Languages for Non-developers: What, How, Where? Invited Talk—Extended Abstract

Juha-Pekka Tolvanen
0000-0002-6409-5972
MetaCase,
Jyväskylä, Finland
jpt@metacase.com

Languages To Raise the Level of Abstraction

PRODUCTIVITY has improved each time programming languages have raised the level of abstraction. This trend continues today with languages that narrow the scope they address, referred to as domain-specific languages (DSLs) [1]. However, many of these DSLs are built by developers for developers and tend to focus on the solution domain rather than the problem domain. These languages typically use text as the specification language, are built on top of IDE tools used by programmers and rely on diff and merge of files for collaboration.

In this talk, we will focus on languages that are more closely aligned with the problem domain rather than the solution domain, thereby addressing the needs of domain experts. Such languages not only raise the level of abstraction beyond programming but also enable non-developers to capture and communicate their knowledge, and, together with appropriate tools, support testing, validation, and feedback. This is important as research has consistently shown that common reasons for project failures, budget overruns, and similar issues are often related to limited understanding and formulating requirements and to the lack of user involvement. By using languages that are close to the problem domain, many typical development tasks—especially those related to requirements specification, checking, and validation—can be performed by non-developers. In many cases, the specifications created by domain experts can also be used to automatically generate code, configurations, tests, deployment instructions, and more.

Characteristics of Non-developer Languages

The talk is based on a review of over 200 industry cases ([2], see Figure 1) involving the creation and use of domain-specific languages with MetaEdit+ tool [3]. MetaEdit+ enables users to create and use domain-specific modeling languages and generators.

Interestingly, most of the analyzed DSLs were developed for use by non-programmers (see Figure 2), which inspired the title of this talk. We will present examples of non-developer languages, such as those used by usability experts, safety engineers, security engineers, insurance experts and instrumentation experts. These sample languages will illustrate how they differ from traditional programming languages or DSLs created for developers and programmers: Languages for non-developers are designed to align more closely with specific domains, representing knowledge through maps, diagrams, matrices, tables, and their combinations rather than plain text alone.

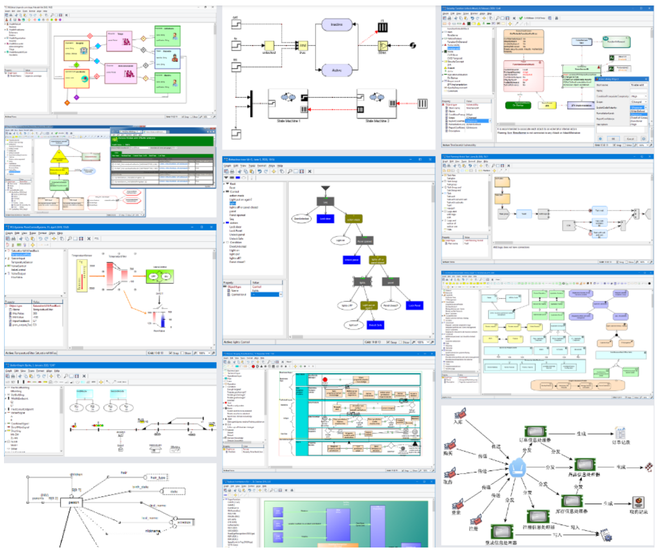
Fig. 1. Examples of DSLs reviewed

We will highlight key findings from the reviewed DSLs, including who created them (Figure 3), their size relative to standardized modeling languages like UML (Figure 4), and whether languages created for domain experts are smaller or larger than those intended for use by programmers, or if the role of the language creator influences size of the language.

How to Create Languages for Non-developers

In the second part of the talk, we will discuss how creation of non-developer languages differs from that of programming languages. While publications on domain-specific languages typically focus on their abstract syntax, defined through metamodels or grammars, we will emphasize aspects relevant to languages used by domain experts, such as the importance of concrete syntax (e.g. following guidelines like those in [4]) and the provision of support for language use, including guidance, animation, and error and warning reporting [5]—features often

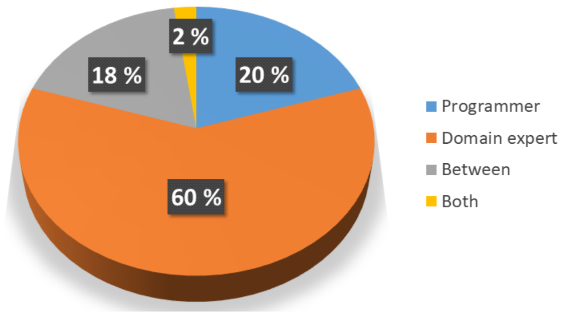

Fig. 2. Primary language user (n=45)

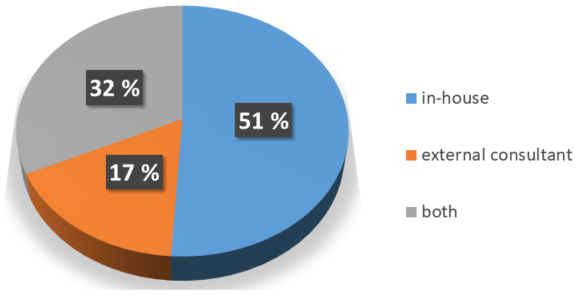

Fig. 3. Who implements domain-specific languages (n=100)

not covered in language specifications (see e.g. languages standardized by OMG, ITU, or The Open Group). Regarding industrially used languages, we will focus on two critical aspects of using DSLs: enabling user participation during language creation and supporting the evolution of the languages, along with the co-evolution of the work made using previous versions of the language [6].

Tool and Process Support

Computer languages also require tools, as tools can transform precise knowledge representations made with DSLs into software code and other artifacts. We will examine the history of tools used for creating DSLs, including the latest advances in this area [7] as well as the effort to create DSLs [8]. Tools are also essential for enabling collaboration, but it is often unrealistic to apply approaches used in traditional programming, such as IDE tools and diffing and merging of files, to languages used by non-developers. Domain experts and non-developers expect tools that are easier to learn and

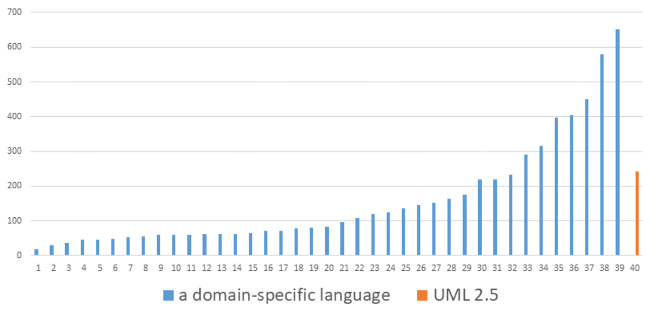

Fig. 4. Size of languages - in terms of size of their metamodel (n=39)

use, as they do not typically use their languages on a daily basis, unlike developers who use programming languages. Domain experts also expect that collaboration, viewing, and managing changes in the artifacts will be simpler and more closely aligned with the problem domain, rather than focusing on tracking changes through character differences within files. Not only the languages but also the entire process related to using them needs to focus on the problem domain.

We conclude by envisioning the role of non-developers in language creation and identifying situations where DSLs are most suitable, as well as pointing areas where they may not be applicable.

References

[1] Fowler, M. 2008. "Domain-Specific Languages", Addison-Wesley
[2] MetaCase. 2024. "DSL of the week", https://www.facebook.com/media/set/?set=a.2102426129807641 (accessed August 2024)
[3] MetaCase. 2023. "MetaEdit+ 5.5 User's Guides", https://metacase.com/support/55/manuals/ (accessed August 2024)
[4] D. Moody. 2009. The "Physics" of Notations, IEEE Transactions on Software Engineering, vol. 35, no. 6
[5] S. Kelly, J-P. Tolvanen. 2021. "Automated Annotations in Domain-Specific Models: Analysis of 23 Cases". STAF Workshops
[6] J-P. Tolvanen and S. Kelly. 2023. "Evaluating Tool Support for Co-Evolution of Modeling Languages, Tools and Models", ACM/IEEE MODELS Conference companion
[7] M. Ozkaya and D. Akdur. 2021. "What do practitioners expect from the meta-modeling tools? A survey", Journal of Computer Languages, Vo 63
[8] J-P. Tolvanen and S. Kelly. 2018. "Effort Used to Create Domain-Specific Modeling Languages". ACM/IEEE Conference on Model Driven Engineering Languages and Systems