Take-home message
**Take-home message**

<table>
<thead>
<tr>
<th>Choreography Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>A model of choreographies of message-passing systems featuring</td>
</tr>
<tr>
<td>- selective participation</td>
</tr>
<tr>
<td>- deadlock and lock freedom <em>by construction</em></td>
</tr>
<tr>
<td>- design-by-contract: constrain payloads of communications</td>
</tr>
</tbody>
</table>

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...
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)
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

wallet → customer: loginDenied

customer → vendor: pay

customer → vendor: reject

customer → wallet: login

customer → wallet: pin
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
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”
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”

well-formedness

specs, not code
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”

![OLW diagram]
- Customer Local viewpoint
- Wallet Local viewpoint
- Vendor Local viewpoint

well-formedness
specs, not code
Choreography = Global spec + Local spec

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 ]
Choreography automata: Interaction, globally

\[ M = q_0 \xrightarrow{\text{w : login}} q_1 \xrightarrow{\text{c : retry}} q_2 \xrightarrow{\text{w : loginOK}} q_4 \xrightarrow{\text{w : loginOK}} q_5 \xrightarrow{\text{v : loginOK}} q_6 \]

\[ q_0 \xrightarrow{\text{c : login}} q_1 \xrightarrow{\text{c : pin}} q_2 \xrightarrow{\text{c : loginOK}} q_4 \]

\[ q_1 \xrightarrow{\text{c : login}} q_0 \xrightarrow{\text{w : retry}} q_2 \xrightarrow{\text{w : loginOK}} q_4 \]

\[ q_3 \xrightarrow{\text{c : loginDenied}} q_5 \xrightarrow{\text{v : loginOK}} q_6 \xrightarrow{\text{c : request}} q_8 \xrightarrow{\text{v : request}} q_7 \]

\[ q_5 \xrightarrow{\text{v : loginOK}} q_6 \xrightarrow{\text{c : request}} q_8 \xrightarrow{\text{c : reject}} q_7 \]

\[ q_6 \xrightarrow{\text{c : reject}} q_8 \xrightarrow{\text{v : reject}} q_7 \]

\[ q_7 \xrightarrow{\text{v : pay}} q_8 \xrightarrow{\text{c : pay}} q_6 \]

\[ q_8 \xrightarrow{\text{c : pay}} q_6 \xrightarrow{\text{v : pay}} q_7 \]
Our global & local specs

Intermediate automata: from interactions to communications

\[
\hat{M} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)
\]

Communicating finite-state machines: Communication, locally

\[
\text{proj}(M, \text{vendor}) = (Q, \Sigma, \Gamma, \delta, q_4, Q_F)
\]
Semantics of CFSMs

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

- $p \rightarrow \cdots \rightarrow r \rightarrow \cdots \rightarrow q$
Semantics of CFSMs

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

Diagram:

- p

- r

- q
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

\[ \implies \text{traces equivalence} \]
Flexibility by example

Selective participation in OLW

\[ M = \]

- \[ \text{c} \rightarrow \text{w}: \text{login} \]
- \[ \text{q}_1 \quad \text{q}_0 \]
- \[ \text{c} \rightarrow \text{w}: \text{pin} \]
- \[ \text{q}_2 \quad \text{q}_1 \]
- \[ \text{w} \rightarrow \text{c}: \text{retry} \]
- \[ \text{q}_0 \quad \text{q}_2 \]
- \[ \text{w} \rightarrow \text{c}: \text{loginOK} \]
- \[ \text{q}_2 \quad \text{q}_4 \]
- \[ \text{c} \rightarrow \text{w}: \text{loginOK} \]
- \[ \text{q}_1 \quad \text{q}_3 \]
- \[ \text{w} \rightarrow \text{c}: \text{loginDenied} \]
- \[ \text{q}_3 \quad \text{q}_2 \]
- \[ \text{c} \rightarrow \text{w}: \text{reject} \]
- \[ \text{q}_4 \quad \text{q}_5 \]
- \[ \text{w} \rightarrow \text{v}: \text{loginOK} \]
- \[ \text{v} \rightarrow \text{c}: \text{request} \]
- \[ \text{q}_6 \quad \text{q}_5 \]
- \[ \text{c} \rightarrow \text{w}: \text{authorise} \]
- \[ \text{c} \rightarrow \text{w}: \text{reject} \]
- \[ \text{q}_7 \quad \text{q}_6 \]
- \[ \text{c} \rightarrow \text{v}: \text{pay} \]
Selective participation in OLW

- at \( q_2 \) 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
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
Theorems

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

\[ q_0 \xrightarrow{r \cdot \text{try}} 0 \]
\[ 0 \leq \text{try} \leq 3 \]

\[ q_1 \xrightarrow{c \rightarrow w : \text{login} \langle \text{account int} \rangle} \]
\[ q_2 \]

\[ q_2 \xrightarrow{r \cdot \text{try}} \text{try} + 1 \]
\[ 0 \leq \text{try} \leq 3 \]

\[ q_3 \]

\[ q_3 \xrightarrow{w \rightarrow c : \text{loginDenied} \langle \text{msg string} \rangle} q_4 \]
\[ \text{try} \geq 3 \land \text{msg} = "5 \text{ min.}" \]

\[ q_4 \xrightarrow{0 \leq \text{try} < 3 \land \text{msg} = "fail"} \]

\[ q_4 \xrightarrow{0 \leq \text{try} \leq 3} \]

Consistency:
- History: \( q_0 \xrightarrow{\lambda} A q_1 \) if \( A \) predicates on known variables
- Temporal satisfiability: the conjunction of the predicates on a path is satisfiable
- Well-formedness of the underlying choreography automaton
DbC vs. choreography automata

### Asserting (an excerpt of) OLW

![Diagram of an automaton with labeled transitions and conditions for each state.]

- $q_0$: $r \cdot \text{try} \mapsto 0$
- $0 \leq \text{try} \leq 3$
- $q_1$: $r \cdot \text{try} \mapsto \text{try} + 1$
- $0 \leq \text{try} \leq 3$
- $q_2$: $0 \leq \text{try} < 3$
- $\land \text{msg} = "fail"
- $q_3$: $\text{try} \geq 3 \land \text{msg} = "5\ min."

### Consistency

- **history senesitiveness**: $\lambda \frac{A}{\rightarrow} q'$, $A$ 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

Mapping
- CA
  - WF checks

Projection
- CFSMs

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

STScript

Code generation
Architecture of CAScr

global protocol OnlineWallet(\textit{role} wallet, \textit{role} customer, \textit{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

STScript
- Code generation

CA
- WF checks

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

Projection

```
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;
  }
}
```
Multiparty global types

Syntax

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

Semantics

\[
\sum_{i \in I} p \rightarrow q_i : m_i \mid G_i \quad \overset{p \rightarrow q_j : m_j}{\longrightarrow} \quad G_j \quad (j \in I) \]

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

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

\[
\sum_{i \in I} p \xrightarrow{\epsilon_i} q_i : m_i ; G_i
\]

\[
G_1 \quad \vdots \quad G_n
\]

\[\mu_r \cdot \text{computes the mapping above} \]

\[\text{checks well-formedness of the resulting choreography automaton} \]

\[\text{generates the TypeScript API of each participant} \]
From global types to choreography automata

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

\[ p \rightarrow q_1 : m_1; G_1 \]

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

\[ G \]

\[ \mu. G \]

\[ G \]

\[ \varepsilon \]
From global types to choreography automata

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

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!