Design-by-Contract for \textit{Flexible} Multiparty Session Protocols

Lorenzo Gheri @ Imperial College
Ivan Lanese @ Focus Team, University of Bologna/INRIA
Neil Sayers @ Imperial College & Coveo Solutions Inc.
Emilio Tuosto @ GSSI
Nobuko Yoshida @ Imperial College

Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233
Take-home message

Choreography Automata

A model of choreographies of message-passing systems featuring selective participation

deadlock and lock freedom by construction

design-by-contract: constrain payloads of communications

CAScr (https://github.com/Tooni/CAScript-Artifact)

A tool chain for top-down choreographic development validating protocols via choreography automata

TypeScript web programming via API generation

Check out our paper or get in touch for details...
Choreography Automata

A model of choreographies of message-passing systems featuring

- selective participation
- deadlock and lock freedom by construction
- design-by-contract: constrain payloads of communications
Take-home message

Choreography Automata
A model of choreographies of message-passing systems featuring
- selective participation
- deadlock and lock freedom by construction
- design-by-contract: constrain payloads of communications

CAScr (https://github.com/Tooni/CAScript-Artifact)
A tool chain for
- top-down choreographic development
- validating protocols via choreography automata
- TypeScript web programming via API generation
### Take-home message

**Choreography Automata**

A model of choreographies of message-passing systems featuring
- selective participation
- deadlock and lock freedom by construction
- design-by-contract: constrain payloads of communications

**CAScr** ([https://github.com/Tooni/CAScript-Artifact](https://github.com/Tooni/CAScript-Artifact))

A tool chain for
- top-down choreographic development
- validating protocols via choreography automata
- TypeScript web programming via API generation

Check out our paper or get in touch for details...
– Prologue –

[ Choreographies, informally ]
The online-wallet protocol

- Customer → Wallet: login
- Customer → Wallet: pin
- Wallet → Customer: retry
- Wallet → Customer: loginOK
- Wallet → Vendor: loginOK
- Vendor → Customer: request
- Customer → Wallet: authorise
- Customer → Wallet: reject
- Customer → Vendor: pay
- Customer → Vendor: reject
The online-wallet protocol ...some modelling problems

What about vendor?
The online-wallet protocol

...some modelling problems

What about vendor?

What about payloads?

customer → wallet: login

customer → wallet: pin

wallet → customer: retry

wallet → customer: loginOK

wallet → vendor: loginOK

vendor → customer: request

customer → wallet: authorise

customer → wallet: reject

customer → vendor: pay

customer → vendor: reject

wallet → customer: loginDenied
Quoting W3C:

“[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...] Each party can then use the global definition to build and test solutions [...] global specification is in turn realised by combination of the resulting local systems”
Quoting W3C:

"[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...] Each party can then use the global definition to build and test solutions [...] global specification is in turn realised by combination of the resulting local systems"
Quoting W3C:

“[…] a contract […] of the common ordering conditions and constraints under which messages are exchanged […] from a global viewpoint […] Each party can then use the global definition to build and test solutions […] global specification is in turn realised by combination of the resulting local systems”
Top-down model-driven development

Quoting W3C:

“[...] a contract [...] of the common ordering conditions and constraints under which messages are exchanged [...] from a global viewpoint [...] Each party can then use the global definition to build and test solutions [...] global specification is in turn realised by combination of the resulting local systems”
– Act I –

[ Choreography Automata ]
Our global & local specs

Choreography automata: Interaction, globally

\[ M = q_0 \]

\[ Q_{proj}(M, vendor) = Q_5 \]

\[ v \rightarrow c: loginOK \]

\[ v \rightarrow c: request \]

\[ c \rightarrow v: reject \]

\[ c \rightarrow v: pay \]
Our global & local specs

Intermediate automata: from interactions to communications

\[ \hat{M} = q_0 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{\epsilon} q_3 \xrightarrow{\epsilon} q_4 \xrightarrow{v c! \text{request}} q_5 \xrightarrow{\epsilon} q_6 \xrightarrow{c v? \text{pay}} q_7 \]

Communicating finite-state machines: Communication, locally

\[ \text{proj}(M, \text{vendor}) = Q_4 \xrightarrow{w v? \text{loginOK}} Q_5 \xrightarrow{v c! \text{request}} Q_6 \xrightarrow{c v? \text{pay}} Q_3 \]
Semantics of CFSMs

Internal step: \( S \xrightarrow{\varepsilon} S' \)
Semantics of CFSMs

Internal step: $S \xrightarrow{\varepsilon} S'$

```
\[
\begin{array}{cccc}
p & \ldots & r & \ldots \\
\circlearrowleft & \circlearrowleft & \circlearrowright & \circlearrowright \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\end{array}
\]
Semantics of CFSMs

Internal step: \( S \xrightarrow{\varepsilon} S' \)

Interaction: \( S \xrightarrow{p \rightarrow_{q: m}} S' \)
Semantics of CFSMs

Internal step: $S \xrightarrow{\varepsilon} S'$

Interaction: $S \xrightarrow{p \rightarrow q : m} S'$
**Theorem.** Choreography automata are bisimilar to their projections

\[ \Rightarrow \text{ traces equivalence} \]
Selective participation in OLW

- $q_0$ to $q_0$: $c \rightarrow w$: login
- $q_0$ to $q_1$: $c \rightarrow w$: pin
- $q_1$ to $q_1$: $w \rightarrow c$: retry
- $q_1$ to $q_2$: $c \rightarrow w$: login
- $q_2$ to $q_2$: $w \rightarrow c$: loginOK
- $q_2$ to $q_3$: $w \rightarrow c$: loginDenied
- $q_3$ to $q_3$: $c \rightarrow w$: reject
- $q_3$ to $q_4$: $c \rightarrow v$: pay
- $q_4$ to $q_4$: $w \rightarrow c$: loginOK
- $q_4$ to $q_5$: $v \rightarrow c$: request
- $q_5$ to $q_5$: $c \rightarrow w$: reject
- $q_5$ to $q_6$: $c \rightarrow v$: pay
- $q_7$ to $q_7$: $c \rightarrow w$: authorise
- $q_6$ to $q_6$: $v \rightarrow c$: request
- $q_6$ to $q_7$: $c \rightarrow v$: pay
- $q_7$ to $q_7$: $w \rightarrow c$: loginOK

From the very beginning, the wallet and customer are aware of the process. The vendor is involved on one branch only, but that's fine: the wallet is aware of the process. Eventually, the vendor is informed by the customer on each branch.
Selective participation in OLW

- at $q_2$ wallet and customer aware from the very beginning
Flexibility by example

Selective participation in OLW

- at q2 wallet and customer aware from the very beginning
  - vendor involved on one branch only, but that’s fine: wallet is aware
Flexibility by example

Selective participation in OLW

- at $q_2$ wallet and customer aware from the very beginning
  - vendor involved on one branch only, but that’s fine: wallet is aware
- at $q_6$ wallet and customer aware from the very beginning
Flexibility by example

Selective participation in OLW

- at $q_2$ wallet and customer **aware** from the very beginning
  - vendor involved on one branch only, but that’s fine: wallet is **aware**
- at $q_6$ wallet and customer **aware** from the very beginning
  - vendor eventually informed by customer on each branch
Correctness by construction

**Theorem.** Projections of well-formed choreography automata are deadlock-free

**Theorem.** Projections of well-formed choreography automata are lock-free
– Act II –

[ Asserted Choreography Automata ]
DbC vs. choreography automata

Asserting (an excerpt of) OLW
DbC vs. choreography automata

Asserting (an excerpt of) OLW

Consistency

- history sensitivity: in $q \overset{\lambda_A}{\rightarrow} q'$, $A$ predicates on known variables
- temporal satisfiability: the conjunction of the predicates on a path is satisfiable
- well-formedness of the underlying choreography automaton
Theorems

Projections are a bit more complicated than for choreography automata

On consistent asserted choreography automata

**Theorem.** Asserted choreography automata are *weakly* bisimilar to their projections

\[ \Rightarrow \] trace equivalence

**Theorem.** Projections of well-formed asserted choreography automata are deadlock-free
– Act III –

[ CAScr ]
Architecture of CAScr

User input
- Scribble protocol
- Participant declaration: server and others

CFSMs
- CA
- WF checks

STScript

code generation

Generated APIs for TypeScript web development
- Node.js (server)
- React (non-server)
Architecture of CAScr

User input
- Scribble protocol
- Participant declaration: server and others

Mapping
- CA
- WF checks

Projection
- CFSMs

Code generation
- STScript

Generated APIs for TypeScript web development
- Node.js (server)
- React (non-server)

Global protocol

```typescript
global protocol OnlineWallet(role wallet, role customer, role vendor) {
    rec AuthLoop {
        login(account: int) from customer to wallet;
        pin(pin: int) from customer to wallet;
        choice at wallet {
            login_ok() from wallet to customer;
            ...
            or login_denied(msg: string) from wallet to customer;
            ...
            or login_retry(msg: string) from wallet to customer;
            continue AuthLoop;
        }
    }
}
```
Architecture of CAScr

- **User input**
  - Scribble protocol
  - Participant declaration: server and others

- **Mapping to**
  - Global protocol `OnlineWallet` for TypeScript
    ```
    global protocol OnlineWallet(\text{role wallet, role customer, role vendor}) {
        \text{rec AuthLoop} {
            \text{login(account: int) from customer to wallet;}
            \text{pin(pin: int) from customer to wallet;}
            \text{choice at wallet} {
                \text{login\_ok() from wallet to customer;}
                \ldots
                \text{or login\_denied(msg: string) from wallet to customer;}
                \ldots
                \text{or login\_retry(msg: string) from wallet to customer;}
            }
            \text{continue AuthLoop;}
        }
    }
    ```
  - CFSMs
    - Node.js (server)
    - React (non-server)

- **Generated APIs**
  - for web development
    - Node.js (server)
    - React (non-server)
Multiparty global types

Syntax

\[ G ::= \sum_{i \in I} p \rightarrow q_i : m_i; G_i | \mu r. G | r | \text{end} \]

Semantics

\[ \sum_{i \in I} p \rightarrow q_i : m_i; G_i \xrightarrow{p \rightarrow q_j : m_j} G_j \ (j \in I) \]

\[ G[\mu r. G / r] \xrightarrow{\alpha} G' \]

\[ \mu r. G \xrightarrow{\alpha} G' \]
From global types to choreography automata

\[ \sum_{i \in I} p \rightarrow q_i : m_i G_i \rightarrow q_1 : m_1 \]

\[ \vdots \]

\[ p \rightarrow q_n : m_n G_n \rightarrow q_n : m_n \]
From global types to choreography automata

\[ \sum_{i \in I} p \rightarrow q_i : m_i \rightarrow G_i \]

\[ \vdots \]

\[ p \rightarrow q_n : m_n \rightarrow G_n \]

\[ \mu \cdot G \rightarrow G \]
From global types to choreography automata

\[
\sum_{i \in I} p_i \rightarrow q_i : m_i G_i
\]

\[
p \rightarrow q_1 : m_1 G_1
\]

\[
p \rightarrow q_n : m_n G_n
\]

\[
\mu \cdot G
\]

\[
\varepsilon \quad \varepsilon
\]

computes the mapping above
checks well-formedness of the resulting choreography automaton

generates the TypeScript API of each participant
From global types to choreography automata

CAScr

- computes the mapping above
- checks well-formedness of the resulting choreography automaton
- generates the TypeScript API of each participant
– Epilogue –

[ ... ]
Summing up

**Choreography Automata (with assertions)**

A theory of choreographies
- with increased expressiveness
- supporting DbC
- providing a basis for (enhanced) tool support for TypeScript web programming

**Plans**
- Consider asynchronous communications
- Applications:
  - inferring a (local) models from APIs and
  - checking their conformance against projections of a global spec
Thank you!