Industrial Experiences on Using DSLs in Embedded Software Development

Juha-Pekka Tolvanen
MetaCase
Ylistönmäentie 31
FI-40500 Jyväskylä, Finland
jpt@metacase.com

1 Introduction

Domain-Specific Modeling (DSM) improves current software development approaches in two ways. First, it raises the level of abstraction beyond programming by specifying the solution in a language that directly uses concepts and rules from a specific problem domain — a Domain-Specific Language (DSL). Second, it generates fully functional production code from these high-level specifications [1].

While companies using DSM have reported productivity increase in various domains (automotive, signal processing, telecom [7]) the figures are often reported without details on how the measurement was performed, which development phases were covered, and what the results were compared with. We go into these details by examining industrial experiences in two different domains: touch screen devices and medical sports computers. These two cases are selected because they are relatively easy to understand from a short description and because there is public material available [2][4]. In both cases the companies have developed their own languages and code generators in-house — with little or no external consultancy help. The companies report productivity increase of 400–1000%.

We investigate also the effort the companies put in to build the DSLs and their generators. The ROI calculation on the two cases shows that the investment pays back very quickly.

2 Touch screen controller

Home automation is one product line at Panasonic and we focus here on home control. The Lifinity Control Panel is a touch screen device that is used to control lights, heating, air-conditioning, and burglar and other alarms. The device also reports on the consumption of energy and may include advanced features like earthquake alerts [4].

Panasonic defined a DSM solution to make their application framework more easily available to developers, enforce its correct use, and remove error-prone manual work. When studying the use of the framework, it became apparent that out of a total of 17 patterns of use, 5 patterns accounted for 60%–80% of user interface applications [4]. The language definition was then restricted to cover those.

One person at Panasonic defined the DSL along with generators that transformed models made with the DSL into the artifacts the company needed (e.g. code, configuration files, links to simulators, and document generation). The modeling language was supported by a tool [3] that provided the functionality needed to work effectively with models, such as reusing models, refactoring and replacing model elements, organizing and handling large models and multi-user model editing. Figure 1 illustrates a typical modeling situation showing the modeling language and supporting tool.

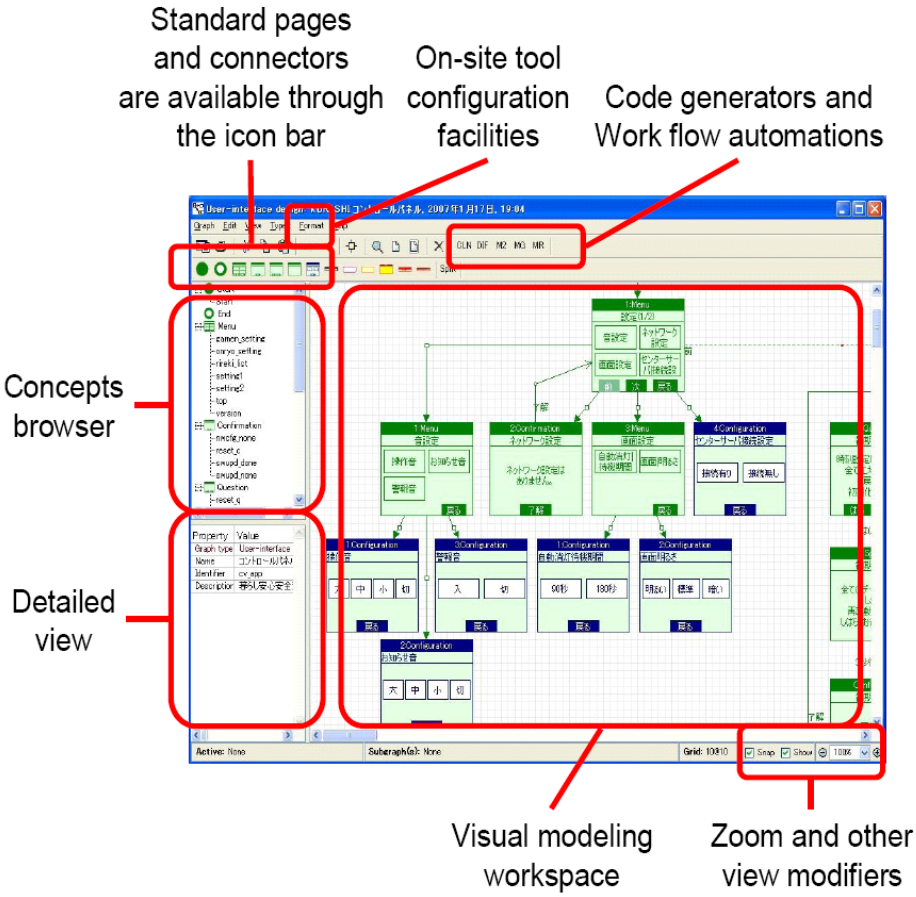

Figure 1. Sample of UI application design [4]

Panasonic compared the use of the DSM solution with the current manual specification, implementation and configuration approach by running a pilot project implementing a large portion of the typical home controller. In the pilot the specification and design was carried out by using the modeling language (Fig. 1) to create models, and the generator to produce the implementation code. Development effort was measured and compared to known practice. This showed a productivity increase of 300–500%.

3 Sports computer

Polar is the leading brand in the sports instruments and heart rate monitoring category. These products measure, analyze and visualize data on measures like heart rate, calories, speed, distance, altitude changes, pedaling rate and cycling power. They also offer various visualization capabilities, logbooks and exercise diaries. The features depend on the product segment and the type of sports the product is designed for, such as cycling, fitness and team sports.

Polar created an in-house DSM solution for the UI application side of the sports computer [2]. This domain was selected because it forms the single largest piece of software, typically requiring 40–50% of the development time. Improvements to UI application development would therefore have the greatest impact on overall development times.

Figure 2 illustrates the use of the language; it is about the simplest possible model. In real cases there may be dozens of elements in a diagram, dozens of diagrams in an application, and dozens of applications in a full product. An element in one diagram can be linked, referred to and reused in

other diagrams, or can be linked to a sub-diagram specifying it in more detail. The diagram is also executable, in that full code can be automatically generated from it.

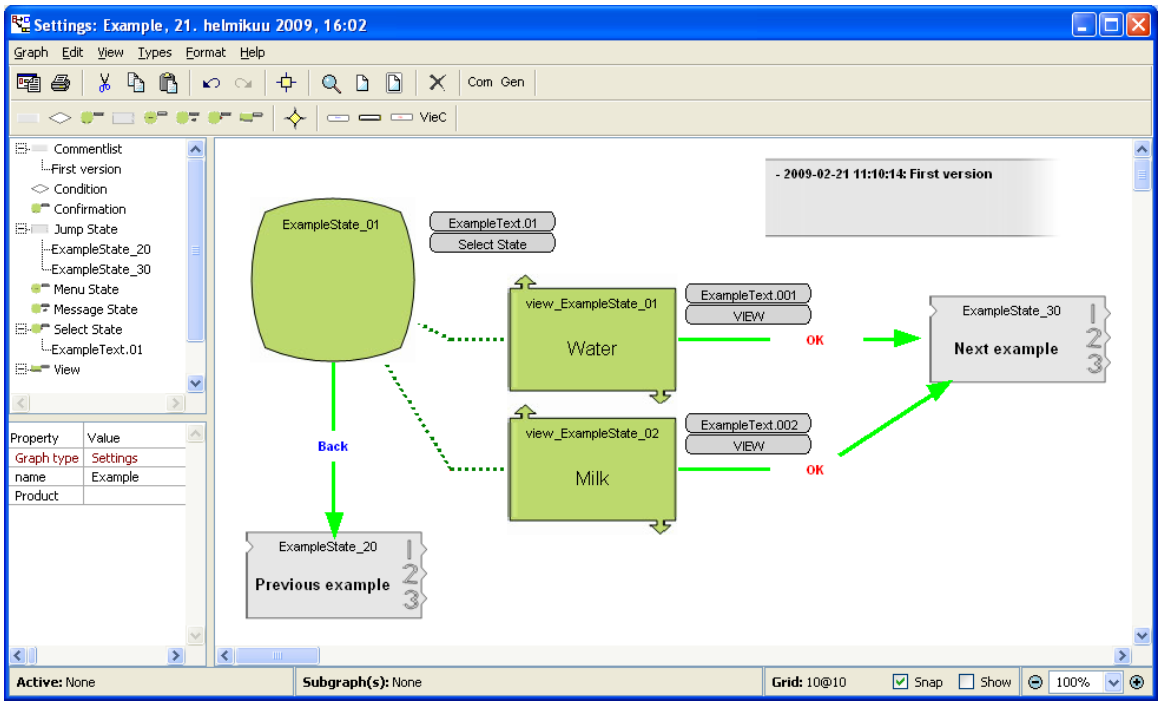

Figure 2. Model of an example UI application [2]

Polar evaluated the influence of DSM in two different ways: 1) by building a large portion of a product in a pilot project and 2) by asking 6 developers to individually implement a small but typical feature. In both cases the development time was measured and participants' experiences were collected. The starting point for DSM use was a UI requirements specification, the same as was used in their ordinary development process. The evaluation did not cover other development phases, such as testing, localization, documentation and providing user manuals.

In the pilot project one engineer developed UI applications whose implementation with their ordinary development approach was calculated to have taken 23 days. With the DSM approach the development took 2.3 days: a 10-fold improvement in productivity. The controlled experiment among 6 developers confirmed this increase in productivity on a smaller task, whose implementation with their ordinary development approach would take about 960 minutes (16 hours). With the DSM approach the development time varied from 75 minutes to 125 minutes, with a mean of 105 minutes. The productivity improvement for the mean time is thus over 900%. A questionnaire of developers' opinions showed they all found DSM to be significantly faster than current practice.

4 Return on investment

The benefits of DSM do not come for free: the company must first invest in implementing the DSLs and generators. Depending on the tooling used, time may also need to be allocated for tool creation and maintenance. In both companies the languages and generators were implemented using MetaEdit+ Workbench [3]. MetaEdit+ automatically provides modeling tools based on the modeling language, so no extra time needed to be spent on tool building. In the following we thus focus only on the effort to define the languages and generators.

4.1 DSM for touch screen devices

Panasonic wanted to compare two things: productivity and the possibility to target new platforms. First the productivity was compared by implementing a large portion of touch screen applications. This enabled comparison to the current practice. Figure 3 shows the comparison results. The black rectangles show the effort in man-days for a single product: 17 man-days with their ordinary coding approach and 4 man-days with DSM. The gray rectangle shows the investment in creating the DSM solution: 15 man-days [4]. A simple calculation shows that Panasonic's investment in DSL pays back already during development of the second product.

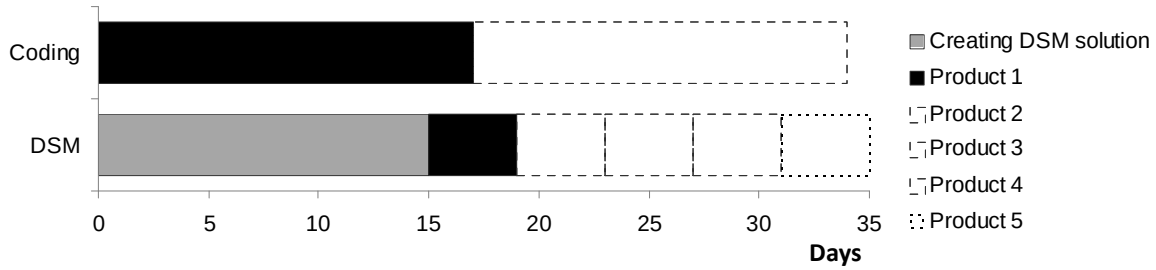

Figure 3. ROI on touch screen development: comparison

In the second phase an additional investment was made to target another platform: a remote controller instead of the touch screen device. Here only the generator was implemented since the same modeling language – and thus the same design models – was used. The generator for the remote controller was implemented in 3 days. As no modeling work was needed the application development costs for the new target were almost zero.

4.2 DSM for sports computers

At Polar, creation of the DSM solution – the modeling language and code generator – took 7.5 working days [2]. The 7.5 days also included the creation of example models specifying UI applications, along with corresponding hand-written code to help build and test the generator. Applying the DSM solution on real examples during its development proved a good approach.

When comparing the time to implement the DSM solution to the productivity improvements it brings, it is evident that the investment pays back very quickly (Fig 4). Based on the experiences from the pilot, implementing a whole product with DSM would take 2.3 days whereas the effort needed with their ordinary approach was 23 days. The data collected shows that even when including the 7.5 days of language and generator creation, the first product was produced over twice as quickly with the DSM approach: 9.8 days rather than 23 days.

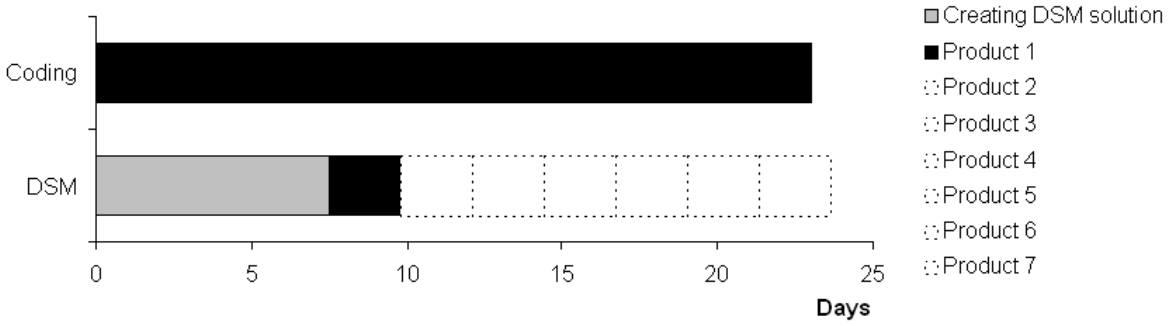

Figure 4. ROI on sports computers development: comparison

5 Concluding remarks

DSLs and generators are claimed to significantly improve the productivity of software development. We investigated two cases in which companies have reported their experiences in detail. Both the cases showed that productivity increased fundamentally, when the modeling language used concepts from the problem domain, at a higher level of abstraction than the code. This shows a drastic difference to experiences on modeling languages (e.g. UML) that operate more on the code level, specifying classes, inheritance etc.

The studies could be extended to cover a larger portion of the development cycle. In addition to serving application implementation, generators could also be created to support other roles and processes: to provide input for testing, generate review reports and documentation. We can expect that future modeling with the same DSM solutions would have worked even better, as developers gained more experience with their use, and as existing models from the initial products could be reused in later products.

References

[1] Kelly, S., Tolvanen, J-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008, http://dsmbook.com

[2] Kärnä et al., Evaluating the use of DSM in embedded UI application development, Procs of DSM'09 at OOPSLA, 2009. http://www.dsmforum.org/events/DSM09/Papers/Karna.pdf

[3] MetaCase, MetaEdit+ Workbench 4.5 SR1 User's Guide, http://www.metacase.com/support/45/manuals/, 2008

[4] Safa, L., Making Of User-Interface Designer A Proprietary DSM Tool, Procs of DSM'07 at OOPSLA, 2007. http://www.dsmforum.org/events/DSM07/papers/safa.pdf

[5] Sprinkle et al., What Kinds of Nails Need a Domain-Specific Hammer? IEEE Software, 4, 2009.