Design-by-Contract for *Flexible* Multiparty Session Protocols (Artifact)

Lorenzo Gheri
Imperial College London, UK

Ivan Lanese
Focus Team, University of Bologna, Italy
Focus Team, INRIA, Sophia Antipolis, France

Neil Sayers
Imperial College London, UK
Coveo Solutions Inc., Canada

Emilio Tuosto
Gran Sasso Science Institute, L'Aquila, Italy

Nobuko Yoshida
Imperial College London, UK

Abstract

We introduce CAScr, the first implementation of Scribble (`http://www.scribble.org`, `https://nuscr.dev/`) that relies on choreography automata, for deadlock-free distributed programming. CAScr supports the main theoretical results and constructions in the related article. CAScr takes the popular *top-down approach* to system development, based on choreographic models, following the original methodology of Scribble and multiparty session types. The top-down approach enables *correctness-by-construction*: a developer provides a global description for the whole communication protocol; by projecting the global protocol, APIs are generated from local CFSMs, which ensure the safe implementation of each participant. The theory of choreography automata in the related article guarantees deadlock freedom for the distributed implementation of flexible global protocols. We target web development, supporting in particular the TypeScript programming language.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Software and its engineering → Formal software verification

Keywords and phrases Choreography automata, design by contract, deadlock freedom, Communicating Finite State Machines, TypeScript programming

Digital Object Identifier 10.4230/DARTS.8.2.21

Funding Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233. Work partially funded by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems). Lanese and Tuosto are partially supported by INdAM as members of GNCS (Gruppo Nazionale per il Calcolo Scientifico). The work is partially supported by EPSRC EP/T006544/1, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/1, EP/N028201/1, EP/T014709/1 and EP/V000462/1, and NCSS/EPSRC VeTSS.

Acknowledgements We thank the anonymous reviewers for their useful comments and suggestions. We thank Franco Barbanera for contributing to this work in its early stages. We thank Fangyi Zhou for their help with building our artifact on top of their software, νScr.

Related Article Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, and Nobuko Yoshida, "Design-By-Contract for Flexible Multiparty Session Protocols", in 36th European Conference on Object-Oriented Programming (ECOOP 2022), LIPIcs, Vol. 222, pp. 8:1–8:28, 2022.
`https://doi.org/10.4230/LIPIcs.ECOOP.2022.8`

Related Conference 36th European Conference on Object-Oriented Programming (ECOOP 2022), June 6–10, 2022, Berlin, Germany

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2022 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact shows how the theory of choreography automata, with selective participation (Section 3 of the related article [1]), can be applied to the verification of distributed web programming in TypeScript, so that deadlock freedom is guaranteed.

Section 5 of [1] provides a broader discussion about our tool. In particular, two main elements of our theory realised in the implementation are:

- in `~/nuscr/lib/mpst/chorautomata.ml`, the theoretical definition of the function $\mathtt{ca}(G, q)$ in Section 5, which translates global types (Scribble protocols) into choreography automata, has been implemented as an algorithm that transforms Scribble protocols into choreography automata;
- The `OnlineWallet` example in the artifact (`~/case-studies/OnlineWallet`) incarnates a deadlock-free implementation of the recurring OLW example in the paper, from protocol description to the final web application.

More detail of our code is presented in Appendix A (also in the document code-structure.pdf); in particular, figure 1 is a more detailed version of Figure 4 in [1].

From a practical point of view, our framework can be immediately used to capture interesting examples, such as `OnlineWallet`. Our implementation provides a solid core for specifying deadlock-free web protocols that allow for selective participation, and automatically generates APIs for TypeScript implementations of arbitrary single-page applications.

2 Content

The artifact package includes:

- the docker image for our CAScr artifact (file cascript-artifact_dev_latest.tar.gz),
- a Readme.md file with the instructions on how to run the docker image, and
- a code-structure.pdf document describing the structure of our code (also in appendix to this document).

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://mega.nz/file/usBhiaTS#sDirwTFzj5_Uaq6JezXz3gKC_oQdTWlq7G98xwNNHe0`.

4 Tested platforms

The artifact can be run on Docker, version 20.10.10 or later. Specifically, it has been tested on Manjaro Linux 21.2.6.

5 License

The artifact is available under license GPLv3 (`https://www.gnu.org/licenses/gpl-3.0.en.html`).

6 MD5 sum of the artifact

bba5020e0c944a55be6951d8cba2acc3

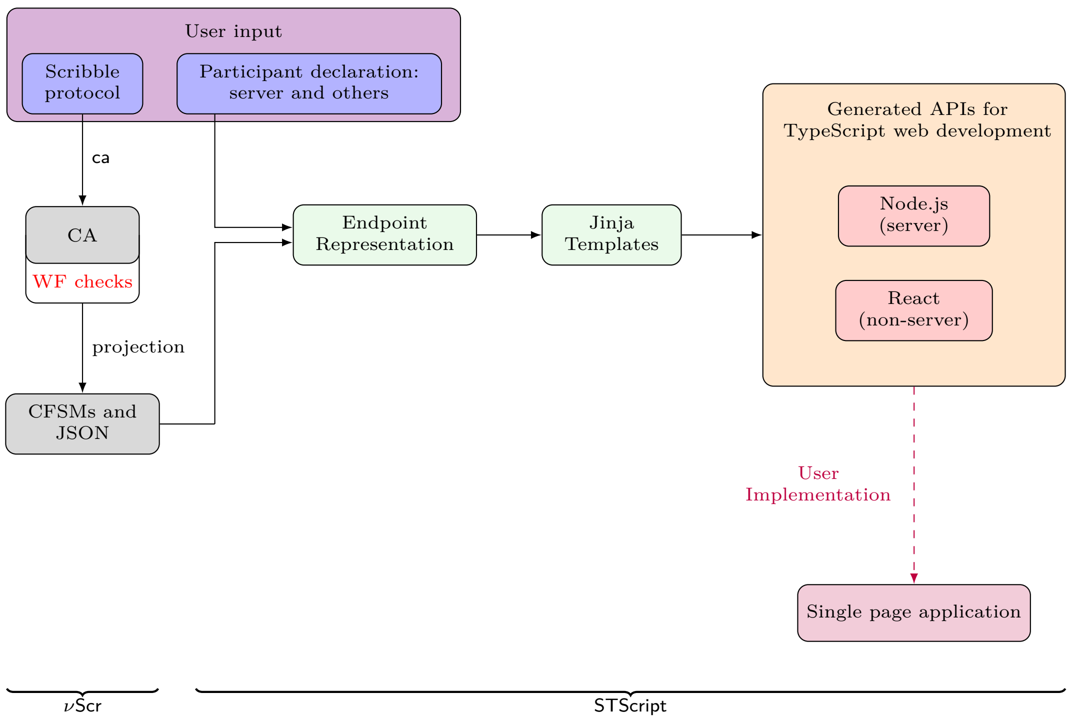

Figure 1 Toolchain for CAScr.

7 Size of the artifact

884.4 MB

A Structure of the Code

Figure 1 shows the structure of our tool CAScr, focusing on its main components. We first observe that the toolchain is obtained by an integration of two pre-existent tools νScr (`https://nuscr.dev/nuscr/`) and STScript (`https://github.com/STScript-2020/`). The codebase of νScr has been modified to accomodate for the generation of choreography automata and their projections, from the protocol description. We have exploited STScript for API generation: STScript is a previously developed tool that targets TypeScript and distributed web development. See also Section 5 of [1], for further discussion.

In what follows, we give pointers to where the components can be found in the codebase; also we point to the location where objects are generated along the execution of the toolchain, for the example protocols that we discuss.

- **User Input.** The user provides as input a Scribble Protocol. Sample Scribble Protocols can be found in the `protocols` folder. The tool is invoked from the command line, with slightly different syntaxes to generate the APIs for the server and for the other roles.

E.g., the syntax to generate the API for Wallet as a Node.js server in the OnlineWallet case study is:

```
python -m codegen \
    protocols/OnlineWallet.scr \
    OnlineWallet Wallet node \
    -o case-studies/OnlineWallet/src
```

In the code above, `protocols/OnlineWallet.scr` is the input file in which the Scribble Protocol `OnlineWallet` is specified, `Wallet` the role to consider, `node` specifies that we want to generate the server code, and the `-o` option gives the output folder.
The syntax for non-server roles, e.g., `Customer` and `Vendor` in the OnlineWallet case study, is as follows.

```
python -m codegen \
    protocols/OnlineWallet.scr \
    OnlineWallet Customer browser -s Wallet \
    -o case-studies/OnlineWallet/client/src
python -m codegen \
    protocols/OnlineWallet.scr \
    OnlineWallet Vendor browser -s Wallet \
    -o case-studies/OnlineWallet/client/src
```

The main differences are that one needs to specify `browser` instead of `node`, and to use the `-s` option to specify what role the server will be.
Default generation does not consider rule Pass when translating the Scribble protocol into a choreography automaton (CA), see discussion in Section 5 of [1]. This default can be overridden by using the `-pass` option when invoking the tool.
We provide as part of the artifact a `build_onlinewallet` script which runs the three commands above.

- **Choreography Automaton generation.** The first step in our tool chain is to take a Scribble Protocol and to generate the corresponding choreography automaton. This is part of the νScr component. The code to do that can be found in the `nuscr/lib/mpst/chorautomata.ml` file, and more precisely in the `of_global_type_total_with_pass` (or ...`without_pass` if rule Pass should not be used) function. νSct also performs the concurrency closure operation on the generated CA (an operation aiming at enforcing well-sequencedness when it does not hold, not described in [1]), using the `concurrency_closure` function. Finally, well-formedness is checked on the concurrency-closed CA (using function `is_well_branched`).
- **CFSMs and JSON.** The generation of CFSMs as abstract representation is customary in Scribble implementation. Localised representations of each role are necessary for code generation. From the concurrency-closed CA, local CFSMs describing single roles are projected. These are produced by νScr both in the DOT format for easy visualisation and as a JSON object that serialises the information about each transition (the roles sending and receiving the message, the payload types, payload names, etc.), making it much easier for CAScript later to process them. This is performed by the `chorautomata.ml` program: the CFSM projection is in the `project` function, and the JSON serialisation is produced by the `to_json` function.
- **Intermediate Representation.** The files describing the CFSM and the JSON information produced by νScr are then read by CAScript. This information has to be converted to Python objects. For the JSON, this is done by Python's `json` library, called in `codegen/cli.py`. For the graph, `cli.py` delegates to `codegen/automata/parser.py`, which uses the `pydot` library

to process the DOT graph, referring to the JSON for information on each edge. It generates a Python object called `Endpoint`, which includes an `EFSM` object, among other things. The terms EFSM and CFSM are used interchangeably.

- **Jinja Templates.** `Endpoint` gets passed through various Jinja templates; the templates are, for the web server role, in `codegen/generator/node/templates` and, for the other roles, in `codegen/generator/browser/templates`. These are TypeScript files with Jinja code (similar to Python) embedded in them. The Jinja code dynamically generates TypeScript files from the `Endpoint` objects.
- **Generated APIs.** Two kinds of APIs are generated.
  - For the server, a Node.js runtime is generated in `Runtime.ts`, which refers to `EFSM.ts`. This file includes a large TypeScript type representing the CFSM. Several auxiliary files, mostly needed for sharing information, are also created. To implement the web server, the developer must instantiate the runtime, and pass to it an object matching the type defined by `EFSM.ts`. The developer's program logic will lie within the implementation of that type.
  - For the browsers, several React components are generated. First, for each state in the CFSM, an abstract React component is produced. There are three possible generations for states: terminal, send, and receive. Each state will result in a file called `Sn.tsx`, where `n` is the state's number identifier. E.g., the first state will be `S0.tsx`. The extension `.tsx` denotes TypeScript code which incorporates HTML syntax, needed since React components are web UI elements. Then, a concrete React component named after the role is generated, for example `Vendor.tsx`. This takes as input all of that role's states, and manages transitions between them, as well as connecting to the server. To make the browser role for `Vendor`, for example, the developer must create React components implementing states `S0.tsx` to `S3.tsx`, then place the `Vendor.tsx` component in their webpage, and pass it all those implementations.
- **Implementation of the Case Studies.** Sample implementations for the case studies are stored in the `case-studies` folder.
- **Single Page Application.** For each case study, one can generate the files and build the application using scripts called `build_case-study-name`, e.g. `build_onlinewallet`. These scripts can be found in the `scripts` folder. Once built, one can run the case study by going to its folder and typing `npm start` in the console. Then, one can visit `https://localhost:8080/` to try the application. For the OnlineWallet case-study, one needs to open it using two browser tabs – one for the Customer, one for the Vendor.

References

1 Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, and Nobuko Yoshida. Design-by-Contract for Flexible Multiparty Session Protocols. In *36th European Conference on Object-Oriented Programming (ECOOP 2022)*, 2022.