Type-safe interactive web service generation from Scribble

Jonathan King, Nicholas Ng, Nobuko Yoshida

18 Dec 2018 — ABCD project meeting
The project

- **Multiparty** Session Types for interactive web applications
- Scribble applied to PureScript & WebSocket
  - PureScript: strongly-typed functional language, compiles to JavaScript
  - WebSocket: full-duplex communication from the browser
- Embedding of local types/Endpoint FSMs as type classes and constraints
- Jonathan King’s final year Master’s project
- 8-months of term time work (concurrent with lectures)
Multiparty Session Types (MPST)

Global type

WF check & Endpoint projection

Endpoint types

Conform

Processes

Specify
- Global msg-passing protocol

Implement
- Endpoint processes

Guarantees
- Communication safety
- Deadlock freedom
- Protocol fidelity

Verification techniques for conformance
- Direct static type checking against implementation
- Runtime monitoring/checks
- APIs/code generation from session types

Jonathan King, Nicholas Ng, Nobuko Yoshida
Type-safe interactive web service generation from Scribble

mrg.doc.ic.ac.uk
Multiparty Session Types (MPST)

Specify
- Global msg-passing protocol

Implement
- Endpoint processes

Guarantees
- Communication safety
- Deadlock freedom
- Protocol fidelity

Verification techniques for conformance
- Direct static type checking against implementation
- Runtime monitoring/checks
- APIs/code generation from session types

Jonathan King, Nicholas Ng, Nobuko Yoshida
Type-safe interactive web service generation from Scribble
Implementations and applications of MPST using Scribble

Global protocol

WF check & Endpoint projection

Endpoint protocols

Endpoint FSMs

APIs for endpoint programming

Global Scribble protocol

global protocol P(role S, C)
{ Msg(int) from S to C;
Reply(bool) from C to S; }

Project

Endpoint Scribble protocol S

local protocol P_S(role C)
{ Msg(int) to C;
Reply(bool) from C; }

Translate

Endpoint FSM for S

Generate

Endpoint APIs (for users), e.g. Java

class S1 { S2 Send(int x) {} }
class S2 { EndRecv(boolean b) {} }

Language & runtime support

G

E1  Ei  EN

EFSM1  EFSMi  EFSMN

APIs for endpoint programming
Implementations and applications of MPST using Scribble

Some example uses:

<table>
<thead>
<tr>
<th>Language</th>
<th>Transports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid session verification (FASE’16)</td>
<td>TCP, SSL/TCP, HTTP</td>
</tr>
<tr>
<td>Explicit connection actions (FASE’17)</td>
<td>Java</td>
</tr>
<tr>
<td>Typestate generation (SCP, 2017)</td>
<td>Java methods</td>
</tr>
<tr>
<td>Linear decomposition (ECOOP’17)</td>
<td>Scala</td>
</tr>
<tr>
<td>TCP, shared mem., Akka actors</td>
<td>TCP</td>
</tr>
<tr>
<td>Session Type Provider (CC’18)¹</td>
<td>F#</td>
</tr>
<tr>
<td>Role-parametric MPST (POPL’19)²</td>
<td>Go</td>
</tr>
<tr>
<td>TCP, shared mem.</td>
<td></td>
</tr>
</tbody>
</table>

¹ Talk before this
² Talk this afternoon
**Implementations and applications of MPST using Scribble**

Some example uses:

<table>
<thead>
<tr>
<th>Language</th>
<th>Transports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid session verification (FASE’16)</td>
<td>Java TCP, SSL/TCP, HTTP</td>
</tr>
<tr>
<td>Explicit connection actions (FASE’17)</td>
<td>Java TCP, SSL/TCP, HTTP</td>
</tr>
<tr>
<td>Typestate generation (SCP, 2017)</td>
<td>Java Java methods</td>
</tr>
<tr>
<td>Linear decomposition (ECOOP’17)</td>
<td>Scala TCP, shared mem., Akka actors</td>
</tr>
<tr>
<td>Session Type Provider (CC’18)</td>
<td>F# TCP</td>
</tr>
<tr>
<td>Role-parametric MPST (POPL’19)</td>
<td>Go TCP, shared mem.</td>
</tr>
</tbody>
</table>

- All target desktop/distributed applications
- Can we apply it to web applications?

---

1 Talk before this
2 Talk this afternoon
Web-based interface to the Scribble tool

- **Check** protocols
- **Project** protocol to endpoint protocols
- **Generate FSM** from Scribble protocols

*Communicates* with server to execute Scribble

```java
global protocol Playground(role C, role S) {
    choice at C {
        Verify(String) from C to S; // WF checks
        Result(QResult) from S to C;
    } or {
        Project(String, String, String) from C to S;
        Result(QResult) from S to C;
    } or {
        FSM(String, String, String) from C to S;
        Result(QResult) from S to C;
    }
    do Playground(C , S);
}
```

(Similar web-based *playground* exist for Go, Rust, etc.)
Scribble-based API generation for the web

<table>
<thead>
<tr>
<th>Language</th>
<th>Transports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid session verification (FASE’16)</td>
<td>Java TCP, SSL/TCP, HTTP</td>
</tr>
<tr>
<td>Explicit connection actions (FASE’17)</td>
<td>Java Java methods</td>
</tr>
<tr>
<td>Typestate generation (SCP, 2017)</td>
<td>Java TCP, shared mem., Akka actors</td>
</tr>
<tr>
<td>Linear decomposition (ECOOP’17)</td>
<td>Scala TCP, shared mem., Akka actors</td>
</tr>
<tr>
<td>Session Type Provider (CC’18)</td>
<td>F# TCP</td>
</tr>
<tr>
<td>Role-parametric MPST (POPL’19)</td>
<td>Go TCP, shared mem.</td>
</tr>
<tr>
<td><strong>This work</strong></td>
<td><strong>JavaScript WebSocket</strong></td>
</tr>
</tbody>
</table>

Challenge: JavaScript not statically typed
PureScript
A strongly-typed functional programming language that compiles to JavaScript

From PureScript homepage:
- Compile to readable JavaScript and reuse existing JavaScript code easily
- An extensive collection of libraries for development of web applications, web servers, apps and more
- Excellent tooling and editor support with instant rebuilds
- An active community with many learning resources
- Build real-world applications using functional techniques and expressive types, such as:
  - Algebraic data types and pattern matching
  - Row polymorphism and extensible records
  - Higher kinded types
  - Type classes with functional dependencies
  - Higher-rank polymorphism

Jonathan King, Nicholas Ng, Nobuko Yoshida
Type-safe interactive web service generation from Scribble
PureScript
A strongly-typed functional programming language that compiles to JavaScript

From PureScript homepage:

- Compile to readable JavaScript and reuse existing JavaScript code easily
- An extensive collection of libraries for development of web applications, web servers, apps and more
- Excellent tooling and editor support with instant rebuilds
- An active community with many learning resources
- Build real-world applications using functional techniques and expressive types, such as:
  - Algebraic data types and pattern matching
  - Row polymorphism and extensible records
  - Higher kinded types
  - Type classes with functional dependencies
  - Higher-rank polymorphism

Jonathan King, Nicholas Ng, Nobuko Yoshida
Type-safe interactive web service generation from Scribble
mrg.doc.ic.ac.uk
PureScript code generation

PureScript types are generated from the EFSMs, informally:

- Each `state` is a type
- Each `transition` is a type class instance

The (multi-parameter) type classes for each kind of `transition`:

- Send: message send
-Recv: message receive
- Select: selection (receive label)
- Branch: branching (send label)
(Simplified) **Send** and **Receive** type classes, parametrised by $s$, $t$, $a$

```java
class Send $s^! t a \mid s^! \rightsquigarrow t a$
class Recv $s^? t a \mid s^? \rightsquigarrow t a$
```

$s^!/s^?$: Sending/Receiving state

- $a$: Payload type
- $t$: State after transition

**Functional dependencies:**

$(t, a)$ uniquely identified by $s^!/s^?$

---

Jonathan King, Nicholas Ng, Nobuko Yoshida

*Type-safe interactive web service generation from Scribble*
EFSM transitions as type classes (2)

(Simplified) **Selection** and **Branching** type classes, parameterised by $ts$

```haskell
class Select $s^! \, ts \mid s^! \rightsquigarrow ts$
```

```haskell
class Branch $s^? \, ts \mid s^? \rightsquigarrow ts$
```

$s^! / s^?$: Selecting/Branching state

$ts$: Row list of tuples $(l_i, t_i)_{i \in |ts|}$

$l_i, t_i$: Branching label $i$, continuation state $i$

**Functional dependencies:**

$ts$ uniquely identified by $s^! / s^?$

Branch (output $t_i$ only)

```
\[
\begin{array}{c}
\begin{array}{c}
\text{Select} (\text{input } t_i \text{ only})
\end{array}
\end{array}
\]
```

```
\[
\begin{array}{c}
\begin{array}{c}
\text{Branch (output } t_i \text{ only)}
\end{array}
\end{array}
\]
```
EFSM as typing constraints

- Constrained by types and functional dependencies
- Effectively embedding EFSMs as types: type checks = conformance
- State transitions linearity ensured by Indexed Monad

```
foreign import data S9 :: Type
foreign import data S9Verify :: Type
foreign import data S9Project :: Type
foreign import data S9FSM :: Type
foreign import data S11 :: Type
foreign import data S12 :: Type
foreign import data S13 :: Type

instance initialClient :: Initial Client S9
instance terminalClient :: Terminal Client Void
instance sendS9Verify :: Send Server S9Verify S11 Verify
instance sendS9Project :: Send Server S9Project S12 Project
instance sendS9FSM :: Send Server S9FSM S13 FSM
instance selectS9 :: Select Server S9 (Cons "project" S9Project (Cons "fsm" S9FSM (Cons "verify" S9Verify Nil)))
instance receiveS11 :: Receive Server S11 S9 Result
instance receiveS12 :: Receive Server S12 S9 Result
instance receiveS13 :: Receive Server S13 S9 Result
```
Scribble playground
Demo

Scribble

Omitted in talk:

- Runtime
- Web framework (Halogen library (?))
- Connection actions
- Error handling and reporting
Summary

**Specify** global protocol in Scribble

**Project** global protocol into local protocols

**Translate** local protocol to EFSM

**Generate** PureScript type constraints of protocol from EFSM

**Write** web application endpoint in PureScript

**Run** type-safe web application

(And we have a new Scribble playground!)