Prisma: A Tierless Language for Enforcing Contract-Client Protocols in Decentralized Applications (Extended Abstract)

David Richter
Technical University of Damstadt, Germany

David Kretzler
Technical University of Damstadt, Germany

Pascal Weisenburger
University of St. Gallen, Switzerland

Guido Salvaneschi
University of St. Gallen, Switzerland

Sebastian Faust
Technical University of Damstadt, Germany

Mira Mezini
Technical University of Damstadt, Germany

Abstract

Decentralized applications (dApps) consist of smart contracts that run on blockchains and clients that model collaborating parties. dApps are used to model financial and legal business functionality. Today, contracts and clients are written as separate programs – in different programming languages – communicating via send and receive operations. This makes distributed program flow awkward to express and reason about, increasing the potential for mismatches in the client-contract interface, which can be exploited by malicious clients, potentially leading to huge financial losses. In this paper, we present Prisma, a language for tierless decentralized applications, where the contract and its clients are defined in one unit. Pairs of send and receive actions that "belong together" are encapsulated into a single direct-style operation, which is executed differently by sending and receiving parties. This enables expressing distributed program flow via standard control flow and renders mismatching communication impossible. We prove formally that our compiler preserves program behavior in presence of an attacker controlling the client code. We systematically compare Prisma with mainstream and advanced programming models for dApps and provide empirical evidence for its expressiveness and performance.

The design space of dApp programming and other multi-party languages depends on one major choice: a *local* model versus a *global* model. In a *local* model, parties are defined in separate programs and their interactions are encoded via send and receive effects. In a *global* language, parties are defined within one shared program and interactions are encoded via combined send-and-receive operations with no effects visible to the outside world. The global model is followed by tierless [19, 9, 4, 5, 11, 25, 26, 20, 27] and choreographic [13, 16, 12] languages. However, known approaches to dApp programming follow the local model, thus rely on explicitly specifying the client–contract interaction protocol. Moreover, the contract and clients are implemented in different languages, hence, developers have to master two technology stacks. The dominating approach in industry is Solidity [15] for the contract and JavaScript for clients. Solidity relies on expressing the protocol using assertions in the contract code, which are checked at run time [1]. Failing to insert the correct assertions may give parties illegal access to monetary values to the detriment of others [17, 14]. In research, contract languages [10, 6, 23, 24, 8, 7, 18, 3] have been proposed that rely on advanced type systems such as session types, type states, and linear types. The global model has not been explored for dApp programming. This is unfortunate given the potential to get by with a standard typing discipline and to avoid intricacies and potential mismatches of a two-language stack. Our work fills this gap by proposing Prisma – the first language that features a *global programming model* for Ethereum dApps. While we focus on the Ethereum blockchain, we believe our techniques

36th European Conference on Object-Oriented Programming (ECOOP 2022).
Editors: Karim Ali and Jan Vitek; Article No. 35; pp. 35:1–35:4
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to be applicable to other smart contract platforms. Prisma enables interleaving contract and client logic within the same program and adopts a *direct style (DS)* notation for encoding send-and-receive operations (with our **awaitCl** language construct) akin to languages with async/await [2, 22]. DS addresses shortcomings with the currently dominant encoding of the protocol's *finite state machines (FSM)* [15, 6, 23, 24, 8, 7]. We argue writing FSM style corresponds to a control-flow graph of basic blocks, which is low-level and more suited to be written by a compiler than by a human. With FSM style, the contract is a passive entity whose execution is driven by clients. whereas the DS encoding allows the contract to actively ask clients for input, fitting dApp execution where a dominant contract controls execution and diverts control to other parties when their input is needed.

In the following Prisma snippet, the `payout` function is a function invoked by the contract when it is time to pay money to a client. In Prisma, variables, methods and classes are separated into two namespaces, one for the contract and one for the clients. The `payout` method is located on the contract via the annotation `@co`. The body of the method diverts the control to the client using `awaitCl(...) { ... }`, hence the contained `readLine` call is executed on the client. Note that no explicit send/receive operations are needed but the communication protocol is expressed through the program control flow. Only after the check `client == toBePayed` that the correct client replied, the current contact balance `balance()` is transferred to the client via `transfer`.

```
@co def payout(toBePayed: Arr[Address]): Unit = {
  awaitCl(client => client == toBePayed) {
    readLine("Press enter for payout") }
  toBePayed.transfer(balance())
}
```

Overall, Prisma relieves the developer from the responsibility of correctly managing distributed, asynchronous program flows and the heterogeneous technology stack. Instead, the burden is put on the compiler, which distributes the program flow by means of selective continuation-passing-style (CPS) translation and defunctionalisation and inserts guards against malicious client interactions.

We needed to develop a CPS translation for the code that runs on the Ethereum Virtual Machine (EVM) since the EVM has no built-in support for concurrency primitives which could be used for asynchronous communication. While CPS translations are well-known, we cannot use them out-of-the-box because the control flow is interwoven with distribution in our case. A CPS translation that does not take distribution into account would allow malicious clients to force the contract to deviate from the intended control flow by sending a spoofed continuation. Thus, it was imperative to prove correctness of our *distributed CPS translation* to ensure control-flow integrity of the contract.

2012 ACM Subject Classification Software and its engineering → Distributed programming languages; Software and its engineering → Domain specific languages; Software and its engineering → Compilers

Keywords and phrases Domain Specific Languages, Smart Contracts, Scala

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2022.35

Related Version *Technical Report*: https://arxiv.org/abs/2205.07780 [21]

Supplementary Material *Software*: https://github.com/stg-tud/prisma

Funding This work has been funded by the German Federal Ministry of Education and Research *iBlockchain project* (BMBF No. 16KIS0902), by the German Research Foundation (DFG, SFB 1119 *CROSSING Project*), by the BMBF and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity *ATHENE*, by the Hessian LOEWE initiative (*emergenCITY*), by the Swiss National Science Foundation (SNSF, No. 200429), and by the University of St. Gallen (IPF, No. 1031569).

References

1. Solidity documentation - common patterns. https://docs.soliditylang.org/en/v0.7.4/common-patterns.html, 2020. Accessed 14-11-2020.
2. Gavin M. Bierman, Claudio V. Russo, Geoffrey Mainland, Erik Meijer, and Mads Torgersen. Pause 'n' play: Formalizing asynchronous c#. In James Noble, editor, *ECOOP 2012 - Object-Oriented Programming - 26th European Conference, Beijing, China, June 11-16, 2012. Proceedings*, volume 7313 of *Lecture Notes in Computer Science*, pages 233–257. Springer, 2012. doi:10.1007/978-3-642-31057-7_12.
3. Sam Blackshear, Evan Cheng, D. Dill, Victor Gao, B. Maurer, T. Nowacki, Alistair Pott, S. Qadeer, Dario Russi, Stephane Sezer, Tim Zakian, and Run tian Zhou. Move: A language with programmable resources. 2019.
4. Kwanghoon Choi and Byeong-Mo Chang. A theory of RPC calculi for client–server model. *Journal of Functional Programming*, 29, 2019. doi:10.1017/S0956796819000029.
5. Kwanghoon Choi and Byeong-Mo Chang. A theory of RPC calculi for client-server model. *J. Funct. Program.*, 29:e5, 2019. doi:10.1017/S0956796819000029.
6. Michael J. Coblenz. Obsidian: a safer blockchain programming language. In Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard, editors, *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, pages 97–99. IEEE Computer Society, 2017. doi:10.1109/ICSE-C.2017.150.
7. Michael J. Coblenz, Gauri Kambhatla, Paulette Koronkevich, Jenna L. Wise, Celeste Barnaby, Jonathan Aldrich, Joshua Sunshine, and Brad A. Myers. User-centered programming language design in the obsidian smart contract language. *CoRR*, abs/1912.04719, 2019. URL: http://arxiv.org/abs/1912.04719, arXiv:1912.04719.
8. Michael J. Coblenz, Reed Oei, Tyler Etzel, Paulette Koronkevich, Miles Baker, Yannick Bloem, Brad A. Myers, Joshua Sunshine, and Jonathan Aldrich. Obsidian: Typestate and assets for safer blockchain programming. *CoRR*, abs/1909.03523, 2019. URL: http://arxiv.org/abs/1909.03523, arXiv:1909.03523.
9. Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. Links: Web programming without tiers. In *Proceedings of the 5th International Conference on Formal Methods for Components and Objects*, FMCO'06, pages 266–296, Berlin, Heidelberg, 2007. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1777707.1777724.
10. Ankush Das, S. Balzer, J. Hoffmann, and F. Pfenning. Resource-aware session types for digital contracts. *ArXiv*, abs/1902.06056, 2019.
11. Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova. Exceptional asynchronous session types: Session types without tiers. *Proceedings of the ACM on Programming Languages*, 3(POPL):28:1–28:29, January 2019. doi:10.1145/3290341.
12. Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. Choreographies as objects, 2020. arXiv:2005.09520.
13. Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In Raja Natarajan and Adegboyega K. Ojo, editors, *Distributed Computing and Internet Technology - 7th International Conference, ICDCIT 2011, Bhubaneshwar, India, February 9-12, 2011. Proceedings*, volume 6536 of *Lecture Notes in Computer Science*, pages 55–75. Springer, 2011. doi:10.1007/978-3-642-19056-8_4.
14. Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2976749.2978309.
15. Mix. These are the top 10 programming languages in blockchain. https://thenextweb.com/hardfork/2019/05/24/javascript-programming-java-cryptocurrency/, 2019. Accessed 14-11-2020.

16 Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Service-oriented programming with jolie. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foundations*, pages 81–107. Springer, 2014. doi:10.1007/978-1-4614-7518-7_4.

17 Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, page 653–663, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3274694.3274743.

18 Reed Oei, Michael J. Coblenz, and Jonathan Aldrich. Psamathe: A DSL with flows for safe blockchain assets. *CoRR*, abs/2010.04800, 2020. URL: https://arxiv.org/abs/2010.04800, arXiv:2010.04800.

19 Christian Queinnec. The influence of browsers on evaluators or, continuations to program web servers. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, pages 23–33. ACM, 2000. doi:10.1145/351240.351243.

20 Gabriel Radanne, Jérôme Vouillon, and Vincent Balat. Eliom: A core ML language for tierless web programming. In Atsushi Igarashi, editor, *Proceedings of the 14th Asian Symposium on Programming Languages and Systems*, APLAS '16, pages 377–397, Berlin, Heidelberg, November 2016. Springer-Verlag. doi:10.1007/978-3-319-47958-3_20.

21 David Richter, David Kretzler, Pascal Weisenburger, Guido Salvaneschi, Sebastian Faust, and Mira Mezini. Prisma: A tierless language for enforcing contract-client protocols in decentralized applications (extended version), 2022. URL: https://arxiv.org/abs/2205.07780, doi:10.48550/ARXIV.2205.07780.

22 Scala. Scala async rfc. http://docs.scala-lang.org/sips/pending/async.html.

23 Franklin Schrans, Susan Eisenbach, and Sophia Drossopoulou. Writing safe smart contracts in flint. In Stefan Marr and Jennifer B. Sartor, editors, *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, April 09-12, 2018*, pages 218–219. ACM, 2018. doi:10.1145/3191697.3213790.

24 Franklin Schrans, Daniel Hails, Alexander Harkness, Sophia Drossopoulou, and Susan Eisenbach. Flint for safer smart contracts. *CoRR*, abs/1904.06534, 2019. URL: http://arxiv.org/abs/1904.06534, arXiv:1904.06534.

25 Manuel Serrano, Erick Gallesio, and Florian Loitsch. Hop, a language for programming the web 2.0. In *Companion to the 21th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA Companion '06, New York, NY, USA, 2006. ACM.

26 Manuel Serrano and Vincent Prunet. A glimpse of Hopjs. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP '16, pages 180–192, New York, NY, USA, 2016. ACM. doi:10.1145/2951913.2951916.

27 Pascal Weisenburger, Mirko Köhler, and Guido Salvaneschi. Distributed system development with ScalaLoci. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):129:1–129:30, October 2018. doi:10.1145/3276499.