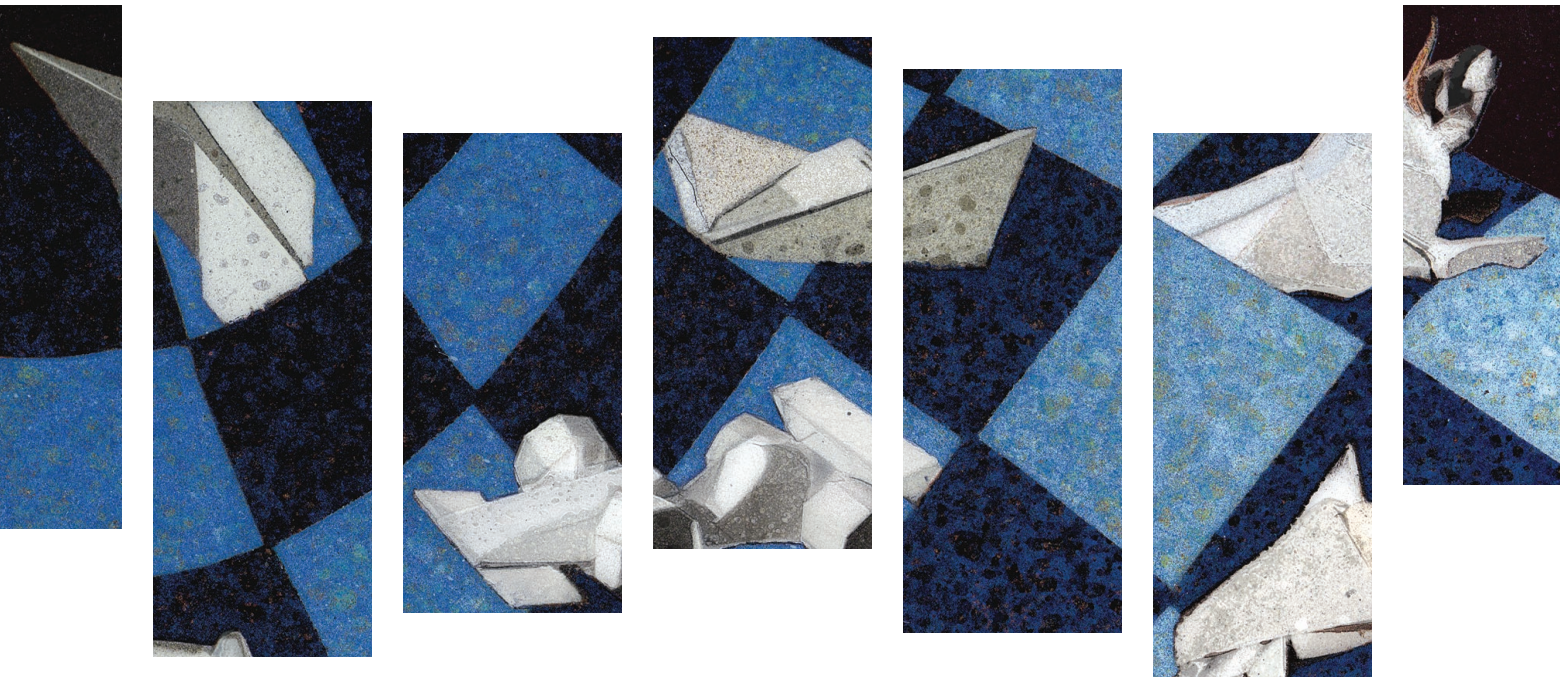

What Kinds of Nails Need a Domain-Specific Hammer?

Jonathan Sprinkle, *University of Arizona*

Marjan Mernik, *University of Maribor*

Juha-Pekka Tolvanen, *MetaCase*

Diomidis Spinellis, *Athens University of Economics and Business*

The landscape of software engineering is littered with languages that were supposedly the next great thing but failed to take the development world by storm. In contrast, many tools actually have changed the technical space in which certain domains expect to operate. Simulink and LabView are touchstones for designers in signal processing and in control. SolidWorks is the lingua franca of mechanical engineers and roboticists building physical devices. Embedded-hardware developers passionately support either VHDL (VHSIC Hardware Description Language)

0740-7459/09/$25.00 © 2009 IEEE

The key way to leverage the benefits of DSL&M approaches is to look for opportunities to employ them.

or Verilog. Some of these tools use textual languages, whereas others follow graphical notations.

Most of us outside these domains have never used these tools or languages, and more than a few of us might never have heard of them. Yet these tools are successful in their niches because each one

- satisfies its domain's requirements and
- streamlines development by restricting user input to parameters within the domain while providing easy access to artifacts (for example, files, plots, and generated code) that help users design or implement these systems.

Domain-specific techniques, languages, tools, and models aren't new: Fortran and Cobol can easily be viewed as domain-specific languages for scientific and business computing, respectively. Their domain is just very wide. What has changed is the technology for creating domain-specific languages (DSLs). Now it's easier to define languages and get tool support for narrower domains—for example, specifying insurance products or developing home automation systems. Such focus offers increased abstraction, making development faster and easier.

In domain-specific approaches, developers construct solutions from concepts representing things in the problem domain, not concepts of a given general-purpose programming language. Ideally, a DSL follows the domain abstractions and semantics as closely as possible, letting developers perceive themselves as working directly with domain concepts. The created specifications might then represent simultaneously the design, implementation, and documentation of the system, which can be generated directly from them. The mapping from the high-level domain concepts to implementation is possible because of the domain specificity: the language and code generators fit the requirements of a narrowly defined domain.

Characteristics of Problems Deserving a DSL&M Approach

Here's a checklist for determining whether a problem merits a DSL&M (domain-specific languages and modeling) approach:

- The domain is well defined.
- The domain has repetitive elements or patterns, such as multiple products, features, or targets.
- The developer community is growing (which usually means a maturing business area and the need for domain-specific notations).
- A clear path exists from requirements analysis and specification to execution.
- Accuracy; expert involvement; and flexibility of the specification, verification, and validation of design are important.
- An intuitive or well-accepted representation is already defined.
- The implementation or specification must serve as documentation.
- The implementation details might be subject to change, but the specification semantics is clear.
- Use by a domain expert (not necessarily a software expert) is intended.
- Amortization of effort justifies investment in DSL&M creation.

The more of these characteristics the problem exhibits, the more likely it will merit a DSL&M solution. However, in some cases, only one characteristic might apply to a problem, but that characteristic is significant enough for the problem to merit a DSL&M solution. Such a trade-off must be carefully considered.

Amortization of Effort

DSL&M allows raising the level of abstraction to hide today's programming languages, in the same way that today's programming languages hide an assembler. Two issues, though, are how much effort goes into developing the domain-specific infrastructure and how long you can use it with your domain.

When amortizing the effort of using DSL&M solutions, you must consider the entire life cycle:

- the effort to create and maintain the language and related code generators,
- the effort and cost to obtain and maintain tool support for the language, and
- the effort for domain experts to learn the language.

You must also weigh that entire effort against these issues:

- productivity increase compared to general-purpose solutions,
- quality improvement compared to general-purpose solutions, and
- the number of expected users or implementations.

If you determine that DSL&M is an appropriate choice, you're ready to select from the various tools to make your design and implementation plan concrete.

How to Leverage Benefits

The key way to leverage the benefits of DSL&M approaches is to look for opportunities to employ them. Refuse the bland conformity of a general-purpose language, and always search for a better way to code a specific requirement. If you find that the general-purpose language's abstractions can't provide the expressiveness you need, it's probably because a DSL is trying to get your attention. Similarly, if you find that you frequently describe your designs using visualizations that are clear to implementers but aren't part of the UML standard, you're itching for a form of domain-specific modeling.

Common DSL&M tools include

- GME (Generic Modeling Environment), www.isis.vanderbilt.edu/projects/gme;
- GMF (Graphical Modeling Framework), www.eclipse.org/gmf;
- LISA (*L*anguage *I*mplementation *S*ystem Based on *A*ttribute Grammars), marcel.uni-mb.si/lisa;
- MetaEdit+, www.metacase.com;
- the Meta-Environment (ASF + SDF [algebraic specification formalism and syntax definition formalism]), www.meta-environment.org; and
- Microsoft DSL tools, code.msdn.microsoft.com/DSLToolsLab.

Designing and implementing a small DSL from scratch is often quite easy. Scripting languages make it straightforward to parse a simple line or XML–based format into a general-purpose language (or another DSL) for existing compilers. In the right hands, this approach can be extremely powerful.

Are DSL&M Technologies Ready for Large-Scale Problems?

Talking about how good a technology could be is nothing compared to showing results. DSL&M solutions have produced many significant results in various domains, including automotive manufacturing, digital signal processing, mobile devices, telecommunications, home automation, and electrical utilities. In terms of quantifiable improvements, Nokia has reported 10× productivity improvement from coding to DSM,[1] Panasonic has reported 5× improvement,[2] Lucent has reported 5× to 10× improvement depending on the domain,[3] and empirical studies have reported 3× improvement—with a significance level of 99 percent.[4]

DSL&M technologies are also qualitatively improving design and implementation by reducing development resources, increasing capability, and changing how systems interact. This record extends past academic problems to include large-scale US government acquisitions,[5] automotive[6,7] and avionics[8] software, command and control systems,[4] secure networks,[9] information-integrated education,[10] medical treatment,[11] autonomous-vehicle development,[12] and many other domains (for examples, see www.dsmforum.org/cases.html). This extends the information technology impact of DSL&M approaches, which has spawned innovations in software product lines.

Not Every Nail Needs This Hammer, ...

... but when DSL&M approaches are applicable, they can greatly decrease the cost of developing software and systems. DSL&M technologies aren't a panacea, and in many cases the initial effort required to create a DSL&M solution might exceed the effort to apply the general-purpose solution. However, the effort that goes beyond the code's development, including maintenance and

About the Authors

Jonathan Sprinkle is an assistant professor of electrical and computer engineering at the University of Arizona. In 2009, he received the university's Ed and Joan Biggers Faculty Support Grant for work in autonomous systems. He previously was the executive director of the Center for Hybrid and Embedded Software Systems at the University of California, Berkeley. His research interests and experience are in systems control and engineering, through modeling and metamodeling. Sprinkle has a PhD in electrical engineering from Vanderbilt University and is a member of the IEEE, ACM, and American Institute of Aeronautics and Astronautics. Contact him at sprinkle@ece.arizona.edu.

Marjan Mernik is a professor in the University of Maribor's Department of Computer Science, where he leads the Programming Methodologies Group. His research interests include programming languages, compilers, grammar-based systems, grammatical inference, and evolutionary computation. Mernik has a PhD in computer science from the University of Maribor and is a member of the IEEE and ACM. Contact him at marjan.mernik@uni-mb.si.

Juha-Pekka Tolvanen is the CEO of MetaCase. He has been involved in domain-specific approaches, notably metamodeling and modeling tools, since 1991. He has acted as a consultant worldwide for modeling-language and code generation development and coauthored *Domain-Specific Modeling: Enabling Full Code Generation* (John Wiley & Sons, 2008). Tolvanen has a PhD in computer science from the University of Jyväskylä. Contact him at jpt@metacase.com.

Diomidis Spinellis is an associate professor in the Athens University of Economics and Business's Department of Management Science and Technology. His research interests include software engineering tools, programming languages, and computer security. He's the author of *Code Reading: The Open Source Perspective* (Addison-Wesley, 2003) and editor of *IEEE Software*'s Tools of the Trade column. Spinellis has a PhD in computer science from Imperial College, University of London. Contact him at dsl@istlab.dmst.aueb.gr.

documentation, often outweighs initial cost estimates. When calculating a DSL&M solution's costs, you must consider the whole development lifetime. Such issues need careful examination, to determine whether the DSL&M infrastructure's contributions can be amortized past the initial development.

As new application areas embrace the impact of software, the need exists for more and different kinds of nails. This opens the door to different kinds of languages, models, and tools that can make an immediate impact in the area. Given the low overhead needed to create DSL&M solutions, they can enable innovative designers to rapidly develop high-impact solutions.

References

1. *Nokia Mobile Phones Case Study*, MetaCase, 2007; www.metacase.com/papers/MetaEdit_in_Nokia.pdf.
2. L. Safa, "The Making of User-Interface Designer, A Proprietary DSM Tool," *Proc. 7th OOPSLA Workshop Domain-Specific Modeling*, tech. report TR-38, Univ. of Jyväskylä, 2007; www.dsmforum.org/events/DSM07/papers/safa.pdf.
3. D. Weiss and C.T.R. Lai, *Software Product-Line Engineering*, Addison Wesley Longman, 1999.
4. R. Kieburtz et al., "A Software Engineering Experiment in Software Component Generation," *Proc. 18th Int'l Conf. Software Eng.* (ICSE 96), IEEE CS Press, 1996, pp. 542–552.
5. "Future Combat Systems Program Completes Integrated Mission Test-1," press release, Boeing, 26 Feb. 2009; www.boeing.com/news/releases/2009/q1/090226b_nr.html.
6. "The MathWorks and Vector Integrate Tools for Model-Based Design and Autosar Applications," press release, 25 Mar. 2009; www.mathworks.com/company/pressroom/articles/article33724.html.
7. G. Karsai, *Automotive Software: A Challenge and Opportunity for Model-Based Software Development*, LNCS 4147, Springer, 2006.
8. M. Schulte, "Model-Based Integration of Reusable Component-Based Avionics Systems—a Case Study," *Proc. 8th IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing* (ISORC 05), IEEE CS Press, 2005, pp. 62–71.
9. *EADS Case Study*, MetaCase, 2007; www.metacase.com/papers/MetaEdit_in_EADS.pdf.
10. R.J. Roselli et al., "Integration of an Intelligent Tutoring System with a Web-Based Authoring System to Develop Online Homework Assignments with Formative Feedback," *Proc. Am. Soc. for Eng. Education Ann. Conf.*, Am. Soc. for Eng. Education, 2008.
11. J.L. Mathe et al., "A Model-Integrated, Guideline-Driven, Clinical Decision-Support System," *IEEE Software*, vol. 26, no. 4, 2009, pp. 54–61.
12. J. Sprinkle et al., "Model-Based Design: A Report from the Trenches of the DARPA Urban Challenge," *Software and Systems Modeling*, Online First, 2009; http://springerlink.metapress.com/content/r4jg67t1177q76w7/fulltext.pdf.