Scribbling Protocols Overview

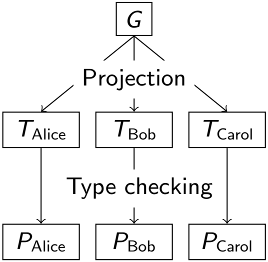
Specification and verification of distributed applications using multiparty session types

The Scribble team

Outline

- Background:
  - Multiparty session types (MPST)
  - The Scribble protocol language
  - Active use case project: Ocean Observatories Initiative
- Scribble by examples
  - Core constructs: message passing, choice, recursion, ...
  - Multiparty protocol validation (well-formedness)
  - Composing subprotocols; interruptible protocols
- Dynamic MPST verification
  - Decentralised runtime monitoring of conversation endpoints

Background 1/4: Multiparty Session Types (MPST)



- Global session type
  - $G = A \to B : m_1;\ B \to C : m_2;\ C \to A : m_3 \ldots$
- Local session types
  - Slice of global protocol relevant to each role
  - Mechanically derived from global protocol
  - $T_A = A!B : m_1;\ A?C : m_3; \ldots$
- Process language
  - Execution model of message passing actions by session participants
- (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 2/4: Scribble protocol description 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

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

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

Background 3/4: 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/OurApproach/WP_TestableArch.pdf`

Background 4/4: Ocean Observatories Initiative (OOI)

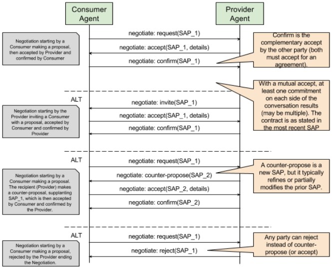
- NSF project ($400M, 5 years) to build a cyberinfrastruture for the acquisition and delivery of oceanography data

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
- Scribble in the OOI: specification, implementation and verification of service and application protocols

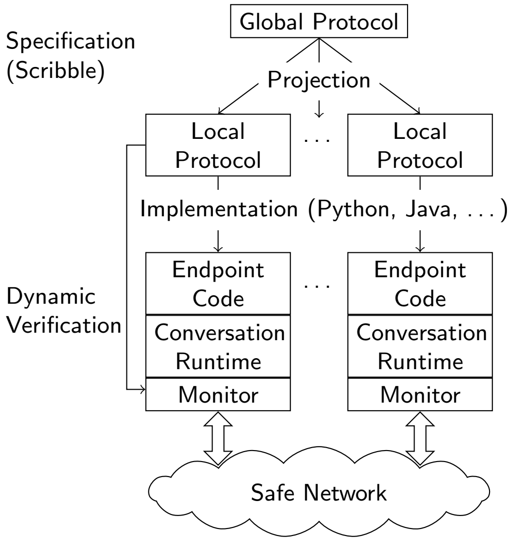
OOI agent negotiation

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

global protocol Negotiate(role Producer as P, role Consumer as C) {
  propose(SAP) from P to C;
  rec START {
    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;
                   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;
                     continue START;
} } } }
```



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

The Scribble Framework



- Scribble global protocols
  - Well-formedness validation
- Scribble local protocols
  - FSM generation (for monitoring)
- (Heterogeneous) endpoint programs
  - Scribble Conversation API
  - (Interoperable) Distributed Conversation Runtime

Global protocol well-formedness (Choice)

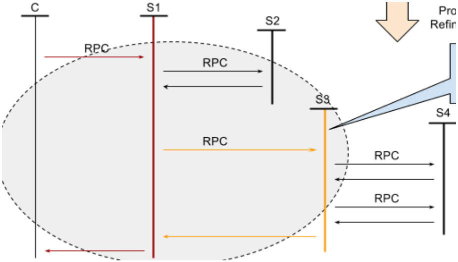
```
global protocol Choice2(role A, role B, role C) {
  choice at A {
    m1() from A to B;
    m2() from B to C;
  } or {
    m1() from A to B;
} }

global protocol Choice3(role A, role B, role C) {
  choice at A {
    m1() from A to B;
    m2() from B to C;
  } or {
    m1() from A to B;
    m3() from B to C;
} }
```

Global protocol well-formedness (Recursion)

```
global protocol Recursion1(role A, role B, role C, role D) {
  rec X {
    m1() from A to B;
    continue X;
  }
  m2() from A to B;
  m3() from C to D;
}
```

RPC composition 1/2

```
global protocol Foo1(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 2/2

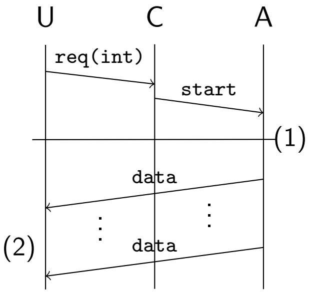
```
global protocol RPC<sig M1, sig M2>(role Client as C, role Server as S)
  M1 from C to S;
  M2 from S to C;
}

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 Foo3(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);
}
```

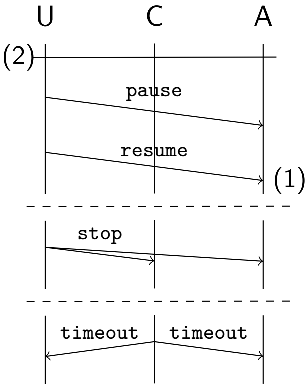
Agent negotiation 2

```
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 P, role CounterParty as 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;
    do NegotiateAux(C as Proposer, P as CounterParty);
} }
```

Resource Access 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)

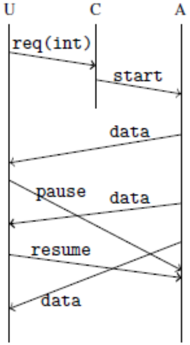




- https://confluence.oceanobservatories.org/display/CIDev/Resource+Control+in+Scribble

Extending MPST with interruptible conversations

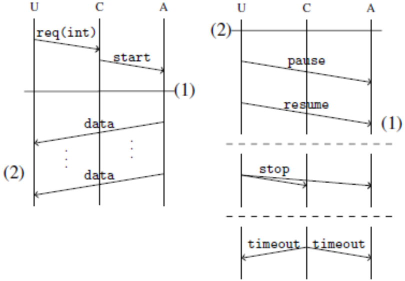
- Well-formed global types traditionally rule out any ambiguities between roles in the flow of the protocol: no messages lost or redundant
  - e.g. structure of non-mixed choice with role well-formedness
- Asynchronous interrupts: inherent "communication races"
  - Interruptible is a mixed choice, also completely optional
  - Concurrent and nested interrupts
  - Asynchronous entry/exit of interruptible blocks by roles



A valid trace

RAC Scribble

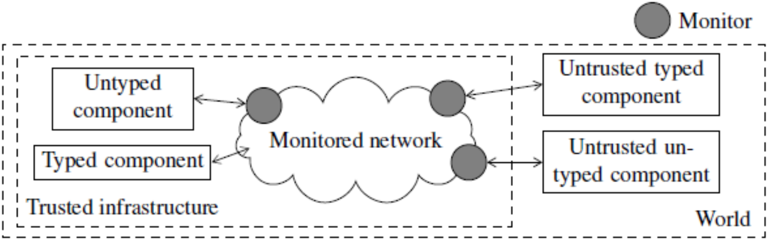
```
global protocol RC(
    role User as U, role Controller as C, role Agent as A) {
  req(int) from U to C;
  start() from C to A;
  interruptible {
    rec X {
      interruptible {
        rec Y {
          data() from A to U;
          continue Y;
      } }
      with {
        pause() by U;
   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



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:
  - Implemented by dynamic local type tracking of scopes

MPST-based distributed protocol monitoring



- 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.