Software and Systems Modeling (2023) 22:847–850
https://doi.org/10.1007/s10270-023-01101-9

EXPERT VOICE

A personal retrospective on language workbenches

Mark van den Brand[1]

Received: 6 January 2023 / Revised: 6 February 2023 / Accepted: 13 February 2023 / Published online: 20 March 2023

Abstract
Model-driven software engineering and specifically domain-specific languages have contributed to improve the quality of software and the efficiency in the development of software. However, the design and implementation of domain-specific languages requires still an enormous investment. Language workbenches are the most important tools in the field of software language engineering. The introduction of language workbenches has alleviated partly the development effort, but there are still a few major challenges that need to be tackled. This paper presents a personal perspective on the development of tools for language engineering and language workbenches in particular and future challenges to be tackled.

Keywords Language engineering · DSLs · Programming environment generators · Language workbenches

1 EAGs and pregmatic

In 1985, I followed the course “Vertalerbouw 2” (Compiler Construction 2) at the Radboud University Nijmegen. In this course, the Extended Affix Grammar (EAG) formalism was introduced. EAGs were closely related to Attribute Grammars; they allowed the syntactic and semantic definition of programming languages. Next to this topic, we were free to choose a compiler construction-related topic to write an essay, and I choose the topic of *Programming Environment Generators*. For this purpose, I read the PhD thesis of Thomas Reps on *Generating Language-Based Environments* [1]. This was my starting point of a long journey in software language engineering and tool development. During my MSc graduation project, I developed a prototype of a programming environment generator based on EAGs. This work was the basis for my PhD thesis “Pregmatic: A Generator For Incremental Programming Environments” [2]. The use of EAGs allowed me to experiment with semantics directed parsing in relation to generated programming environments. EAGs supported both the definition of the syntax of a language and the definition of semantic functions. The exchange values between nodes in the syntax trees were facilitated via *affixes*. The semantic functions in combination with the affixes provided a mechanism to specify simple semantic rules, mainly related to type checking, but more complex semantic rules turned out to be challenging.

2 ASF+SDF Meta-Environment

After my PhD, I moved to the University of Amsterdam and started working in the group of Paul Klint. The group was developing the ASF+SDF Meta-Environment [3], a programming environment generator built on top of the Centaur System [4]. The ASF+SDF Meta-Environment supported the definition of (programming) languages where the syntax of the language was defined via the Syntax Definition Formalism (SDF) and the semantics via the Algebraic Specification Formalism (ASF). The Meta-Environment was a fully integrated environment that provided editors for the SDF and ASF parts. It was a closed environment from a language definition point of view because there were no back-doors to use other languages to define aspects of a language.

The ASF+SDF Meta-Environment was used to do research on, among others, legacy software and domain-specific languages (DSLs) [5]. My contributions were on pretty printing [6] and later on the development of the ASF2C compiler [7]. In both cases, I used ASF+SDF and the Meta-Environment as implementation vehicles.

Communicated by Benoit Combemale, Juan de Lara, and Romina Eramo.

✉ Mark van den Brand
m.g.j.v.d.brand@tue.nl

[1] Department of Mathematics and Computer Science, Eindhoven University of Technology, Groene Loper, 5612 AZ Eindhoven, The Netherlands

After 5 years at the University of Amsterdam, my scientific journey continued at CWI and contributed to a new version of the ASF+SDF Meta-Environment which was no longer based on Centaur. Also, I got more involved in the research on domain-specific languages. The new version of the ASF+SDF Meta-Environment was used in research, education and industrial projects related to DSL design and reverse engineering [5].

The learning curve for ASF+SDF was rather steep. The SDF formalism was a great vehicle to define the syntax of languages, because of its modularity and the underlying powerful SGLR parsing technology [8]. However, definition of the semantics of languages still proved to be the real challenge. Although ASF allowed for abstraction and modularity in the definition of semantic aspects of a language, it was often tedious to define the full semantics, including a code generator.

3 LDTA, SLE, and language workbenches

At the beginning of 2000, the scientific community on software language engineering started to take shape and to become independent of the regular Compiler Construction community. In 2001, the workshop series on Language Descriptions, Tools, and Applications (LDTA)[1] started. In 2008, this workshop was merged with the International Workshop on Language Engineering (ATEM) and continued as the International Conference on Software Language Engineering (SLE).[2] LDTA and SLE were stepping stones to create a community of researchers and practitioners working on software language engineering together. In these workshops and conferences, software language engineering researchers and tool builders gathered to present their latest research results and showcased their tools developed for improving and easing the adoption of software language engineering, such as Spoofax [9], JastAdd [10], MontiCore [11], Silver [12], MPS [13], GEMOC [14], Rascal [15], and ANTLRWorks/Xtext[3] [16].

The software language engineering tools became *language workbenches (LWBs)*, a term coined by Martin Fowler [17] around 2005. The term was immediately picked up by the software language engineering community because it brought together the developments in domain-specific languages and integrated development environments (IDEs). The SLE community organized a few language workbench challenges [18,19], with the goal to show the strengths (and weaknesses) of the various tools. The aforementioned list of language engineering tools, or LWBs, is a mix of academic tools and some tools used in industry, such as MPS and Xtext. The transfer of features explored in the academic tools to the industrialized tools could have been stronger.

4 DSLs in the high-tech industry

In 2006, I moved from CWI to the Eindhoven University of Technology. This move was not only geographically, but I moved from being a developer of language engineering tools to become a user of language engineering tools as well. First, I started teaching a course on Generic Language Technology, initially based on ASF+SDF and the ASF+SDF Meta-Environment; later, I started using Eclipse and Xtext and finally Rascal. For each of these language workbenches, students experienced a steep learning curve. However, once students grasped the basic principles, they were able to create interesting and challenging (small) DSLs. The Eindhoven area is characterized by a large number of companies working on high-tech equipment. This turned out to be a perfect environment for promoting software language engineering. Several companies were interested in adopting DSLs. These companies developed and used DSLs because there was a need to increase the level of abstraction, moving from, for instance, C code to models in the DSL [20]. This involves the development of non-trivial code generators, and the development of these generators is costly and time-consuming. The code generators need to be tested and validated. The lack of properly defined semantics of the DSL hinders developing and testing of the code generators.

Collaborations between the local industry and our university started and led to interesting research projects. There are two examples that I want to highlight. The first is the work by Ulyana Tikhonova. She worked on the development of semantic building blocks for DSLs within ASML [21]. In this project, she applied these semantic building blocks to define the dynamic semantics of a DSL used for describing the scheduling of tasks related to the processing of wafers. The semantic building blocks were mapped to Event-B which allowed for verification of the dynamic semantics of the DSL using the Event-B tools [22].

The other example is the use of MPS at Canon Production Printing (The Netherlands) and their way of working. They have a group of engineers with a broad knowledge on MPS and software language engineering. When a new DSL is required for a specific domain/application area within Canon Production Printing, engineers are teamed up with the domain experts [23].

5 My final reflections

A lot of research has been performed in the area of semantics (e.g., denotational semantics, operational semantics, and action semantics) of programming languages with very inter-

[1] http://ldta.info/.

[2] http://www.sleconf.org/.

[3] https://www.eclipse.org/Xtext/.

esting and useful results. The transfer of these results to DSL design and language workbenches has been hampered by the fact that the descriptions and implementations are too large. Ulyana Tikhonova's work was a small step in the creation of re-usable semantic building blocks to formalize the semantics of DSLs in a very specific application area. Eelco Visser followed a different approach in Spoofax, by defining small DSLs to describe different semantic aspects such as name resolution, scoping rules, type checking, and the dynamic semantic rules [24,25]. The current generation of language workbenches still have a strong focus on syntax, but the mechanisms to define the (static and dynamic) semantics of DSLs are still experimental, although GEMOC [14] offers facilities for executing and debugging DSLs, which in other LWBs are either lacking or primitive.

The Canon Production Printing example illustrated another shortcoming in relation to the design and implementation of DSLs. Although multiple text books on programming languages, language design, and software language engineering exist, among others, [17,26,27], there is no text book that proposes a methodology for designing and developing DSLs. Before creating a DSL, it is important to identify and understand the application area or (problem) domain, the goal of the language, the involved stakeholders, and the technical environment. There is an overlap with regular requirements engineering except the software language engineering makes it more complicated. For instance, what are the language concepts needed by the domain or what is the best (textual/graphical) representation of the language concepts for the end-users? Multiple iterations may be needed to obtain a usable DSL. Most of the existing literature solely focuses on the technical challenges when creating DSLs. The existing language workbenches are excellent tools to specify a DSL, but do not support the above mentioned steps before the actual creation of a DSL. The current generation of language workbenches have a strong focus on language engineer tool smiths but do not really support end-users. Recent developments in the area of language workbenches for block-based languages are a promising step [28].

Acknowledgements I would to thank Mauricio Verano Merino and Ivan Kurtev for proofreading and providing feedback.

References

1. Reps, T.W.: Generating Language-Based Environments. MIT Press, Cambridge (1984)
2. van den Brand, M.G.J.: PREGMATIC—a generator for incremental programming environments. PhD thesis, Radboud University Nijmegen (1992)
3. Klint, P.: A meta-environment for generating programming environments. ACM Trans. Softw. Eng. Methodol. **2**(2), 176–201 (1993). https://doi.org/10.1145/151257.151260
4. Borras, P., Clément, D., Despeyroux, T., Incerpi, J., Kahn, G., Lang, B., Pascual, V.: CENTAUR: the system. In: Henderson, P.B. (ed.) Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Massachusetts, USA, 28–30 Nov, 1988, pp. 14–24 (1988). https://doi.org/10.1145/64135.65005
5. van den Brand, M., van Deursen, A., Klint, P., Klusener, S., van der Meulen, E.: Industrial applications of ASF+SDF. In: Wirsing, M., Nivat, M. (eds.) Algebraic Methodology and Software Technology, 5th International Conference, AMAST '96, Munich, Germany, 1–5 July, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1101, pp. 9–18 (1996). https://doi.org/10.1007/BFb0014303
6. van den Brand, M., Visser, E.: Generation of formatters for context-free languages. ACM Trans. Softw. Eng. Methodol. **5**(1), 1–41 (1996). https://doi.org/10.1145/226155.226156
7. van den Brand, M., Heering, J., Klint, P., Olivier, P.A.: Compiling language definitions: the ASF+SDF compiler. ACM Trans. Program. Lang. Syst. **24**(4), 334–368 (2002). https://doi.org/10.1145/567097.567099
8. Visser, E.: Syntax definition for language prototyping. PhD thesis, University of Amsterdam (1997)
9. Wachsmuth, G., Konat, G.D.P., Visser, E.: Language design with the spoofax language workbench. IEEE Softw. **31**(5), 35–43 (2014). https://doi.org/10.1109/MS.2014.100
10. Ekman, T., Hedin, G.: The JastAdd system—modular extensible compiler construction. Sci. Comput. Program. **69**(1–3), 14–26 (2007). https://doi.org/10.1016/j.scico.2007.02.003
11. Krahn, H., Rumpe, B., Völkel, S.: Monticore: a framework for compositional development of domain specific languages. Int. J. Softw. Tools Technol. Transf. **12**(5), 353–372 (2010). https://doi.org/10.1007/s10009-010-0142-1
12. Wyk, E.V., Bodin, D., Gao, J., Krishnan, L.: Silver: an extensible attribute grammar system. Electron. Notes Theor. Comput. Sci. **203**(2), 103–116 (2008). https://doi.org/10.1016/j.entcs.2008.03.047
13. Voelter, M.: Language and IDE modularization and composition with MPS. In: Lämmel, R., Saraiva, J., Visser, J. (eds.) Generative and Transformational Techniques in Software Engineering IV, International Summer School, GTTSE 2011, Braga, Portugal, 3–9 July, 2011. Revised Papers. Lecture Notes in Computer Science, vol. 7680, pp. 383–430 (2011). https://doi.org/10.1007/978-3-642-35992-7_11
14. Combemale, B., Barais, O., Wortmann, A.: Language engineering with the GEMOC studio. In: 2017 IEEE International Conference on Software Architecture Workshops, ICSA Workshops 2017, Gothenburg, Sweden, 5–7 Apr, 2017, pp. 189–191 (2017). https://doi.org/10.1109/ICSAW.2017.61
15. Klint, P., van der Storm, T., Vinju, J.J.: EASY meta-programming with rascal. In: Fernandes, J.M., Lämmel, R., Visser, J., Saraiva, J. (eds.) Generative and Transformational Techniques in Software Engineering III—International Summer School, GTTSE 2009, Braga, Portugal, 6–11 July, 2009. Revised Papers. Lecture Notes in Computer Science, vol. 6491, pp. 222–289 (2009). https://doi.org/10.1007/978-3-642-18023-1_6

16. Bovet, J., Parr, T.: Antlrworks: an ANTLR grammar development environment. Softw. Pract. Exp. **38**(12), 1305–1332 (2008). https://doi.org/10.1002/spe.872
17. Fowler, M.: Domain-Specific Languages. The Addison-Wesley signature series (2011). http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0321712943,00.html
18. Erdweg, S., van der Storm, T., Völter, M., Boersma, M., Bosman, R., Cook, W.R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., Konat, G.D.P., Molina, P.J., Palatnik, M., Pohjonen, R., Schindler, E., Schindler, K., Solmi, R., Vergu, V.A., Visser, E., van der Vlist, K., Wachsmuth, G., van der Woning, J.: The state of the art in language workbenches—conclusions from the language workbench challenge. In: Erwig, M., Paige, R.F., Wyk, E.V. (eds.) Software Language Engineering—6th International Conference, SLE 2013, Indianapolis, IN, USA, 26–28 Oct, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8225, pp. 197–217 (2013). https://doi.org/10.1007/978-3-319-02654-1_11
19. Erdweg, S., van der Storm, T., Völter, M., Tratt, L., Bosman, R., Cook, W.R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., Konat, G.D.P., Molina, P.J., Palatnik, M., Pohjonen, R., Schindler, E., Schindler, K., Solmi, R., Vergu, V.A., Visser, E., van der Vlist, K., Wachsmuth, G., van der Woning, J.: Evaluating and comparing language workbenches: existing results and benchmarks for the future. Comput. Lang. Syst. Struct. **44**, 24–47 (2015). https://doi.org/10.1016/j.cl.2015.08.007
20. Mooij, A.J., Hooman, J., Albers, R.: Gaining industrial confidence for the introduction of domain-specific languages. In: IEEE 37th Annual Computer Software and Applications Conference, COMPSAC Workshops 2013, Kyoto, Japan, 22–26 July, 2013, pp. 662–667 (2013). https://doi.org/10.1109/COMPSACW.2013.83
21. Tikhonova, U., Manders, M., van den Brand, M., Andova, S., Verhoeff, T.: Applying model transformation and event-b for specifying an industrial DSL. In: Boulanger, F., Famelis, M., Ratiu, D. (eds.) Proceedings of the 10th International Workshop on Model Driven Engineering, Verification and Validation MoDeVVa 2013, Co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, Florida, USA, October 1st, 2013. CEUR Workshop Proceedings, vol. 1069, pp. 41–50 (2013). http://ceur-ws.org/Vol-1069/07-paper.pdf
22. Tikhonova, U.: Reusable specification templates for defining dynamic semantics of DSLs. Softw. Syst. Model. **18**(1), 691–720 (2019). https://doi.org/10.1007/s10270-017-0590-0
23. Schindler, E., Moneva, H., van Pinxten, J., van Gool, L., van der Meulen, B., Stotz, N., Theelen, B.: Jetbrains MPS as core DSL technology for developing professional digital printers. In: Bucchiarone, A., Cicchetti, A., Ciccozzi, F., Pierantonio, A. (eds.) Domain-Specific Languages in Practice: With JetBrains MPS, pp. 53–91 (2021). https://doi.org/10.1007/978-3-030-73758-0_3
24. Konat, G.D.P., Kats, L.C.L., Wachsmuth, G., Visser, E.: Declarative name binding and scope rules. In: Czarnecki, K., Hedin, G. (eds.) Software Language Engineering, 5th International Conference, SLE 2012, Dresden, Germany, 26–28 Sept, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7745, pp. 311–331 (2012). https://doi.org/10.1007/978-3-642-36089-3_18
25. Vergu, V.A., Neron, P., Visser, E.: Dynsem: A DSL for dynamic semantics specification. In: Fernández, M. (ed.) 26th International Conference on Rewriting Techniques and Applications, RTA 2015, Warsaw, Poland. LIPIcs, vol. 36, pp. 365–378 (2015). https://doi.org/10.4230/LIPIcs.RTA.2015.365
26. Lämmel, R.: Software Languages: Syntax, Semantics, and Metaprogramming (2018). https://books.google.nl/books?id=mi1bDwAAQBAJ
27. Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C.L., Visser, E., Wachsmuth, G.: DSL engineering-designing, implementing and using domain-specific languages (2013). http://www.dslbook.org
28. Merino, M.V., van Wijk, K.: Workbench for creating block-based environments. In: Fischer, B., Burgueño, L., Cazzola, W. (eds.) Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2022, Auckland, New Zealand, 6–7 Dec, 2022, pp. 61–73 (2022). https://doi.org/10.1145/3567512.3567518

Mark van den Brand is a full professor of Software Engineering and Technology in the Department of Mathematics and Computer Science and a visiting professor at Royal Holloway, University of London. His current research activities are on model driven engineering, domain-specific languages, meta-modeling, model management, digital twins, and automotive software engineering. His research is industry inspired; he works with most of the high-tech companies in the Eindhoven (The Netherlands) region. He is project leader of a research project on Digital Twins, the focus in this project on the orchestration and management of models involved in the Digital Twins. He has been an invited lecturer and keynote speaker at various conferences, workshops, and doctoral schools. He was and is member of PCs on workshops and conferences related to software engineering, language engineering, rewriting, reverse engineering, and software maintenance. He initiated the special issues of Science of Computer Programming devoted to academic software development (Experimental Software and Toolkits) and since 2007 has been guest editor of six of these special issues. He is on the editorial board of the journals Science of Computer Programming and Computer Languages (COLA). He is deputy Editor-in-Chief of platinum open access journal JOT. He is one of the Editors-in-Chief of the open access Journal of Software Engineering for Autonomous Systems (JSEAS).