### CHAPTER 16

## CONCLUSION

Domain-Specific Modeling (DSM) is nothing new. Rather, it gathers together a number of existing techniques into a cohesive whole. Some of the techniques, like component frameworks, are well known and widely used. Others, like creating your own graphical languages, have previously been used by a much smaller number of people. Many of these techniques have been hyped to solve the software crisis; used singly, all have failed. The task has been too large and the problem too complex for any one silver bullet.

The interesting thing about silver bullets is that they are not available off the shelf (Gray, 2005):

Like darning socks, making bullets is a dying art. Used to be just about everyone with a need for ammo poured their own, using iron or even wooden molds. These days only a few diehard hobbyists still do it, and they use aluminum molds. But even fewer people still make silver bullets.

Actually, not many people ever made silver bullets ... At 1,764F, molten silver would ruin traditional and modern bullet molds. They could have been fashioned using jewelers’ methods, but that would require a new plaster mold for every bullet. Frankly, I think people spent a lot more time talking about silver bullets than they did turning them out.

DSM solutions toowill generally require building for your specific situation. There are sound economic reasons for this. Assume for a minute that you are lucky enough

Domain-Specific Modeling: Enabling Full Code Generation, Steven Kelly and Juha-Pekka Tolvanen Copyright # 2008 John Wiley & Sons, Inc.

408

NO SWEAT SHOPS—BUT NO FRITZ LANG’S METROPOLIS EITHER

that some kind vendor has already built a DSM tool that is perfect for your case. Unless your exact problem happens to be identical to one shared by a number of companies, the vendor will find itself with a customer base of one. You can be certain that the next version of their tool will generalize to be applicable to a wider market—and hence no longer be such a perfect fit for you. Even if you stick with the first version, you will find your own problem and solution domains evolve, again spoiling the fit. These problems will be familiar to anybody who has bought clothes: “one-size-fits-all” doesn’t, and many clothes that fit 10 years ago no longer seem so fitting in either style or size.

16.1 NO SWEAT SHOPS—BUT NO FRITZ LANG’S METROPOLIS EITHER

When the authors wanted to get some shirts with our company logo, there were of course no shops that already carried such products. Paying seamstresses to embroider each shirt by hand would have been one approach, and indeed the common one a few decades ago. Looking at software written by a given team today, there are a number of similarities with that approach. Building software is labor intensive and uses generic tools. We may have progressed from the needle and thread of Assembler to the sewing machines of 3GLs and IDEs, but the tools are largely similar across all developers. Despite the similar tools, the code written across all projects will show a huge amount of variation. However, the code written in this one project should recognizably belong together. The company coding standards, component libraries, shared architecture, and in-house style guide should lead to the work of each developer all bearing the “mark” of this company and team. If it does not, the situation would be similar to each seamstress embroidering a different version of the company logo.

This shared set of properties, or commonality, is what DSM can leverage to improve productivity. Most organizations will already devote efforts to codifying and sharing this information, and some may automate parts of the process. Further automation has generally been prevented by lack of time or insufficient payback. Similar problems faced the embroiderers: they could have built a machine that would just embroider our logo, but we needed our order by the next week, and the cost of building the machine would be higher than the size of the order.

Fortunately, the shop we went to had a configurable embroidery machine, into which the expert embroiderer could specify our logo, and which was then able to produce that logo automatically and reliably. Entering the logo into the machine required a new set of skills and a start-up cost. Once entered, anyone in the shop could run the machine to sew the logo on different material, in different colors, rotated, scaled, stretched, or duplicated in an endless pattern. These differences are analogous to the variability between different pieces of code from the same team, although clearly in software there is far morevariation per piece of code in the same project than there is per sewing of the same logo in embroidery. The basic situation was however the same: for all the stakeholders, this configurable automation approach represented a clear saving over the other choices.

CONCLUSION

Clearly, the expert embroiderer who could specify a new logo to the machine had an important role. But what of the other workers who would use the machine then and on subsequent occasions to embroider the logo onto shirts, caps, and so on? Have they become de-skilled like the workers in Fritz Lang’s Metropolis, little more than automatons pushing the buttons and pulling the levers of a machine? Far from it! Their skills are still very much in demand, working with the customer to choose the right material, color, size, and placement of the logo. If people with no aesthetic skills were let loose with the machine, all we would have would be the ability to produce terriblelooking clothes very quickly.

The important role of design is thus still present: only the tedious, time-consuming, and error-prone task of turning that design into finished product is automated. The same is true for the modelers in DSM: if we let people with no abstract thinking or analytical skills loose with a DSM tool, the code produced may be notionally bug-free thanks to the modeling language rules and the generator, but the resulting system will be unlikely to do anything useful.

The idea of DSM is thus not to replace developers with generators, but to enable those same developers to produce the same systems faster, more reliably and with higher quality. That after all is the problem of the software crisis: our projects are late (or fail) and contain too many bugs. DSM brings together a number of techniques to address that problem in a way that has proved repeatable across a wide variety of problem and solution domains.

16.2 THE ONWARD MARCH OF DSM

Unlike recent technologies such as Java or UML, DSM is not something that a vendor can market and push through to adoption with their tools. With those technologies, a gatekeeper in each organization had to be persuaded to adopt, and the job was done. With DSM, pure persuasion is not enough: the organization also needs to find someone and give them the task and resources to create a new modeling language and generator.

For many readers, you will be in the role of that gatekeeper or DSM solution creator. You are perhaps sufficiently convinced by the logic behind DSM to accept that it could work, but unsure if it will work for you. We hope that the material in this book will go some way to helping you feel confident enough to take a stab at trying out DSM in your organization. The gains are worth it.

For the first time in decades, our industry is presented with a way of development that has been consistently shown to increase productivity several times above the current norm. The people who can make the change happen are not the tool vendors, nor the managers, nor the mass of developers, although all these have a role to play. Instead, the choice is with the people who are best placed to make it, the expert developers. The task ahead of you is challenging but worthwhile.

Of course, as a busy lead developer you don’t have time for it. But every time you explain something to another developer, or correct their bug, or remind them to look at the in-house style guide—or even just realize why you don’t have time—remember that there is a better way.