Designing, Verifying and Monitoring Protocols

inspired by Scribble

Versions presented:
Luxembourg-Singapore Workshop on Security and Privacy — 2 March 2016
InfoLab21, Lancaster University, United Kingdom — 17 November 2015
Queen Mary University of London, United Kingdom — 11 November 2015
PSI: 10th Ershov Informatics Conference, Kazan, Russia — 25–27 August 2015

Ross Horne

Nanyang Technological University, Singapore

2 March 2016
Sessions in Distributed Systems

Client driven two phase commit (2PC) as a *sequence diagram.*
Sessions in Distributed Systems

Client driven two phase commit (2PC) as a *global session type* (based on Scribble\(^1\)).

\[
\begin{align*}
\{ \text{par} & \quad p\text{\_begin}(\text{Payload}) \text{ from Client to Participant} \\
& \quad \text{and} \\
& \quad l\text{\_begin}(\text{Payload}) \text{ from Client to Leader} \\
\} ; \\
\text{prepare}(\text{Timestamp}) \text{ from Participant to Leader} ; \\
\{ \text{par} & \quad c\text{\_commit}(\text{Timestamp}) \text{ from Leader to Client} \\
& \quad \text{and} \\
& \quad p\text{\_commit}(\text{Timestamp}) \text{ from Leader to Participant} \\
\}
\end{align*}
\]

Local session types for roles Client, Participant and Leader (based on Scribble).

Client: \{ \text{par} \sim p\text{-}begin(Payload) \text{ to Participant}
and \sim l\text{-}begin(Payload) \text{ to Leader}
\}
\text{c\text{-}commit(Timestamp) from Leader}

Leader: \text{l\text{-}begin(Payload) from Client ;}
\text{prepare(Timestamp) from Participant ;}
\{ \text{par} \sim p\text{-}commit(Timestamp) \text{ to Participant}
and \sim c\text{-}commit(Timestamp) \text{ to Client}
\}

Participant: \text{p\text{-}begin(Payload) from Client ;}
\sim \text{prepare(Timestamp) to Leader ;}
\text{p\text{-}commit(Timestamp) from Leader}
A Semantics for Multi-party Session Types

- How do we know that the projection is correct?

- How do we know when a protocol of one type can do everything that a protocol of another type can do?

- How can we determine when a collection of local types are compatible?

We need a **semantics**!
Sessions in Distributed Systems

Local session types for roles **Client**, **Participant** and **Leader**.

**Client:**
\[
\begin{align*}
\text{Client:} & \quad \{ \par \sim p.\text{begin}(\text{Payload}) \text{ to Participant} \\
& \quad \text{and } \sim l.\text{begin}(\text{Payload}) \text{ to Leader} \\
& \quad \} ; \\
& \quad c.\text{commit}(\text{Timestamp}) \text{ from Leader}
\end{align*}
\]

**Leader:**
\[
\begin{align*}
\text{Leader:} & \quad l.\text{begin}(\text{Payload}) \text{ from Client} ; \\
& \quad \text{prepare}(\text{Timestamp}) \text{ from Participant} ; \\
& \quad \{ \par \sim p.\text{commit}(\text{Timestamp}) \text{ to Participant} \\
& \quad \text{and } \sim c.\text{commit}(\text{Timestamp}) \text{ to Client} \\
& \quad \}
\end{align*}
\]

**Participant:**
\[
\begin{align*}
\text{Participant:} & \quad p.\text{begin}(\text{Payload}) \text{ from Client} ; \\
& \quad \sim \text{prepare}(\text{Timestamp}) \text{ to Leader} ; \\
& \quad p.\text{commit}(\text{Timestamp}) \text{ from Leader}
\end{align*}
\]
Multi-party Compatibility

par

{  par \sim p\_begin(Payload) to Participant
    and \sim l\_begin(Payload) to Leader
}

\textit{c\_commit(Timestamp)} from Leader

and

l\_begin(Payload) from Client;
\textit{prepare(Timestamp)} from Participant;

{  par \sim p\_commit(Timestamp) to Participant
    and \sim c\_commit(Timestamp) to Client
}

and

p\_begin(Payload) from Client;
\sim prepare(Timestamp) to Leader;
p\_commit(Timestamp) from Leader
Multi-party Compatibility

{  par ~ p\_begin(Payload) to Participant
    and p\_begin(Payload) from Client

    and ~ l\_begin(Payload) to Leader
    and l\_begin(Payload) from Client
}

{  par ~ prepare(Timestamp) to Leader
    and prepare(Timestamp) from Participant
}

{  par ~ p\_commit(Timestamp) to Participant
    and p\_commit(Timestamp) from Leader

    and ~ c\_commit(Timestamp) to Client
    and c\_commit(Timestamp) from Leader
}
Multi-party Compatibility
A Semantics for Session Types in the Calculus of Structures

atomic interaction
par \sim A \text{ and } B \rightarrow \{\} \quad \text{only if} \quad A \text{ is a subsort of } B

seq
\text{par} \{T;U\} \text{ and } \{V;W\} \rightarrow \{\text{par } T \text{ and } V\} ; \{\text{par } U \text{ and } W\}

switch
\text{par} \{\text{sync } T \text{ and } U\} \text{ and } V \rightarrow \text{sync } T \text{ and } \{\text{par } U \text{ and } V\}

left choice \quad \text{right choice} \quad \text{tidy}
T \text{ or } U \rightarrow T \quad T \text{ or } U \rightarrow U \quad \{\} \& \{\} \rightarrow \{\}

external choice
\text{par } T \text{ and } \{U \& V\} \rightarrow \{\text{par } T \text{ and } U\} \& \{\text{par } T \text{ and } V\}

medial
\{T;U\} \& \{V;W\} \rightarrow \{T \& V\} ; \{U \& W\}

context closure
\mathcal{C}\{T\} \rightarrow \mathcal{C}\{U\} \quad \text{only if} \quad T \rightarrow U

congruence
T \rightarrow U \quad \text{only if} \quad T \equiv U

\(T;\,,\{\}\) is a monoid and \((T,\text{par},\{\})\) and \((T,\text{sync},\{\})\) are commutative monoids.
Proof and Multi-party Compatibility

Definition (Proof)
A sequence of rewrites that ends with the unit ($\{ \}$) is a proof. ²

Definition (Multi-party compatibility)
If the parallel composition of all roles (and channels) is provable then the local protocols are multi-party compatible. ³ ⁴

Proposition
The multiset of projections from any global protocol to it’s local protocols for roles (and channels) is multi-party compatible.

Subtyping

Which protocol is a subtype of the other protocol?

I.e., can one protocol do everything that another protocol can do in every context?
Check for Subtyping

Definition
A local type $T$ is a subtype of local type $U$, written $T \leq U$, if and only if $\text{par} \sim T$ and $U$ is provable.

Firstly apply De Morgan properties to find the complement of Leader.

Leader: $l\_\text{begin}(\text{Payload})$ from Client;
prepare($\text{Timestamp}$) from Participant;
{
    \text{par} \sim p\_\text{commit}(\text{Timestamp})$ to Participant
    and $\sim c\_\text{commit}(\text{Timestamp})$ to Client
}

$\sim$Leader: $\sim l\_\text{begin}(\text{Payload})$ from Client;
$\sim$prepare($\text{Timestamp}$) from Participant;
{
    \text{sync} p\_\text{commit}(\text{Timestamp})$ to Participant
    and $c\_\text{commit}(\text{Timestamp})$ to Client
}
Check for Subtyping

Definition
A local type $T$ is a subtype of local type $U$, written $T \leq U$, if and only if par $\sim T$ and $U$ is provable.

par $\sim l\_begin(Payload) \text{ from Client} ;$
$\sim prepare(Timestamp) \text{ from Participant} ;$
{
    sync p\_commit(Timestamp) \text{ to Participant and c\_commit(Timestamp) to Client}
}

and
{
    par prepare(Timestamp) \text{ from Participant and l\_begin(Payload) from Client}
} ;
{
    par $\sim p\_commit(Timestamp) \text{ to Participant and } \sim c\_commit(Timestamp) \text{ to Client}
}

The above is provable, hence Leader $\leq$ Leader'. Hence Leader' can do everything Leader can do in any context.

For global protocols apply subtyping point-wise, hence 2PC $\leq$ 2PC'.
Example 2PC with the option for the participant to abort.

Client**:**

\[
\{ \text{par} \sim p\text{-begin(Payload)} \text{ to Participant and } \sim l\text{-begin(Payload)} \text{ to Leader} \}
\]

\[
\{ \text{commit(Timestamp) from Leader or } c\text{-abort(Error) from Leader} \}
\]

Participant**:**

\[
\{ p\text{-begin(Payload) from Client ; } \}
\]

\[
\{ \sim p\text{-commit(Timestamp) from Leader and } \sim c\text{-commit(Timestamp) from Client } \}
\]

\[
\{ p\text{-abort(Error) to Leader } \}
\]

Leader**:**

\[
\{ l\text{-begin(Payload) from Client; } \}
\]

\[
\{ \{ \text{prepare(Timestamp) from Participant ; } \}
\]

\[
\{ \{ \text{par} \sim p\text{-commit(Timestamp) to Participant and } \sim c\text{-commit(Timestamp) to Client } \}
\]

\[
\} \text{ or } \{ p\text{-abort(Error) from Participant ; } \sim c\text{-abort(Error) to Client } \}
\]

Due to internal choice **Leader ≤ Leader**'' and **Client ≤ Client**''.

However, due to external choice **Participant'' ≤ Participant**.
Coherence

Definition (Coherence)
A multiset of local types \((T_i)_{i \in I}\), where \(I\) is a set of roles and channels, is coherent (with respect to \(G\)) if there exists a global type \(G\) such that for all \(i \in I\), \(G \mid_i \leq T_i\).

\(\text{Leader''}^{''}, \text{Participant''}^{''}\) and \(\text{Client''}^{''}\) (plus channels) are coherent with respect to \(\text{2PC''}\):

\begin{verbatim}
par p_begin(Payload) from Client to Participant
and l_begin(Payload) from Client to Leader ;

choice at Participant {
    prepare(Timestamp) from Participant to Leader ;
    par c_commit(Timestamp) from Leader to Client
    and p_commit(Timestamp) from Leader to Participant
}

} or {
    p_abort(Error) from Participant to Leader ;
    c_abort(Error) from Leader to Client
}
\end{verbatim}
Coherence
Interoperability: the Sync Operator

- The Digital Ocean API can create instances in separate zones using one message.
- The Google Compute Engine API requires a separate message for each zone.

The protocol below is part of a mediator between the APIs of the two Cloud providers.

```
sync post(JSON) from Client
and {  par post1(JSON) to Server
      and post2(JSON) to Server }

{  {  sync alert(Error) from Server
      and anything
      and alert(Error) to Client }
    or
      {  sync response1(JSON) from Server
         and response2(JSON) from Server
         and response(JSON) to Client }
  }
```

The sync operator is used to synchronise inputs from the servers.
Interoperability: the Sync Operator

How do I know the mediator protocol is correct?

Digital Ocean Client: \[ \sim post(JSON) \text{ to Server;}
\{\]
\[\quad \text{response(JSON) from Mediator}
\text{ or}
\quad \text{alert(Error) from Mediator}
\}\]

Mediator: \[\text{sync post(JSON) from Client}
\text{ and}\{\]
\[\quad \text{par \sim post1(JSON) to Server}
\text{ and } \sim post2(JSON) \text{ to Server}\} ;
\{\]
\[\quad \text{sync alert(Error) from Server}
\text{ and anything}
\text{ and } \sim \text{alert(Error) to Client}\} \]
\[\text{or}\]
\[\quad \text{sync response1(JSON) from Server}
\text{ and response2(JSON) from Server}
\text{ and } \sim \text{response(JSON) to Client}\}
\}

2 × Google Compute Server: \[ post(JSON) \text{ from Mediator;}
\{\]
\[\quad \sim \text{response(JSON) to Mediator}
\text{ and}
\quad \sim \text{alert(Error) to Mediator}
\} \]
Subsorting

The subtyping relation agrees with standard subtyping for I/O types. Assume the following subsort relation holds:

\[ \text{nat} \leq \text{int} \]

The following hold:

- \( \sim c(\text{int}) \) to \( P \leq \sim c(\text{nat}) \) to \( P \) (contravariance).
  
  We can send something more specific (\text{nat}) when something more general (\text{int}) is expected.

- \( c(\text{nat}) \) from \( P \leq c(\text{int}) \) from \( P \) (covariance)
  
  We can be ready to receive something more general (\text{int}), when something more specific (\text{nat}) arrives.

Any preorder, e.g. subtyping for XML Schema, can be used for subsorting.
Properties of Subtyping: Cut Elimination

Theorem (Cut Elimination)
If $C\{\text{sync } T \text{ and } \sim T\}$ is provable, then $C\{\{\}\}$ is provable.

[snip: 70 pages of proof] 5

Corollary (Transitivity)
Subtyping is transitive, i.e. if $T \leq U$ and $U \leq V$, then $T \leq V$.

Corollary
Any coherent multiset of local types, is multiparty compatible.

Theorem (Feasibility)
Deciding the provability of a local type is a PSPACE-complete problem.

Applications to Security and Future Collaboration

» Monitoring: Runtime monitors generated from local session types can be used to detect when a participant violates permitted protocols. Scenarios include:

» distributed systems spanning organisation boundaries, such as a distributed database with replicas in multiple Cloud providers.

» virtualization, where virtual machines are leased for a particular purpose only.

» microvirtualization, where untrusted software is executed safely in an isolated process.

» Type checking: Security protocols themselves can be specified using session types. For example, an implementation of a client in an OAuth protocol can be checked against the local type for clients to ensure conformance.

» Verification: Dependent typed extensions are sufficiently powerful to be used to prove the correctness of security protocols themselves. Attacks can be discovered and the absence of certain attacks can be certified.
Example of Session Types for OAuth: Globally

**OAuth protocol as a sequence diagram.**
Example of Session Types for OAuth: Locally

**App:**
\[
\sim \text{initiate}(app\_ID, \text{scope}) \text{ to Server} ;
\]
\[
\{\} \text{ or } \{
\text{authorisation\_code}(\text{code}) \text{ from Server} ;
\sim \text{exchange}(app\_ID, \text{secret}, \text{code}) \text{ to Server} ;
\{\} \text{ or } \{
\text{access\_token}(\text{token}) \text{ from Server} ;
\sim \text{request}(\text{token}) \text{ to Resource} ;
\text{response}(\text{data}) \text{ from Resource}
\}
\}
\]

**Server:**
\[
\text{initiate}(app\_ID, \text{scope}) \text{ from App} ;
\sim \text{login\_page}(app\_ID, \text{scope}) \text{ to Owner} ;
\{\} \text{ or } \{
\text{authenticate}(\text{name}, \text{password}) \text{ from Owner} ;
\{\} \& \{
\sim \text{authorisation\_code}(\text{code}) \text{ to App} ;
\text{exchange}(app\_ID, \text{secret}, \text{code}) \text{ from App} ;
\{\} \& \{
\sim \text{access\_token}(\text{token}) \text{ to App}
\}
\}
\}
\]

**Resource:**
\[
\{\} \text{ or } \{
\text{request}(\text{token}) \text{ from App} ;
\sim \text{response}(\text{data}) \text{ to App}
\}
\]

**Owner:**
\[
\text{login\_page}(app\_ID, \text{scope}) \text{ from Server} ;
\{\} \& \{
\sim \text{authenticate}(\text{name}, \text{password}) \text{ to Server}
\}
\]
Conclusion

A proof theoretic foundation for session types:

- The first *session type system* expressed in the *calculus of structures* enabling:
  
  - a natural notion of **multi-party compatibility** (using provability);
  
  - A **consistent** notion of **subtyping** (using linear implication);

- Projection from *global types* guarantees multi-party compatibility.

Applications to security include:

- Runtime monitoring to detect violations of specified protocols.

- Type checking code for conformance to a role in a security protocol.

- Verification of security protocols themselves in dependently typed extensions.

Future extensions include fixed points or replication to enable the analysis of protocols with unbounded participants and the behaviour of attackers with the ability to initiate unbounded sessions.
Extra Example: Tiu’s Counterexample

Role $P$: \( \sim\text{begin}(\text{Data}) \) to $Q$ ;

\[
\{ \\
\quad \text{par } \sim\text{fun}(\text{Control}) \text{ to } Q \\
\quad \text{and } \text{done}(\text{Data}) \text{ from } Q \\
\} 
\]

Role $Q$: \{ \\
\quad \text{par } \text{begin}(\text{Data}) \text{ from } P \\
\quad \text{and } \text{fun}(\text{Control}) \text{ from } P \\
\} ; \\
\quad \sim\text{done}(\text{Data}) \text{ to } P

Coordinating middleware:

\(\text{sync } \text{begin}(\text{Data}) \text{ to } Q \text{ and } \sim\text{begin}(\text{Data}) \text{ from } P\)

\(\text{sync } \text{fun}(\text{Control}) \text{ to } Q \text{ and } \sim\text{fun}(\text{Control}) \text{ from } P\)

\(\text{sync } \text{done}(\text{Data}) \text{ to } P \text{ and } \sim\text{done}(\text{Data}) \text{ from } Q\)
Extra Example: Tiu’s Counterexample

par { 
  \sim\begin{Data} \to Q \; \\ 
  \{ 
    \par \sim\text{fun}(\Control) \to Q \\
    \text{and } \text{done}(\Data) \from Q
  \}
\}
} 
and { 
  \{ 
    \par \begin{Data} \from P \\
    \text{and } \text{fun}(\Control) \from P
  \}
\}; 
\sim\text{done}(\Data) \to P 
} 
and { 
  \text{sync } \begin{Data} \to Q \text{ and } \sim\begin{Data} \from P 
\} 
} 
and { 
  \text{sync } \text{fun}(\Control) \to Q \text{ and } \sim\text{fun}(\Control) \from P 
\} 
and { 
  \text{sync } \text{done}(\Data) \to P \text{ and } \sim\text{done}(\Data) \from Q 
\}
Extra Example: Tiu’s Counterexample (deep step)

\[
\begin{align*}
\text{par} \{ \\
&\sim \text{begin}(\text{Data}) \text{ to } Q \\
&\sim \text{fun}(\text{Control}) \text{ to } Q \\
&\text{done}(\text{Data}) \text{ from } Q \\
\} \\
\text{and} \{ \\
&\text{begin}(\text{Data}) \text{ from } P \\
&\text{fun}(\text{Control}) \text{ from } P \\
&\text{codone}(\text{Data}) \text{ to } P \\
\} \\
\text{and} \{ \\
&\text{sync } \text{begin}(\text{Data}) \text{ to } Q \text{ and } \sim \text{begin}(\text{Data}) \text{ from } P \\
&\text{sync } \text{fun}(\text{Control}) \text{ to } Q \text{ and } \sim \text{fun}(\text{Control}) \text{ from } P \\
&\text{sync } \text{done}(\text{Data}) \text{ to } P \text{ and } \sim \text{done}(\text{Data}) \text{ from } Q
\}
\end{align*}
\]
Extra Example: Tiu’s Counterexample

{ 
  par 
  \sim begin(Data) \text{ to } Q 
  \text{ and } 
  begin(Data) \text{ from } P 
  \text{ and } 
  \text{ sync } begin(Data) \text{ to } Q \text{ and } \sim begin(Data) \text{ from } P 
} ; 
par 
\sim fun(Control) \text{ to } Q 
\text{ and } 
fun(Control) \text{ from } P 
\text{ and } 
\text{ sync } fun(Control) \text{ to } Q \text{ and } \sim fun(Control) \text{ from } P 
} ; 
par 
\text{ done(Data) from } Q 
\text{ and } 
\text{ codone(Data) to } P 
\text{ and } 
\text{ sync } \text{ done(Data) to } P \text{ and } \sim \text{ done(Data) from } Q 
}
Extra Example: Tiu's Counterexample

{
    sync {
        par \sim begin(Data) to Q and begin(Data) to Q
    }
    and {
        par \sim begin(Data) from P and begin(Data) from P
    }
}
{
    sync {
        par \sim fun(Control) to Q and fun(Control) to Q
    }
    and {
        par \sim fun(Control) from P and fun(Control) from P
    }
}
{
    sync {
        par \sim done(Data) to P and done(Data) to P
    }
    and {
        par \sim done(Data) from Q and done(Data) from Q
    }
}
Extra Example: Tiu’s Counterexample

Tiu’s counterexample is coherent with respect to:

\begin{align*}
\text{begin}(Data) & \text{ from } P \text{ to } Q; \\
\text{fun}(Function) & \text{ from } P \text{ to } Q; \\
\text{done}(Data) & \text{ from } Q \text{ to } P
\end{align*}