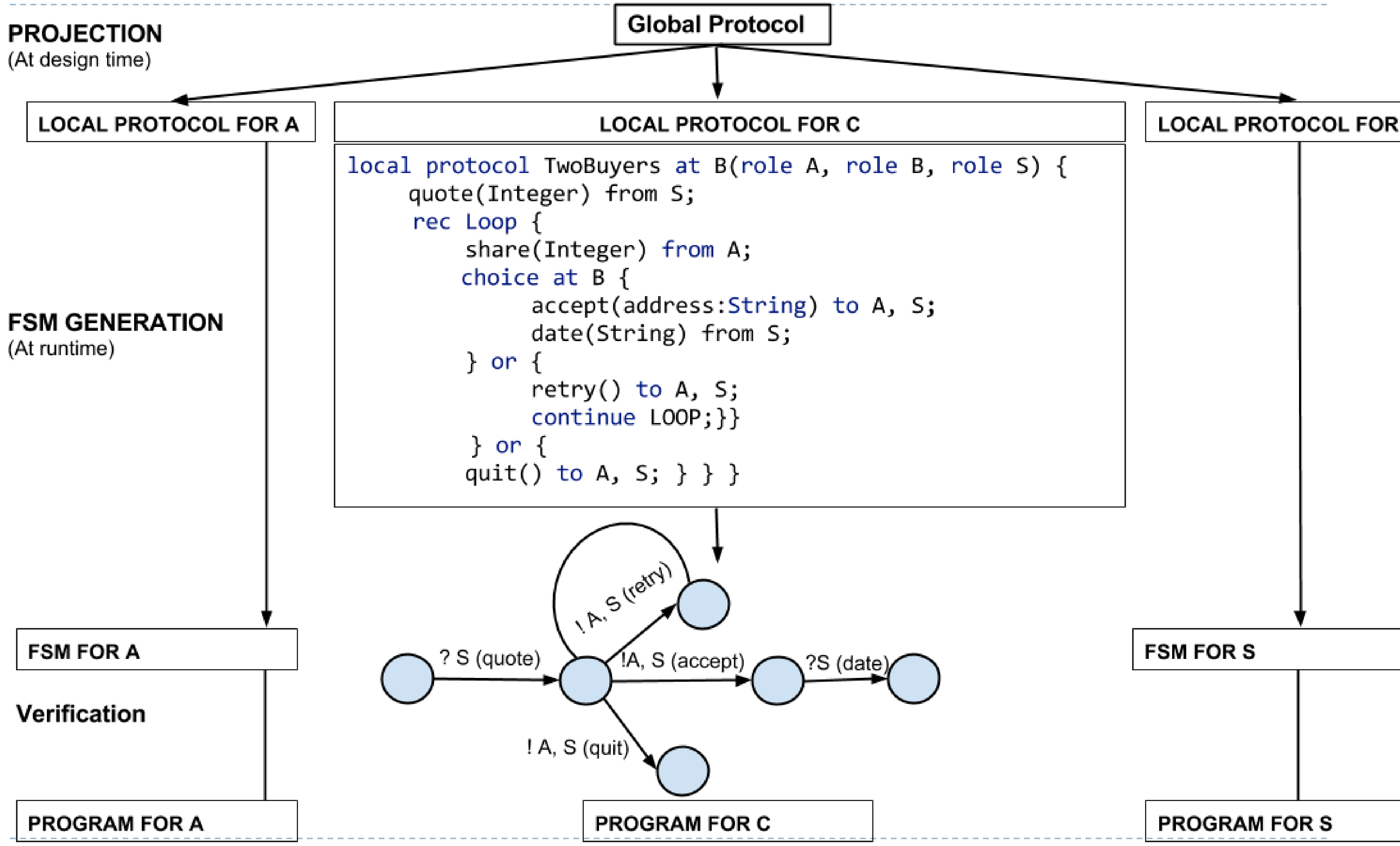
Buyer: A local projection

```
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



```
local protocol TwoBuyers at B(role A, role B, role S) {
     quote(Integer) from S;
     rec Loop {
          share(Integer) from A;
          choice at B {
                accept(address:String) to A, S;
                date(String) from S;
          } or {
                retry() to A, S;
                continue LOOP;}}
          } or {
          quit() to A, S; } } }
```

It’s Demo time

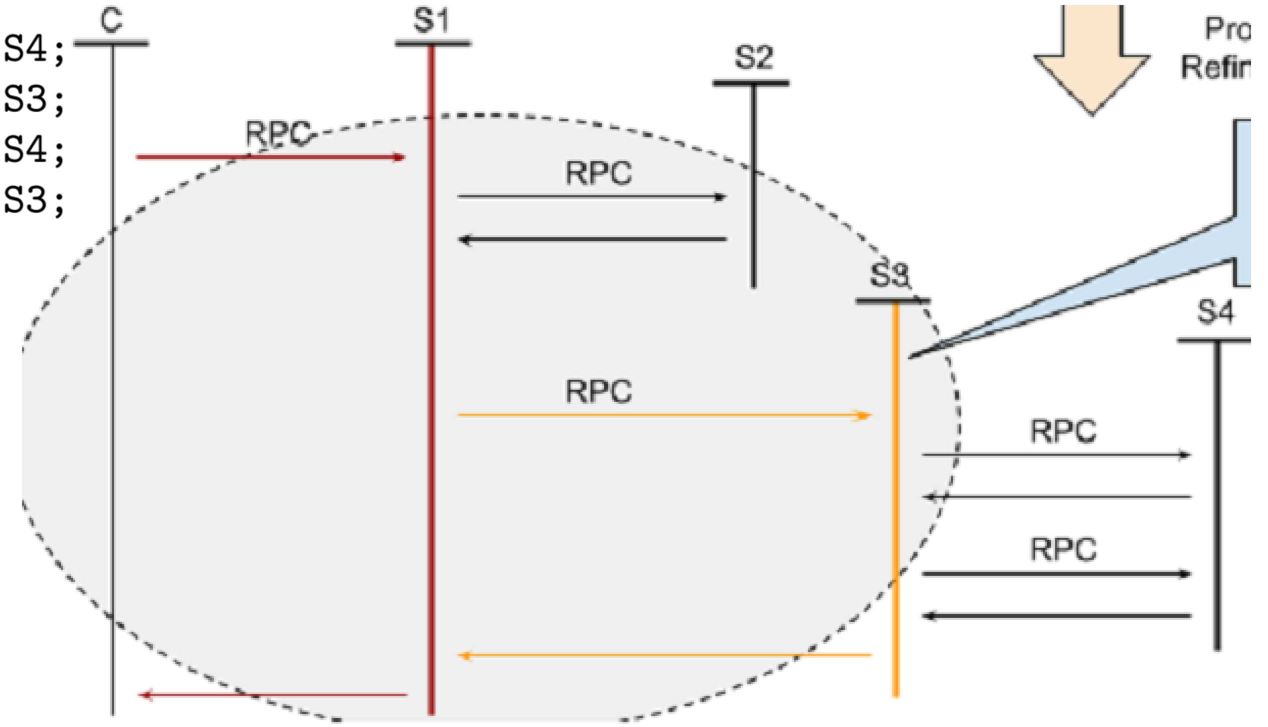
- Internal" CC Runtime component monitoring
- [DEMO]

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

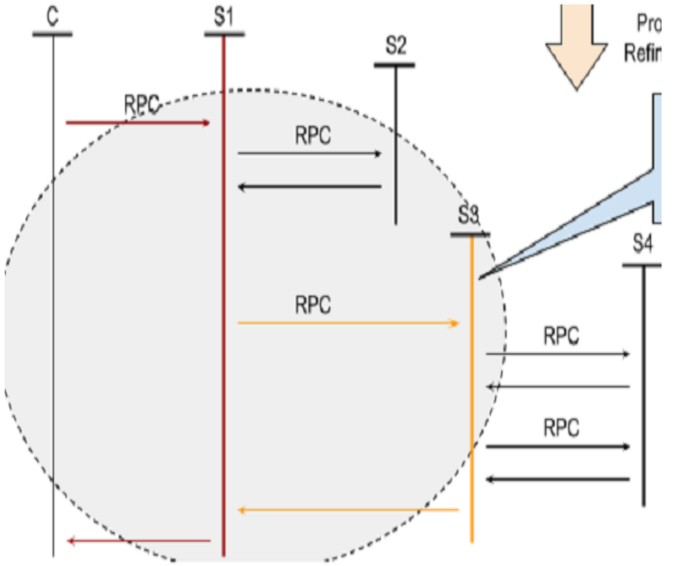
```
// Direct specification
global protocol P3(role C, role S1, role S2, role S3, role S4)
{
    () from C to S1;
        () from S1 to S2;
        () from S2 to S1;
        () from S1 to S3;
            () from S3 to S4;
            () from S4 to S3;
            () from S3 to S4;
            () from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



Scoping

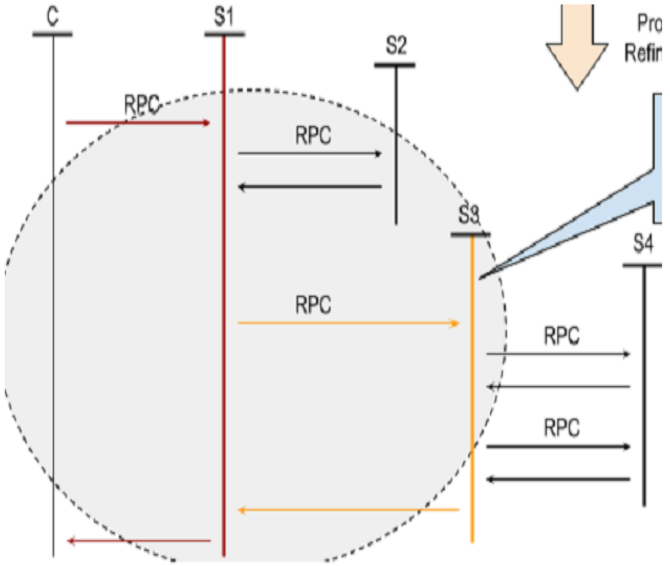
```
global protocol ServiceCall(role Client, role Service) {
    () from Client to Server;
    () from Server to Client;
}

// 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;
}
```



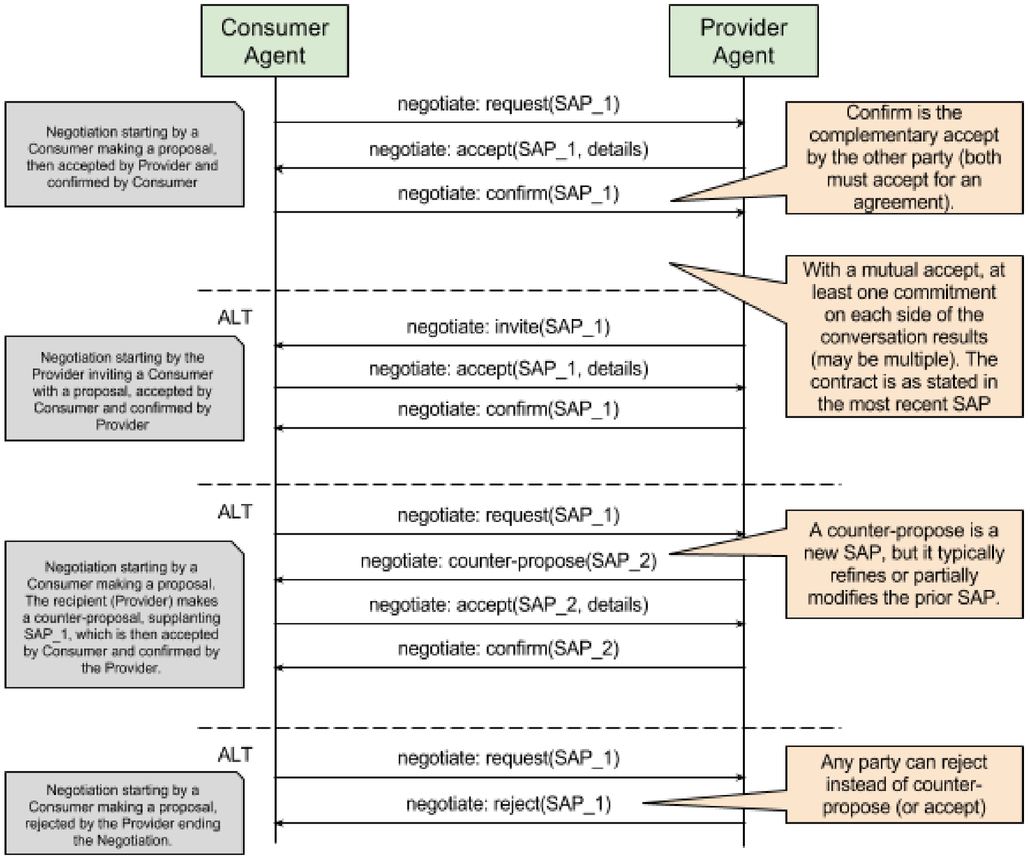
Scoping

```
// "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;
}
```



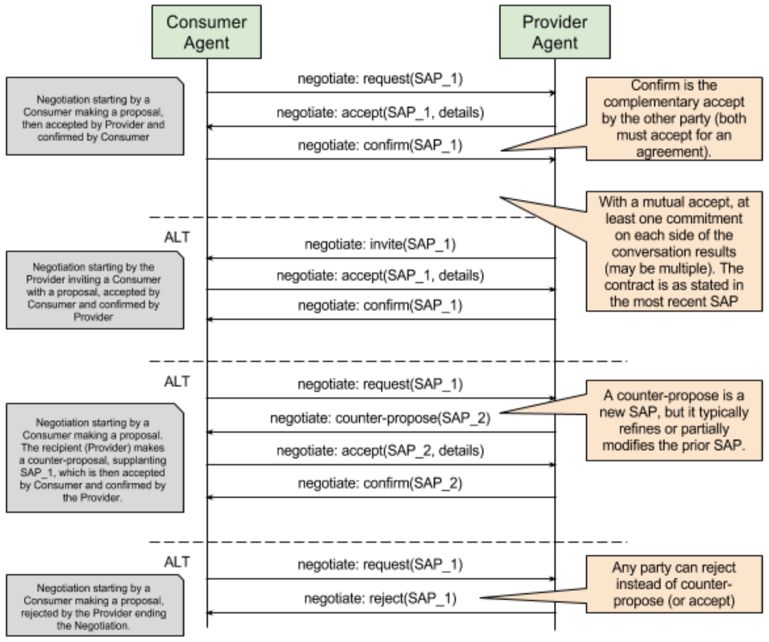
Agent Negotiation

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


Consumer Agent
Provider Agent
negotiate: request(SAP_1)
negotiate: accept(SAP_1, details)
negotiate: confirm(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
ALT
negotiate: invite(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
ALT
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. 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.
ALT
negotiate: request(SAP_1)
negotiate: reject(SAP_1)
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)

Negotiation protocol in Scribble

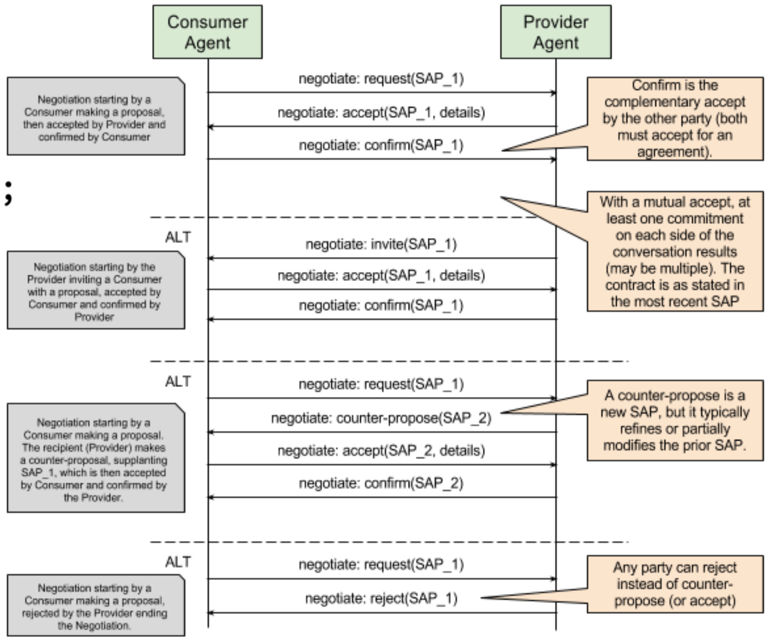
```
global protocol Negotiation1(role I, role C) {
    propose(SAP) from I to C;
    rec START {
        choice at C {
            accept() from C to I;
            confirm() from I to C;
        } or {
            propose(SAP) from C to I;
            choice at I {
                accept() from I to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


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)

Negotiation protocol in Scribble

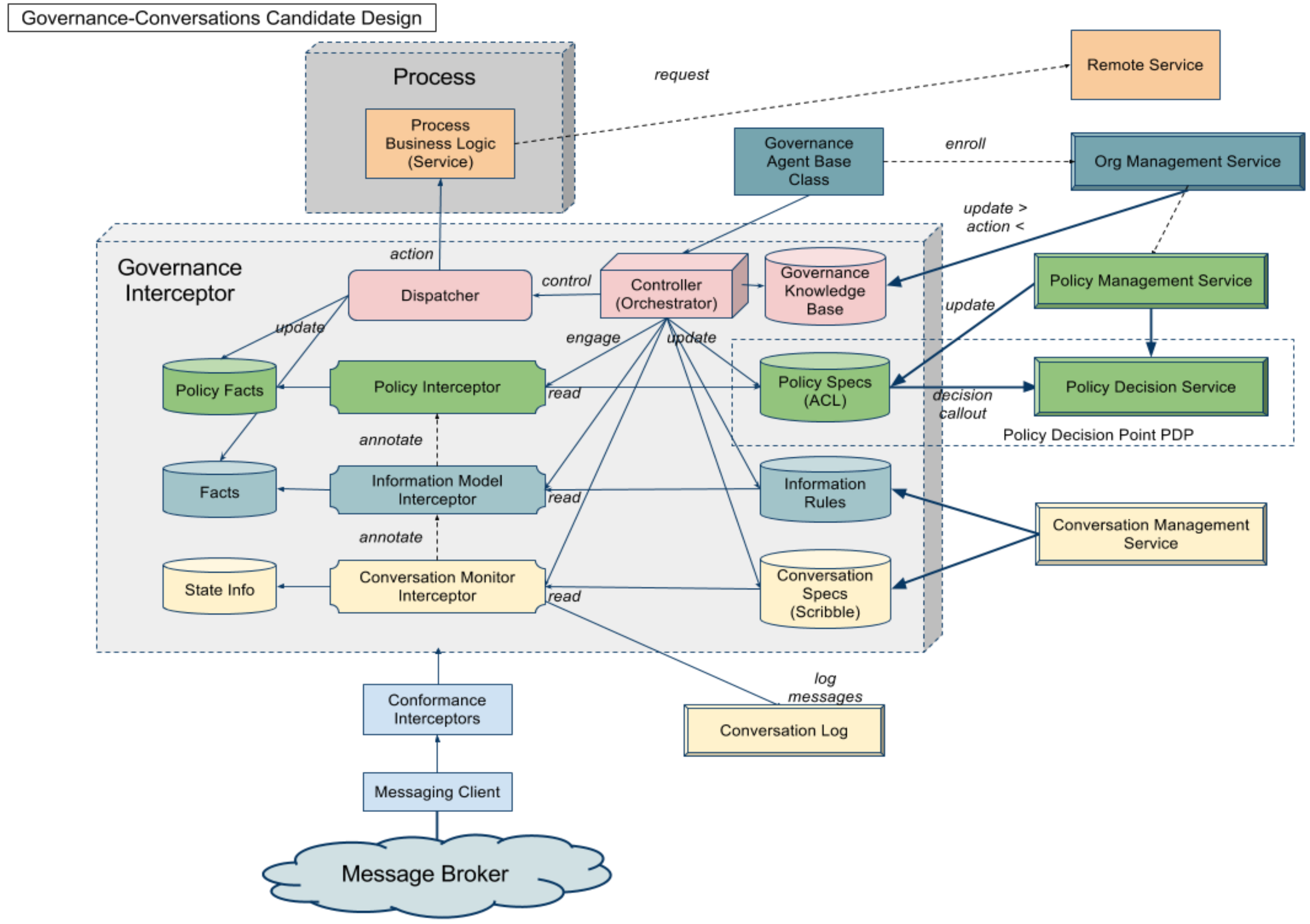
```
global protocol Negotiation2(role I, role C) {
    propose(SAP) from I to C;
    do NegotiationAux(I as I, C as C);
}

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;
        }
    }
}
```



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)

Governance Framework



Scribble annotations

Annotations = Scribble Construct @{Logic}

```
…
{assertion: payment>=1000}
offer(payment: int) from C to I;
…
```

```
…
@{deadline: 5s}
offer(payment:  int) from C to I;
…
```

```
…
@{commitment: create(C, I, payment)}
offer(payment: int) 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.

Multiparty Sessions for Runtime Monitors

$$
\begin{array}{lcl}
A & ::= & \mathsf{tt} \mid \mathsf{ff} \mid e_1 = e_2 \mid e_1 < e_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \\
e & ::= & v \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \; \mathtt{mod} \; e_2 \qquad S ::= \mathsf{bool} \mid \mathsf{int} \mid \mathsf{string} \\
G & ::= & \mathtt{r}_1 \rightarrow \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I} \mid G_1 \mid G_2 \mid G_1 ; G_2 \mid \mu \mathsf{t}.G \mid \mathsf{t} \mid \epsilon \mid \mathsf{end} \\
T & ::= & \mathtt{r}!\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid \mathtt{r}?\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid T_1 \mid T_2 \mid T_1 ; T_2 \mid \\
 & & \mu \mathsf{t}.T \mid \mathsf{t} \mid \epsilon \mid \mathsf{end} \\
P & ::= & \overline{a}\langle s[\mathtt{r}] : T\rangle \mid a(y[\mathtt{r}] : T).P \mid k[\mathtt{r}_1, \mathtt{r}_2]!l\langle e\rangle \mid k[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_{i \in I} \mid \\
 & & \mathsf{if}\; e \;\mathsf{then}\; P \;\mathsf{else}\; Q \mid P \mid Q \mid \mathbf{0} \mid \mu X.P \mid X \mid P ; Q \mid (\nu a)\, P \mid (\nu s) P \\
N & ::= & [P]_\alpha \mid N_1 | N_2 \mid \mathbf{0} \mid (\nu a)N \mid (\nu s)N \mid \langle r \; ; \; h\rangle \\
r & ::= & a \mapsto \alpha \mid s[\mathtt{r}] \mapsto \alpha \qquad h ::= m \cdot h \mid \emptyset \qquad m ::= \overline{a}\langle s[\mathtt{r}] : T\rangle \mid s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle
\end{array}
$$

Formal Semantics

$$
\begin{array}{lcl}
[\overline{a}\langle s[\mathtt{r}] : T\rangle]_\alpha \mid \langle r \,;\, h\rangle & \longrightarrow & [0]_\alpha \mid \langle r \,;\, h \cdot \overline{a}\langle s[\mathtt{r}] : T\rangle\rangle \\
[a(y[\mathtt{r}] : T).P]_\alpha \mid \langle r \,;\, \overline{a}\langle s[\mathtt{r}] : T\rangle \cdot h\rangle & \longrightarrow & [P[s/y]]_\alpha \mid \langle r \cdot s[\mathtt{r}] \mapsto \alpha \,;\, h\rangle \;\dagger \\
[s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v\rangle]_\alpha \mid \langle r \,;\, h\rangle & \longrightarrow & [0]_\alpha \mid \langle r \,;\, h \cdot s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle\rangle \;\dagger\dagger \\
[s[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \langle r \,;\, s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle \cdot h\rangle & \longrightarrow & [P_j[v/x_j]]_\alpha \mid \langle r \,;\, h\rangle \;\dagger\dagger\dagger
\end{array}
$$

$$\dagger : r(a) = \alpha \qquad \dagger\dagger : r(s[\mathtt{r}_2]) \neq \alpha \qquad \dagger\dagger\dagger : r(s[\mathtt{r}_2]) = \alpha$$

- processes P located at principals α
  - Abstracts local applications
- router r
  - abstracts network routing information updated on-the-fly

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

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

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

Satisfaction

The satisfaction relation $\models N : \Sigma$ relates networks and specification:

- if Σ expects an input, N should be able to process it.
- if N performs an output, Σ should be expecting it.
- still holds after reduction (coinductive definition).

Satisfaction equivalence

If $N_1 \cong N_2$ and $\models N_1 : \Sigma$ then $\models N_2 : \Sigma$.

Results (Safety)

Local Safety

$\models [P]_\alpha \mid \mathsf{M} : \alpha : \langle \Gamma; \Delta \rangle$ 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 networks behave as expected.

Results (Transparency)

Local Transparency

If $\models [P]_\alpha : \alpha : \langle \Gamma; \Delta \rangle$, then $[P]_\alpha \approx ([P]_\alpha \mid \mathrm{M})$ with $\mathrm{M} = \alpha : \langle \Gamma; \Delta \rangle$.

- unmonitored correct processes are undistinguishable from their monitored counterparts.
- allows one to mix monitored and typechecked processes.

Global Transparency

Assume N and N have the same global transport $\langle r \,;\, h \rangle$.
Assume:

1. N is fully monitored w.r.t. Σ and
2. $N = M \mid \langle r \,;\, h \rangle$ is unmonitored but $\models M : \Sigma$.

We have $\mathrm{N} \cong N$.

- monitors does not alterate behaviors of correct networks.
- monitor actions are not observable on correct components.

Results (Fidelity)

- a configuration is consistent: when it corresponds to a well-formed array of global types $(G_1, \ldots, G_n)$ through projection.
- conformance is satisfaction + receivability (queue can be emptied).

Session Fidelity

Assume:

1. configuration $\Sigma; \langle r \; ; \; h\rangle$ is consistent,
2. network $N \equiv M | \langle r \; ; \; h\rangle$ conforms to configuration $\Sigma; \langle r \; ; \; h\rangle$.

For any ℓ, whenever we have $N \xrightarrow{\ell}_{\mathsf{g}} N'$ s.t. $\Sigma; \langle r \; ; \; h\rangle \xrightarrow{\ell}_{\mathsf{g}} \Sigma'; \langle r' \; ; \; h'\rangle$, it holds that $\Sigma'; \langle r' \; ; \; h'\rangle$ is consistent and N' conforms to $\Sigma'; \langle r' \; ; \; h'\rangle$.

- consistence is preserved by reduction,
- at any time, the network correspond 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
- http://icmrg.herokuapp.com/