Choosing the Best Level of Abstraction for Your Domain-Specific Language

Juha-Pekka Tolvanen

MetaCase
Ylistönmäentie 31, FI-40500 Jyväskylä, Finland
jpt@metacase.com, www.metacase.com

Abstract

Aligning a language closer to the problem to be tackled, regardless if the language is internal or external, represented as a graphical, textual, map, matrix etc. will offer major improvements. Empirical studies [1, 4, 5, 6] have reported improvements for example in error detection and prevention, quality of code, productivity, maintainability and in having better communication and understanding of the system developed.

The biggest improvements, as reported by industry experiences [10], are achieved when the level of abstraction is raised as high as possible: Appropriate language-level abstraction is the same as the problem domain – or as close as possible to the problem domain. Based on partly public cases [e.g. 2, 6, 7, 8, 9, 11] we describe results from developing and testing different type of software, such as consumer electronics, telecommunication, product lines, medical, and automation systems, with domain-specific languages, models and generators. These industry cases report 5-10 fold productivity increase.

These cases also show that when raising the level of abstraction closer to the actual problem domain, traditional textual approach, dominating programming language design, is not necessarily the most suitable way to specify ideas and find solutions in the problem domain. Instead the domain and the most "natural" way to express and find solutions in that domain should guide the language designers. For example, graphical block languages of Simulink and LabView are today de-facto way for engineers in signal processing and in control engineering, spreadsheets are heavily used in accounting, telecom standards are specified in MSCs of SDL, etc. Using a suitable representation for the language also helps in making programming easier.

We argue that the domain – area of interest – that the language addresses should drive the language design, not the capabilities of a particular implementation technology or tool. Still DSLs can't survive without infrastructure such as tooling support. Too often companies have started to build this infrastructure along their DSLs, just to find out few years later that tool development is not their core business. And that building the tooling support took more time and effort that expected – yet to remember that tools need to be maintained too.

Since DSLs expressing usually some part of a company's domain are the core knowledge, not the tools, language workbenches provide one solution. They enable

creating the infrastructure needed, such as editors, checkers and generators (for a comparison of the creation functionality see [3]). Some of these tools also provide advanced editing and refactoring capabilities, support for maintaining DSL definitions, sharing DSLs to the developers, updating specifications when the DSL changes, scaling development to larger teams including concurrent editing support etc. We are not aware of studies which compare language workbenches beyond creating relatively small languages. Such studies are warmly welcomed and would greatly contribute to the DSL community.

References

1. Kieburtz, R. et al., A Software Engineering Experiment in Software Component Generation, 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, March, (1996)
2. Kelly, S., Tolvanen, J-P., Domain-Specific Modeling: Enabling full code generation, Wiley-IEEE Society Press (2008)
3. El Kouhen, A., Dumoulin, C., Gerard, S., Boulet, P., Evaluation of Modeling Tools Adaptation, http://hal.archives-ouvertes.fr/hal-00706701/ (2012)
4. Kärnä et al., Evaluating the use of DSM in embedded UI application development, Procs of DSM'09 at OOPSLA, 2009.
5. Lan, C., Ramesh, B., Rossi, M., Are Domain-Specific Models Easier to Maintain Than UML Models? Software, IEEE , vol.26, no.4, pp.19,21, July-Aug. (2009)
6. Leitner, A., Preschern, C., Kreiner, C., Effective development of automation systems through domain-specific modeling in a small enterprise context, Journal of Software and Systems Modeling, October (2012)
7. Puolitaival, O-P., Kanstrén, T., Rytky, V-M., Saarela, A., Utilizing Domain-Specific Modelling for Software Testing, Procs of the 3rd International Conference on Advances in System Testing and Validation Lifecycle (2011)
8. Safa, L., The Making Of User-Interface Designer: A Proprietary DSM Tool, Procs of DSM'07 at OOPSLA, 2007.
9. Schulz, S., Tolvanen, J-P., Domain-Specific Modeling with MBT - Experiences from Web Application Testing, ETSI test conference (2012)
10. Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D., What Kinds of Nails Need a Domain-Specific Hammer?, IEEE Software, July/Aug (2009)
11. Weiss, D., Lai, R., Software Product Line Engineering: A Family-based Software Development Process, Addison-Wesley (1999)