Designing Asynchronous Multiparty Protocols with Crash-Stop Failures (Artifact)

Ping Hou
University of Oxford, UK

Ameen Izhac
University of Oxford, UK

Nobuko Yoshida
University of Oxford, UK

Abstract

As communication in distributed software systems grows increasingly complex, specifying *protocols* and maintaining their correctness under evolving requirements has become essential. This paper investigates an automatic transformation of distributed protocols, proposing general *validity conditions* that guarantee the correctness of the transformed protocols. We target *Multiparty Session Types* (MPST), where a global protocol (type) ensures that well-typed distributed programs communicate without type errors or stuck states – guaranteeing deadlock-freedom and liveness by construction – with a focus on enhancing *resilience* to failures in distributed systems. While various fault-tolerant session type theories have been proposed, existing approaches require programmers to explicitly write failure-handling constructs, a burdensome and error-prone task that risks incorrect or unintended behaviours.

We introduce three automatic transformations from global protocols without failure handling to fault-tolerant variants, each reflecting a distinct failure-handling or recovery strategy tailored to specific reliability requirements. The resulting protocols satisfy key validity conditions:

1. preservation of type safety, deadlock-freedom, and liveness; and
2. preservation of communication causality.

We implement these protocol transformations and API generation for Scala in our toolchain, Tutus, and evaluate it on MPST protocols, including real-world case studies. The results demonstrate that Tutus scales with protocol size and transformation complexity while maintaining practical overhead, highlighting the effectiveness of our approach.

2012 ACM Subject Classification Software and its engineering → Source code generation; Software and its engineering → Concurrent programming languages; Theory of computation → Process calculi; Theory of computation → Distributed computing models

Keywords and phrases Session Types, Concurrency, Failure Handling, Code Generation, Scala

Digital Object Identifier 10.4230/DARTS.9.2.9

Funding Work supported by: EPSRC EP/T006544/2, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/2, EP/N028201/1, EP/T014709/2, EP/V000462/1, EP/X015955/1, NCSS/EPSRC VeTSS, and Horizon EU TaRDIS 101093006.

Acknowledgements We thank the anonymous reviewers for their useful comments and suggestions. We thank Jia Qing Lim for his contribution to the Effpi extension. We thank Alceste Scalas for useful discussions and advice in the development of this paper and for his assistance with Effpi.

Related Article A.D. Barwell, P. Hou, N. Yoshida, F. Zhou, "Designing Asynchronous Multiparty Protocols with Crash-Stop Failures", in 37th European Conference on Object-Oriented Programming (ECOOP 2023), LIPIcs, Vol. 263, pp. 30:1–30:29, 2023.
`https://doi.org/10.4230/LIPIcs.ECOOP.2023.30`

Related Conference 37th European Conference on Object-Oriented Programming (ECOOP 2023), July 17-21, 2023, Seattle, Washington, United States

1 Scope

The artifact presents Tutus, a code generation toolchain supporting automatic crash-stop failure generation for Multiparty Session Type (MPST) protocols and Scala Effpi code generation. Tutus is written in Haskell and implements both global and local types from the related article.

Global types are derived from a subset of the Scribble syntax [2] accepted by νScr, extended to support our crash-handling model, which is consumed by Tutus as input. Protocol-conforming Scala code is generated via projection to local types. Runtime types, as indicated in the related article, are not supported in Tutus since these are not used when specifying protocols. Generated code uses an extended form of the Effpi concurrency library; although executable upon generation, the code can be extended and integrated with existing systems by the programmer.

The artifact contains protocol specifications for all examples presented in the related article. The artifact additionally includes dependencies and configuration files in order to facilitate the execution of generated code.

For more details, please consult Section 6 in the related article, Appendix G in the full version [1], and the `README` file in the artifact.

2 Content

The artifact is packaged as a Docker image, containing the source code of Tutus, our tool, and our extended Effpi concurrency library. The artifact also includes the benchmarks used in the paper to evaluate our toolchain.

We enumerate the contents of the home user directory (`/home/mpst/`) below (* indicates an executable file):

- `Lib/Tutus/`: contains the source code for our Tutus tool. We use the Stack build system.
- `build.sbt`: is the Scala sbt build file used to compile and run the generated code.
- `effpi/`: contains the extended Effpi concurrency library. Note that references to authors and/or copyright holders are to *original* authors and/or copyright holders of the library.
- `examples/`: contains example protocols. files in `effpi`.
- `project/`: configuration files used by `build.sbt`.
- `runScala.sh`*: script for running a single Scala file generated by Tutus.

The home user directory may also contain the below subdirectories.

- `scala/`: default output directory for generated code, produced by Tutus.
- `effpi_sandbox/`: used to run generated code, produced by `runScala.sh`.

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://zenodo.org/record/7974824`. The source files can be accessed at `https://github.com/AmeenIzhac/TutusArtefact`.

4 Tested platforms

The artifact has been tested under Linux (Ubuntu 22.04.01) and macOS (Ventura 13.3.1, M2). In principle, it should be able to run under a correct installation of Docker.

5 License

The artifact is available under the MIT licence (`https://opensource.org/license/mit/`).

6 MD5 sum of the artifact

94cc09960ca3a9558cc30925291eca5d

7 Size of the artifact

1.3 GiB

A Additional Information

For additional information, readers are invited to consult the `README.md` file in the Docker image, which contains information on how to use the artifact. Alternatively, the `README` file is available online at `https://github.com/AmeenIzhac/TutusArtefact/blob/main/README.md`.

B On Functionality and Reusability

The artifact is functional with respect to four aspects:

1. Tutus can be used to generate crash safe Scribble code from Scribble protocol specifications,
2. generate Scala code to implement multiparty protocols,
3. protocol specifications for all examples given in the related article are included, and
4. Tutus generates Scala code in negligible time.

Instructions for code generation can be found in the aforementioned `README.md` file. The example protocol specifications can be found in the corresponding directory in home user directory of the artifact. For our given examples, the time needed for code generation is within milliseconds. The raw results files when can be generated and found in `Lib/Tutus/bench`. These results were taken running Tutus directly on the reported test machine, and not within the Docker container. Accordingly, generation times taken within the docker container may be subject to increased variance due to the small average generation times. The user can run our benchmarks for all examples, using the instructions given in the `README.md` file.

Since Tutus takes Scribble protocol files as input, its application is not limited to those provided in the artifact, and is therefore reusable. The accepted syntax our Scribble language variant is described in the `README.md` file within the artifact, and the user may write communications protocols of their own. Tutus will benchmark a subset of protocol files in the `Lib/Tutus/bench/protocols` directory.

The code generated by Tutus is standard Scala, and can therefore be extended and integrated into novel or existing systems. To facilitate this, we include both our extended version of the Effpi concurrency library and the `sbt` build and configuration files needed to compile and run the generated code. Although default values are used for message payloads and branch selection when sending messages, these can be specialised with values and code meaningful to the user and wider system.

References

1 Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Designing asynchronous multiparty protocols with crash-stop failures. *CoRR*, abs/2305.06238, 2023. `arXiv:2305.06238`, `doi:10.48550/arXiv.2305.06238`.

2 Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The scribble protocol language. In *8th International Symposium on Trustworthy Global Computing - Volume 8358*, TGC 2013, pages 22–41, Berlin, Heidelberg, 2014. Springer-Verlag. doi:10.1007/978-3-319-05119-2_3.