A Language Engineering Approach to Support the P4 Coding Ecosystem

Alexandre Lachance, Sébastien Mosser

McMaster Centre for Software Certification (McSCert)
McMaster University, Faculty of Engineering, Dept. of Computing and Software
Hamilton, Ontario, Canada – {lachaa2, mossers}@mcmaster.ca

Context. P4 as a language is becoming a de facto standard to support the definition of *Software Defined Networks* (SDNs). The research and industrial communities associated with P4 are blooming, with an intense effort dedicated to language definition/formalization [1], optimized compilation [2], static code analysis [3,4,5], … However, from a P4 developer point of view, the language suffers from this effervescence: available tools are not necessarily interoperable, and more importantly, the research effort focuses today on the language, not on the developers who use it daily. In this context, the *Kaloom-TELUS-ETS Research Chair on DevOps for SDNs* is investigating how the DevOps paradigm from the software engineering community can be leveraged and adapted to the specificities of SDNs. The underlying idea of DevOps is to put developments and operations at the same level in a continuous loop. By focusing on culture more than on technology, a DevOps-based approach relies on three "ways" [6]: *(i)* identifying flow and reifying such flows into continuous delivery pipelines, *(ii)* leveraging the flows and creating a fast continuous feedback loop, and finally *(iii)* use this feedback to create opportunities to learn continuously.

Challenge. The specificities of SDNs trigger challenges in applying DevOps principles (i.e., supporting the "three ways") to such an ecosystem. As DevOps is about creating a culture of "*continuous feedback*", contributing to this direction for SDNs means identifying which kind of *feedback* is relevant and what *continuous* means in the context of P4 development. The critical point here is to provide tools and technologies to support a shift from a language-centred process (i.e., compiler driven) to a developer-centred one (i.e., *software language engineering*, SLE).

Contribution. Leveraging our expertise in embedded language design [7] and continuous feedback loops for developer-centred ecosystems [8,9], we initiated a discussion with the industrial research chair research partners to identify how SLE approaches could support this thriving ecosystem from a DevOps point of view. Among the different leads identified at this stage, the most immediate result was to realize how diverse is the P4 community. Such diversity means that any attempt to put the P4 developers at the center of the ecosystem must be made with integration in mind. To support such a vision, we propose to the P4 community an integrative Language Server-based approach, depicted in Fig. 1.

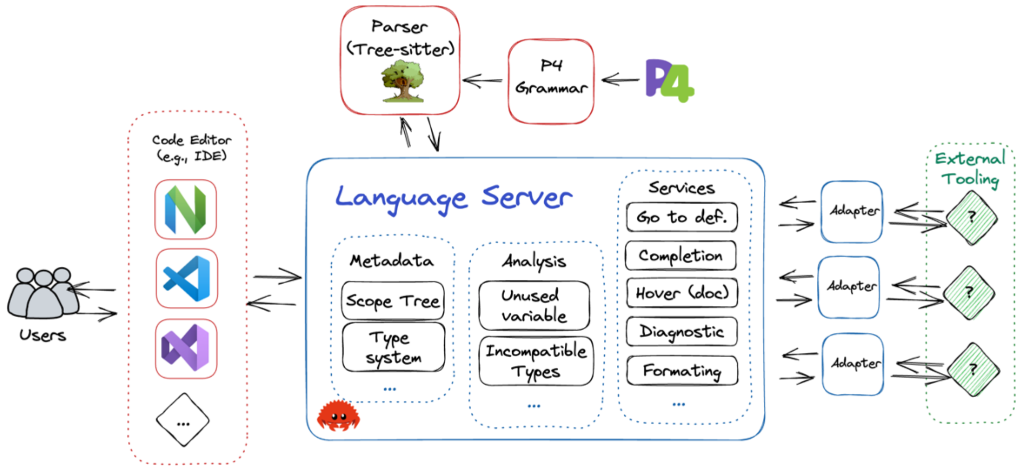

Figure 1. Architecture of the proposed P4 Language Server integrative approach.

This architecture relies on a Language Server as a pivot element between *(i)* the users of the language, (ii) a lightweight representation of the language grammar design to support a short feedback loop, and *(iii)* external tooling that is developed independently. A classical way to deliver services to developers is to use a *Language Server Protocol* (LSP), a technology popularized by Microsoft in 2016 that supports the standard communication between an editor and a language server in charge of providing language-specific services such as syntax highlighting, formatting, or navigation. The server relies on an abstract representation of the P4 file obtained by directly translating the P4 standard into a parser. Where the reference compiler (p4c) is a fully-fledged compilation toolchain targeting multiple architectures, the abstract representation used as an underlying foundation is designed with the rationale of supporting analysis to provide feedback to the developers. It complements the compilation toolchain in the ecosystem by providing a lighter representation that a language engineer can use to write a diagnostic (e.g., unused variable, incompatible types assignment). According to the integration-driven rationale of the architecture, these analyses are developed following an inversion of control pattern: the language server acts here as a framework that provides context to each analysis, and the language engineer in charge of designing their analysis has to focus on their very problem, thanks to a “plugin” approach. In addition to these analyses (exposed as “diagnostic” according to the LSP terminology), classical linting services can be offered (e.g., detection of bad smells, violation of best practices), as well as classical navigability mechanisms (e.g., go-to declaration, go-to definition). Finally, some static analyses can require external tooling: for example, control-flow analysis can leverage graph databases [3], or fault detection can use Z3 [5]. Consequently, the architecture relies on the adapter pattern to provide “hooks” one can plug external tooling into the language server. Again, the server acts as a framework here, delegating to the registered external analysis the computation of the diagnostics return to the users.

Results. We started the development of the architecture recently. Following an agile approach, we first focused on developing a walking skeleton. The objective of this proof of concept is to demonstrate the three components of the architecture: integration through LSP, lightweight P4 representation to support analysis, and finally, integration of external tooling. To date, our reference implementation is integrated into the NeoVim editor and offers services such as syntax highlighting and variable renaming. From the analysis point of view, we propose a basic type compatibility validation and a demonstration of how external tooling can be integrated. This (preliminary) reference implementation is available as an open-source framework (https://github.com/ace-design/p4-lsp). The server is implemented in Rust, a language providing safety-by-design guarantees. The lightweight representation of the P4 language is defined as a Tree-Sitter grammar (a reference tool suite to design such elements). We have also released an open-sourced version of the language as a tree-sitter module (https://github.com/ace-design/tree-sitter-p4) based on the latest version of the P4 standard**. In addition to this talk proposal, we can offer a tabletop demonstration of the integrative capabilities of the approach**.

Conclusions. The P4 language is at the center of a vibrant environment. We propose in this talk to describe a reference architecture that aims to support the integration of various approaches and aiming to put the developer back at the centre of the development ecosystem, following the DevOps principles of continuous feedback.

Authors. *Alexandre Lachance* is graduate student at McMaster. His research is related to supporting static analysis and code coverage tooling for the P4 language as part of the Kaloom-TELUS-ETS industrial research chair on DevOps for SDNs. *Sébastien Mosser* (PEng, Ph.D.), is an Associate Professor of Software Engineering at McMaster and an executive member of the McSCert research centre. His research interests are software design, DevOps, scalability, modelling and language engineering.

References

[1] R. Doenges, M. Tahmasbi Arashloo, S. Bautista, A. Chang, N. Ni, A. Parkinson, R. Peterson, A. Solko-Breslin, A. Xu, and N. Foster. 2021. **Petr4: formal foundations for P4 data planes**. Proc. ACM Program. Lang. 5, POPL, Article 41 (January 2021), 32 pages.

[2] P. Wintermeyer, M. Apostolaki, A. Dietmüller, and L. Vanbever. 2020. P2GO: **P4 Profile-Guided Optimizations**. In Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20). Association for Computing Machinery, New York, NY, USA, 146–152.

[3] D. Lukács, G. Pongrácz, M. Tejfel**. Are Graph Databases Fast Enough for Static P4 Code Analysis?** ICAI 2020: 213-223

[4] D. Lukács, G. Pongrácz, M. Tejfel. **Control flow based cost analysis for P4**. *Open Computer Science*, vol. 11, no. 1, 2021, pp. 70-79.

[5] F. Ruffy, T. Wang, A. Sivaraman: Gauntlet. **Finding Bugs in Compilers for Programmable Packet Processing**. CoRR abs/2006.01074 (2020)

[6] G. Kim, P. Debois, J. Willis, and J. Humble. 2016. **The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations**. IT Revolution Press.

[7] S. Bonnieux, D. Cazau, S. Mosser, M. Blay-Fornarino, Y. Hello, G. Nolet. **MeLa: A Programming Language for a New Multidisciplinary Oceanographic Float**. Sensors 20(21): 6081 (2020)

[8] S. Mosser, V. Reihnarz, and C. Pulgar. **Modelling Agile Backlogs as Composable Artefacts to support Developers and Product Owners**. *Journal of Object Technology., 2022.*

[9] A. Lapointe-Boisvert, S. Mosser, S. Trudel. **Towards Modelling Acceptance Tests as a Support for Software Measurement**. MoDELS (Companion) 2021: 827-832