API Generation for Multiparty Session Types, Revisited and Revised Using Scala 3 (Artifact)

Guillermina Cledou
HASLab, INESC TEC, Porto, Portugal
University of Minho, Braga, Portugal

Luc Edixhoven
Open University of the Netherlands, Heerlen, The Netherlands
NWO-I, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Sung-Shik Jongmans
Open University of the Netherlands, Heerlen, The Netherlands
NWO-I, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

José Proença
CISTER, ISEP, Polytechnic Institute of Porto, Portugal

Abstract

Construction and analysis of distributed systems is difficult. Multiparty session types (MPST) constitute a method to make it easier. The idea is to use type checking to statically prove deadlock freedom and protocol compliance of communicating processes. In practice, the premier approach to apply the MPST method in combination with mainstream programming languages has been based on API generation. With this artifact, and the companion paper, we *revisit* and *revise* this approach.

Regarding our *"revisitation"*, using Scala 3, we present the existing API generation approach, which is based on deterministic finite automata (DFA), in terms of both the existing states-as-classes encoding of DFAs as APIs, and a new states-as-type-parameters encoding; the latter leverages match types in Scala 3. Regarding our *"revision"*, also using Scala 3, we present a new API generation approach that is based on sets of pomsets instead of DFAs; it crucially leverages match types, too. Our fresh perspective allows us to avoid two forms of combinatorial explosion resulting from implementing concurrent subprotocols in the DFA-based approach. We implement our approach in a new API generation tool, called Pompset. This artifact contains Pompset.

2012 ACM Subject Classification Software and its engineering → Software notations and tools

Keywords and phrases Concurrency, pomsets (partially ordered multisets), match types, Scala 3

Digital Object Identifier 10.4230/DARTS.8.2.19

Funding *Guillermina Cledou*: Funded by the European Regional Development Fund (ERDF), and the Operational Programme for Competitiveness and Internationalisation (COMPETE 2020): POCI-01-0145-FEDER-029946 (project DaVinci).
Sung-Shik Jongmans: Funded by the Netherlands Organisation of Scientific Research (NWO): 016.Veni.192.103.
José Proença: Funded by the European Regional Development Fund (ERDF), and the Operational Programme for Competitiveness and Internationalisation (COMPETE 2020): POCI-01-0145-FEDER-029946 (project DaVinci); the Fundação para a Ciência e a Tecnologia (FCT), within the CISTER Research Unit: UIDP/UIDB/04234/2020; the ERDF and FCT, Portugal 2020 Partnership Agreement, Norte Portugal Regional Operational Programme (NORTE 2020): NORTE-01-0145-FEDER-028550 (REASSURE); and the ECSEL Joint Undertaking (JU): grant agreement No 876852 (VALU3S).

Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany

Related Article Guillermina Cledou, Luc Edixhoven, Sung-Shik Jongmans, and José Proença, "API Generation for Multiparty Session Types, Revisited and Revised Using Scala 3", in 36th European Conference on Object-Oriented Programming (ECOOP 2022), LIPIcs, Vol. 222, pp. 27:1–27:28, 2022. https://doi.org/10.4230/LIPIcs.ECOOP.2022.27

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

In our companion paper, through examples, we present a new encoding of multiparty session types (MPST) as APIs in Scala 3, using pomset-based formal semantics. Our artifact is an open-source proof-of-concept tool that is capable of applying this encoding to automatically generate APIs.

The MAIN CLAIM about the artifact's functionality in § 5 of our paper is that our tool can automatically generate APIs using an encoding "in accordance with § 4.2". We substantiate it in detail in Section A below. We note that the MAIN CLAIM is "qualitative" instead of "quantitative": the paper *does not* report on any performance experiments (run-times, memory usage, etc.). Besides the MAIN CLAIM, we list two additional features of our tool in § 5 of the paper:

- To enhance error messages, the tool inserts additional type constraints in generated APIs (not present in § 4). We explicitly indicate where these type constraints are placed when we discuss the generated code in § A below.
- To provide additional pomset support, the tool has visualisation and analysis capabilities.

2 Content

The artifact package includes:

- `README.md` – instructions to run/recompile/change the tool.
- `LICENSE` – the MIT license of the artifact.
- `paper.pdf` – a pre-print of the companion paper.
- `pompset.ova` – a virtual image (based on the TACAS'22 Artifact Evaluation VM, which is based on Ubuntu 20.04) with the tool installed and prepared to be recompiled.
- `pompset-bin` – the pre-compiled (JavaScript) documents and HTML files; to run the tool, open `pompset-bin/index.html` in a browser.
- `pompset-src` – the source-code.

3 Tested platforms

To run: This artifact requires only a web-browser with JavaScript.
To compile: One of the follow requirements must be met:

- Scala Building Tools (sbt) and Java Runtime Environment (JRE);
- Docker; or
- VirtualBox (or another virtualization program) to open an Open Virtual Appliance file (`.ova`).

4 License

The artifact is available under an MIT license, included in the `LICENSE` file.

5 MD5 sum of the artifact

557dab5a53589ecef7d64c1c13abce9d

6 Size of the artifact

- **Total (zipped):** 6.01 GiB
- Some of the files (unzipped):
  - **pompset.ova:** 6.08 GiB
  - **pompset-bin:** 14.54 MiB
  - **pompset-src:** 17.11 MiB

A Substantiating the Main Claim

The MAIN CLAIM about the artifact's functionality in § 5 of our paper is that our tool can automatically generate APIs using an encoding "in accordance with § 4.2". To substantiate the MAIN CLAIM:

- The tool contains the same examples as in § 4. In the tool, they can be loaded by clicking the corresponding buttons on the left. We note that the output of our tool (generated APIs) is *not completely identical* to the code in the listings in § 4 (e.g., the tool needs to apply name mangling, which we omitted from the paper for clarity); we already state this in § 5, though. However, there is still a strong similarity between the output of the tool and the examples in the paper. Specifically, the correspondence can be observed as follows:
  - Examples 16-17 ∼ "1Master-2Workers, relaxed"
    - Match type `M$Pom$Send`, `M$Pom$Recv`, `M$Pom$Final` in Examples 16-17 correspond with match types `M$Pom1$Send`, `M$Pom1$Recv`, and `M$Pom1$Final` in tab `M$State` of the generated code. (We note that the fork identifier is the first element of a configuration in Examples 16-17, while it is the last element of a configuration in the generated code.)
    - Match types `M$SendReturn`, `M$RecvArgument`, and `M$ForkReturn` in Examples 16-17 correspond with match types `M$SendReturn`, `M$RecvARgument`, and `M$ForkReturn` in tab `M$State` of the generated code.
    - The function that implements the master in Example 17 corresponds with the code in the yellow panel on the left.
    - To see the generated code, plus the implementation of the master and workers, in action:
      1. Open another tab in the browser and go to `https://scastie.scala-lang.org`. Scastie is a "playground" for Scala. (The generated code can also be integrated with Scala projects in an IDE, but to keep the artifact lightweight, we use Scastie here.)
      2. Copy the code in the "All" tab (manually, or using the "Copy" button in the top right of the code panel), and paste it into Scastie.
      3. Copy the code in the yellow panel on the left, and paste it into Scastie somewhere (e.g., below the previously pasted code).
      4. Click the "Run" button in Scastie. Some console output should be printed, meaning that the code was successfully compiled and run.

5. To force a compile-time error for protocol-incompliant behaviour, for instance:
   - Change `send(W1, Work)` into `send(W2, Work)` in `master` (i.e., the master sends work to worker 2 twice), and click the "Run" button again. An error message will appear in `master`.
   - Change `s1.recv(...)` into `s1` in `master` (i.e., the master does not receive from worker 1 but immediately returns), and click the "Run" button again. An error message will appear in `master`.
   - Add `s.send(W1, Work);` in `master` before `println("#1"); s` (i.e., the master sends extra work to worker 1 after receiving), and click the "Run" button again. An error message will appear in `master`.

- Example 18 $\sim$ "1Master-3Workers, relaxed"
  - The function that implements the master in Example 18 corresponds with the code in the yellow panel on the left.
  - To see the generated code, plus the implementation of the master and workers, in action, the same steps as above ("1Master-2Workers, relaxed") can be applied.
- Example 19 "Seller-Buyer, relaxed"
  - Match types `S$Pom1$Send`, `S$Pom1$Recv`, `S$Pom2$Send`, and `S$Pom2$Recv` in Example 19 correspond with `S$Pom1$Send`, `S$Pom1$Recv`, `S$Pom2$Send`, and `S$Pom2$Recv` in tab `S$State` of the generated code.
  - Match types `S$SendReturn` and `S$RecvArgument` in Example 19 correspond with `S$SendReturn` and `S$RecvArgument` in tab `S$State` of the generated code.
  - To see the generated code, plus the implementation of the master and workers, in action, the same steps as above ("1Master-2Workers, relaxed") can be applied.
- Additionally, "1Master-2Workers, Basic" and "Seller-Buyer, Basic" are oversequentialised versions (vs. the "relaxed" ones), as presented in § 1, Example 1-2. The remaining ten "anonymous" examples demonstrate several other communication patterns. (We note that "Ex. 7" and "Ex. 8" are actually unrealisable, as indicated by the realisability analysis.)

- The top-left input form of the tool allows the user to write its own global types from scratch.