Designing Interruptible Protocols in Distributed Systems

Leonid Nosovitskiy, Adam Barwell
University of St Andrews

Abstract

In today's interconnected world, distributed systems rely on communication protocols to coordinate interactions – systems range from trading (e.g. FIX [1]) to IoT (e.g. MQTT [2]). As systems evolve, verifying implementations becomes difficult because of message ordering bugs, deadlocks, and protocol mismatches, which cause severe failures. The Knight Capital incident (2012) [3] is an example: an update failed to reach one server, which executed a stale protocol and emitted unexpected messages, causing **$440 million** in losses in **45** minutes. Reports attribute the incident to human error, as there was no mechanism to automatically check code conformance to the new design. A related failure occurred at Nasdaq during Facebook's IPO [4], where faulty transaction choreography led to starvations and deadlocks, with Nasdaq paying **$10 million**.

Such errors arise not only from incorrect designs but also from ambiguity in natural-language specifications, which can be misinterpreted even when code appears correct. Multiparty Session Types (MPST) [5] address this by providing a formal framework to specify and statically check protocols, enforce specified patterns across multiple participants, while guaranteeing liveness and deadlock-freedom before execution.

Promising efforts to bring formal methods into industrial software development, include temporal logic at AWS [6] and MPST-based tools in several projects [7, 8], which generate code from protocol specifications for languages including *Scala* [9], *Java* [8], and *OCaml* [10]. However, adoption is still limited because some tools provided weak code-conformance support, assumed reliable processes, and impose usability barriers (requiring specialist expertise to understand the generated code and the practice of treating protocol specifications as write-once artifacts, which makes subsequent development difficult). While recent advances support conformance checking and fault-tolerance semantics [9], there are still barriers. Our research targets these, aiming to lower the entry cost, decreasing reliability assumptions, and supporting simpler modeling of complex patterns. For example, our toolchain [11] propagates protocol-level annotations into generated code skeletons so the compiler guides developers to fill in missing details (which also allows us to refactor existing code skeletons leaving concrete implementations untouched), and it supports semi-automatic refactoring to introduce fault tolerance into protocols originally designed for reliable processes.

Currently our toolchain supports straightforward protocol structures, but many distributed systems use more advanced mechanisms such as interrupts – for example, parallel farm-like protocols that drop stragglers [12] and ACID transactions that roll back on interruption [13]. Demangeon *et al.* [7] address interruptible protocols, but their approach can create orphan messages, lacks true recovery, and relies on atomic all-or-nothing interrupt multicast – which can leave participants inconsistent in faulty settings.

In our current work we aim to support fault-tolerant interruptible protocols in more realistic settings. Building on [7], we introduced a more expressive, language-level abstraction for interruptible blocks with recovery in *Scribble* [14], together with an encoding into established constructs. This removes the need for atomic all-or-nothing interrupt multicast assumptions, avoids orphan messages, and preserves state consistency. This is ongoing work; next we plan to provide correctness proof of the encoding, extend the encoding with parallel composition (e.g. [15]), and integrate our fault-tolerance refactoring [11]. We will evaluate the approach against benchmark implementations under both normal and faulty conditions.

References

[1] Y. Chauhan, "Financial information exchange (fix) protocol," https://medium.com/@yuvchauhan15/financial-information-exchange-fix-protocol-ec3bcd4a6edd, Jun. 2025, medium.

[2] "Mqtt: The standard for IoT messaging," https://mqtt.org/, 2024, accessed: 2025-11-13.

[3] H. Dolfing, "Case study 4: The $440 million software error at knight capital," 2019, accessed: 2025-03-13. [Online]. Available: https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

[4] U.S. Securities and Exchange Commission, "Sec charges nasdaq for failures during facebook ipo," https://www.sec.gov/newsroom/press-releases/2013-2013-95htm, June 2013.

[5] N. Yoshida and L. Gheri, “A Very Gentle Introduction to Multiparty Session Types,” in *16th International Conference on Distributed Computing and Internet Technology*, ser. LNCS, vol. 11969. Springer, 2020, pp. 73–93.

[6] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, “How amazon web services uses formal methods,” *Communications of the ACM*, 2015. [Online]. Available: https://www.amazon.science/publications/how-amazon-web-services-uses-formal-methods

[7] R. Demangeon, K. Honda, R. Hu, R. Neykova, and N. Yoshida, “Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python,” *FMSD*, pp. 1–29, 2015.

[8] N. Yoshida, “Programming Language Implementations with Multiparty Session Types,” in *Active Object Languages: Current Research Trends*. Springer Nature Switzerland, 2024, pp. 147–165.

[9] A. D. Barwell, P. Hou, N. Yoshida, and F. Zhou, “Designing asynchronous multiparty protocols with crash-stop failures,” *CoRR*, vol. abs/2305.06238, 2023.

[10] K. Imai, R. Neykova, N. Yoshida, and S. Yuen, “Multiparty Session Programming With Global Protocol Combinators,” in *34th European Conference on Object-Oriented Programming (ECOOP 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), R. Hirschfeld and T. Pape, Eds., vol. 166. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, pp. 9:1–9:30. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2020.9

[11] L. Nosovitskiy and A. D. Barwell, “Refactoring communication protocols for crash safety,” in *Informal Proceedings of the 11th International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE 2025)*, Birmingham, UK, Jul. 2025. [Online]. Available: https://wpte2025.github.io/pre-proceedings.pdf

[12] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, “Redundancy techniques for straggler mitigation in distributed optimization and learning,” *Journal of Machine Learning Research*, vol. 20, no. 72, pp. 1–47, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-148.html

[13] “Real-time deterministic database management,” McObject LLC, 33309 1st Way South, Suite A-208, Federal Way, WA 98003, USA, Tech. Rep.

[14] N. Yoshida, R. Hu, R. Neykova, and N. Ng, “The scribble protocol language,” in *Trustworthy Global Computing*, M. Abadi and A. Lluch Lafuente, Eds. Cham: Springer International Publishing, 2014, pp. 22–41. [Online]. Available: https://doi.org/10.1007/978-3-319-05119-2_3

[15] G. Cledou, L. Edixhoven, S.-S. Jongmans, and J. Proença, “API Generation for Multiparty Session Types, Revisited and Revised Using Scala 3,” in *36th European Conference on Object-Oriented Programming (ECOOP 2022)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), K. Ali and J. Vitek, Eds., vol. 222. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 27:1–27:28. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2022.27