Visual domain-specific modelling: Benefits and experiences of using metaCASE tools

Steven Kelly (stevek@metacase.com) & Juha-Pekka Tolvanen (jpt@jyu.fi)
MetaCase Consulting & University of Jyväskylä

1 Introduction

Jackson (Jackson 95) recognises the vital difference between an application's domain and its code: two different worlds, each with its own language, experts, ways of thinking etc. A finished application forms the intersection between these worlds. The difficult job of the software engineer is to build a bridge between these worlds, at the same time as solving problems in both worlds.

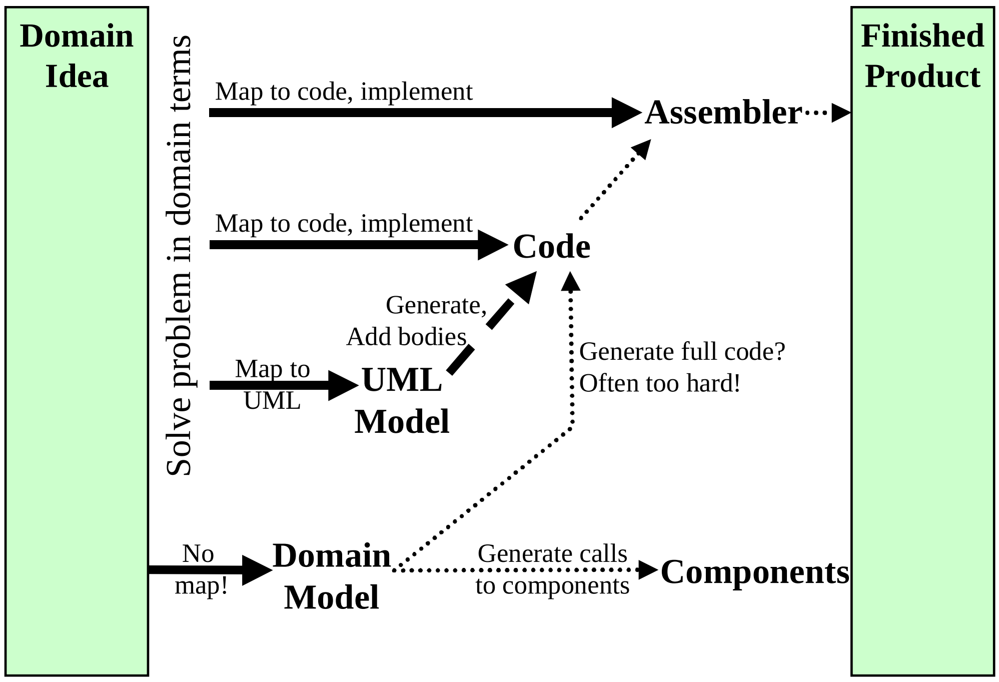

Figure 1. Moving from domain idea to finished product

Figure 1 shows four different ways in which this bridge-building has occurred: how developers have moved from an initial domain idea to a finished product. In the first two ways, the problem had to be solved in domain terms first, then this solution was mapped to the world of the implementation platform, and implemented there. The invention of more powerful chips, assemblers, and compilers steadily narrowed the domain-product gap from the right inwards, allowing the final 'hand-made' artefacts of the designer to be at a higher and higher level of abstraction. These artefacts could then automatically be transformed into the finished product.

The introduction of modelling languages such as UML changes surprisingly little: the problem must still be solved first in domain terms, with little or no tool support. This solution must then be mapped to the core UML models representing the implementation in code, from which in general a relatively small percentage of the finished code can be automatically generated. The developer must then fill in the method bodies by hand: the largest part of the implementation. Thus the developer still has to solve the problem twice: once in domain terms (often in his head and on the backs of envelopes), and once in code terms. He still has to perform the mapping from the domain solution to the code solution, now with an extra ‘stepping stone’ of UML — planted firmly near the code side. In fact, in cases where UML models do not provide adequate mappings to code, the developer must now solve the problem three times!

Primarily from industry, there have been moves towards a way of building software that removes this resource-intensive and error-prone mapping and duplicated (or even triplicated) problem solving. The dream is that a developer would be able to develop the solution once only, as a model in domain terms, from which the finished product could be automatically generated. Such an approach has already been seen to work dramatically effectively in a limited range of situations (notably embedded systems based on state machines).

However, the gap between a model in domain terms and the necessary finished product code has generally been too wide for automatic generation. Similarly, the work involved in creating the infrastructure necessary to enable domain-specific modelling and generation has been significant, reducing the number of situations in which this approach is applicable.

In this paper we look at how the benefits of this domain modelling approach can be achieved for a far wider range of applications, by bringing together results from several research areas. In particular we find that metaCASE technology significantly reduces the workload in the hardest parts of the work. As a demonstration of the possibilities of this approach, we look at how Nokia uses the MetaEdit+ metaCASE tool to build its mobile phones, and the benefits they gained from it.

2 Converging research areas

All of the research areas below are large; each has its own substantial body of literature and experience. In an article like this, we can do no more than scratch the surface of each.

2.1 Methods and CASE

Empirical studies (Necco 87, Fitzgerald 95) have consistently shown that only around half of all development projects use methods. Among those using methods, over 50% either modify the methods to better fit to their need or even develop their own methods (Russo 95, Hardy 95).

In a standard CASE tool, the method supported by the tool is fixed: it cannot be changed. In a metaCASE tool, there is complete freedom to change the method, or even develop an entirely new method. Both models and metamodels (method descriptions) are stored as first-class elements in the repository. This allows an organisation to develop a method that suits their situation and needs, and to store and disseminate that knowledge to all developers. The tool and method then guide developers, provide a common framework for them to work in, and integrate the work of the whole team.

Research prototypes and even commercial metaCASE tools have existed for many years, but only recently have there been tools which are mature, user-friendly and stable for both the method developer and the method user. One of the most widely known and used metaCASE tools (Isazadeh97, Alderson 99) is MetaEdit+ (Kelly 1996), which was used in the case study below.

2.2 Visual programming languages

A large body of research (see e.g. Burnett 99) exists for various visual languages that either map to existing textual languages, or else allow execution together with their own interpreters or compilers. Space does not allow further elaboration here: suffice it to say that there are many proven and useful ways of using visual models to represent executable programs, for a wide variety of applications.

2.3 Domain-specific modelling

In a domain-specific method, the models are made up of elements representing things that are part of the domain world, not the code world. In spite of — and in parallel with — the trend towards standardised code-world methods such as UML, a global survey (Seppänen 96) found that the direction in many companies is away from 'universal' methods and towards domain-specific methods implemented with metaCASE technology.

A particularly fertile area for domain-specific modelling is families of products, where there is a large amount of code shared between several products. Even using simple text-based modelling tools, experiences at Lucent (Weiss 99) show that domain-based modelling for product line configuration can be economically viable if there are more than 2-4 product lines (or variants).

2.4 Code generation

Standard CASE tools and methods can generate a small percentage of operational code, and a large percentage of data description code. Real gains — (near-) 100% generation, are generally based on some form of state charts (Harel 87). Harel's early work on these progressed to code generation in the Statemate and Rhapsody (Harel 97) tools. Other products such as ObjectTime (Selic 94) (now Rose/RT) claim even better results, backed up by independent reports (IDC 99).

The main effort in modern research on such state-based generation is towards code that performs well in terms of time and memory usage (e.g. Burst 98). Such a trend is necessary in embedded software, but less so for other kinds of software.

2.5 Components

A natural progression from earlier research in modularization, abstract data types, object orientation and reuse, components form one of the 'hot topics' of today. By enabling programmers to work at a higher level of abstraction, and by leveraging previous successful programming effectively, components promise much. The main cloud on the horizon is how to help the developer find components, indeed even know that there might be one that does something he wants.

3 How can metaCASE bring these together?

To get to a situation of domain modelling followed by full automatic code generation, we must provide three things: a modelling tool with a domain-specific method, a code generator, and a domain-specific component library. For any realistic chance of being able to achieve these, we need an expert: a good developer who has already developed several products in this domain, developed the architecture behind the product, or has been responsible for forming the component library for the product. Figure 2 shows these three elements that must be made by the expert, along with how they will be used by the normal user.

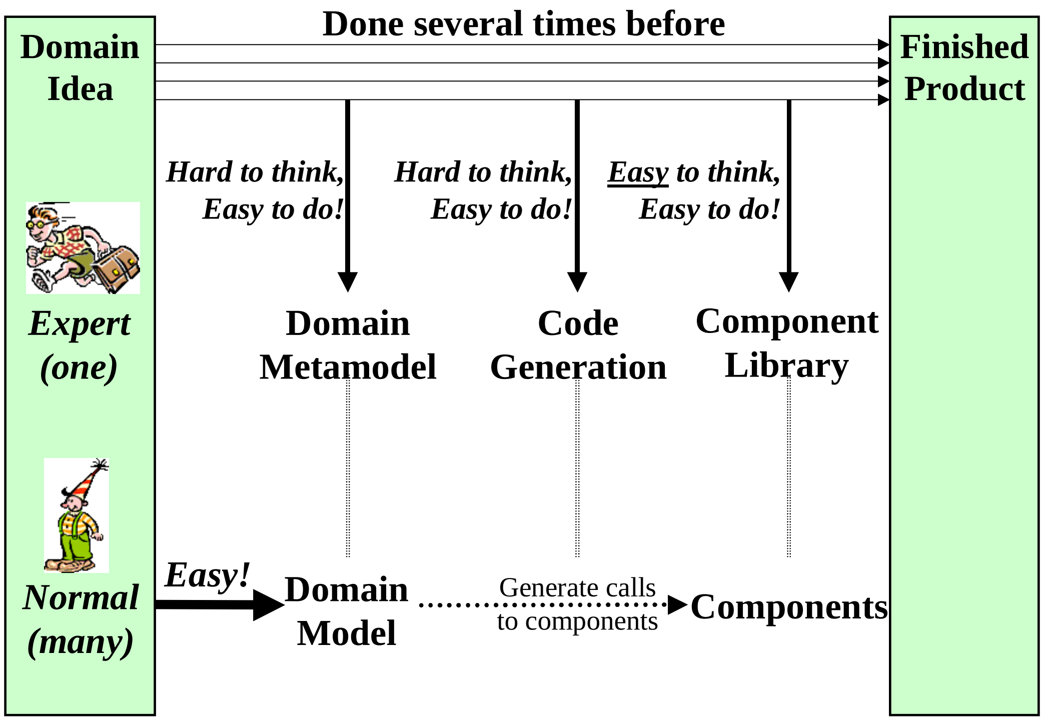
Done several times before
Domain Idea
Finished Product
Hard to think, Easy to do!
Hard to think, Easy to do!
Easy to think, Easy to do!
Expert (one)
Domain Metamodel
Code Generation
Component Library
Normal (many)
Easy!
Domain Model
Generate calls to components
Components

Figure 2. Leveraging experts to enable others

3.1 Assembling the component library

Starting on the right, the component library is not always necessary, but it makes the task of code generation development significantly easier. Often, code components already exist from earlier development cycles, at least in the form of reusable pieces of code. Further developing these pieces of code into true components is a relatively easy task for the expert, requiring only the normal developer programming tools.

In addition to components developed in-house, the library can of course contain third-party components. Whilst the component library here is thus nothing new, the fact that it will be intrinsically part of the development process ensures that the components there will actually be used.

3.2 Developing the domain metamodel

A harder task is the development of the domain metamodel; the concepts and notation the end user will build his model with. The experience and intuition of the expert, combined with hints from the component library, domain rules and architects are the real sources of clues. The expert should also consider the issue of code generation, which will be his next task: a good mapping from the domain metamodel to combinations of components will make that much easier. Metamodelling languages can be applied here to describe both the domain rules and their mappings (e.g. Hillegersberg et al. 1998). Use of metamodelling languages, such as GOPRR (Kelly et al. 1996), is reported by several successful metamodelling efforts (cf. Tolvanen and Lyytinen 1993, Rossi and Brinkkemper 1996, Hillegersberg et al. 1998, Merunka and Polak 1999).

In the creation of a domain metamodel, the support offered by the metaCASE tool is vital. A metaCASE tool that allows rapid prototyping is practically a necessity: the expert can then make a part of the metamodel as a prototype, and instantly test it by making an example model. Similarly, the more the metaCASE tool can automatically make of the finished CASE environment, the better: the expert can then concentrate on the hard problem of developing the method, and the CASE tool implementation of it, with all its menus, toolbars, icons and behaviour, is instantly and automatically available.

With a good metaCASE environment, then, the main work is in establishing the content and structure of the method: the metaCASE environment makes creation of tool support for the method a simple task, of the order of a day or two.

3.3 Developing the code generator

The code generation definition forms the final task, conceptually if not chronologically: in practice there will be a large degree of parallelism and incrementality between all three tasks. In this phase the expert specifies how code using the components can be automatically generated from the structures in the users' models. The metaCASE tool should provide the necessary functionality for creating such generation scripts, and should guide the expert where possible by allowing him to reference and use the concepts in the metamodel.

Of all the phases, code generation probably varies the most between domains. In some domains it will be possible to produce a large fraction of code with a relatively simple code generation scripting language, such as is already provided in metaCASE tools. In other domains, it may be necessary to use a more powerful language to operate on data exported from the CASE tool. The most important goal is that the end user should be able to use the code generation simply: the cost of developing the code generation is then defrayed over many users.

4 Case study: Nokia Mobile Phones

As with any case study where significant benefits have achieved, reporting them in an academic way faces two difficulties: the danger of sounding triumphalist or overly commercial when providing evidence from case study participants, and the problem of providing enough detail without revealing trade secrets. Hopefully the reader still finds this section useful, and can overlook the areas where we have fallen foul of these difficulties.

4.1 Application domain

Mobile phones and mobile computing is one of the fastest growing markets today. There are roughly 500 million mobile phone users world-wide, and the 1 billion mark is expected to be broken in 2002 (Nokia 1999) (compared to an estimate of 2005 one year earlier). New innovations and phone features, such as Internet capabilities and WAP, are changing the way people communicate and make business. At the same time the software inside the mobile phone has become the most critical part from the consumer's point of view: it determines the winners.

For the manufacturers, time to market and quality are critical: innovative features and problem-free operation guarantee the publicity and good reviews, which lead to top sales figures. In this environment, where releasing a product a month before a competitor translates to millions of dollars, the productivity of the development process and tools is vital.

4.2 Objectives for development tools

To compete more effectively, NMP was looking for development tools that would improve the productivity of development teams by an order of magnitude. These improvements were to be achieved by applying the following strategies:

- Working at higher abstraction levels so that designers don't have to know everything. They can focus on designs rather than on how to implement it in code.
- Encapsulate domain knowledge so that only the relevant part of mobile phone software is captured. This lets development teams focus on the characteristics of the needed functionality. The domain knowledge contained in the method also makes for an easier, faster learning curve for new employees or transferred personnel.
- Link designs to code generators so that designers are effectively 'writing code' as they design (but they don't realise it immediately!).
- Underpin the development process with a tool effective enough that no one will want to develop outside the tool.

4.3 The tool search

In the beginning, a team in NMP's Advanced Development Group evaluated prominent CASE tools. The result of the tool evaluation was a disappointment. The tools examined were found to be inflexible in terms of method extensions, code generators and process support. Most available tools support published methods that were not domain-oriented. They allow the description of almost any application, but only by often clumsy mappings from the application domain to the tool's own concepts, losing vital understanding of the domain in the process. "UML and other methods say nothing about mobile phones. We were looking for more", said David Narraway, Project Manager at NMP. Further, existing methods assumed 'clean-sheet' implementations, whereas the majority of NMP projects built on existing previous work.

Most importantly, NMP realised that there was a need for tools that fitted the domain, rather than for tools which required the domain and organisation to change. A team in the Advanced Development Group of NMP had already developed a phone architecture, components and code generators to partly automate creation of mobile phone user-interfaces. A graphical design tool was needed to obtain the full advantage from these other tools. Hence, they decided to undertake the development of their own CASE solution.

Quite early on, NMP chose a metaCASE approach. There was a strong need for method flexibility — as the domain evolves, so should the method — and for applying the new development objectives quickly. Among a number of metaCASE tools evaluated, MetaEdit+ was selected: "It was the most flexible, allowed us to define our own design syntax quickly, and test ideas quickly while developing the method," summarised David Narraway. Report generation capabilities and links to code generation were also highly valued.

4.4 Application of metaCASE technology

As in many other development projects, existing previous work was available in the form of software components. At NMP, previous research collaboration had also produced an in-house tool prototype, which would later form part of the code generation.

The main work of developing the method was carried out by Software Technologist Jyrki Okkonen. Initial prototypes of the method were quickly achieved and tested, revealing further possibilities for improvement. From an early stage, actual users began modelling in the method, even as it was still being developed.

The code generation solution adopted in this case was to produce part of the code directly from MetaEdit+, and part by exporting from MetaEdit+ to the code generator mentioned above, which then generated the rest of the code.

4.5 Results

By developing and implementing their own domain-specific method with metaCASE technology Nokia has achieved what they were seeking. Indeed, even to this day the flexibility of the metaCASE approach is being used regularly to improve the method, now in use in ten sites by over 100 developers. In many ways this success was achieved through the benefits of the approach per se, rather than any particularly magnificent implementation in MetaEdit+. Of the features of this case that are particular to MetaEdit+, incremental method development and automatic model update were perhaps the most important. Below are some of the main benefits, with quotes from project leader David Narraway.

- Order of magnitude productivity gains: "A module that was expected to take 2 weeks even with the new tool now took 1 day from the start of the design to the finished product"
- Domain-oriented method allowed developers to concentrate on the required functionality, and shifted the focus from code to design.
- Results from code generation more than fulfilled expectations. "In many cases we can generate 100% of the required code. This is a result of innovative method development: domain-oriented metamodels provide the ideal way to link designs to code and to software components."
- Documentation was improved. Documentation reports implemented in MetaEdit+ followed NMP's standards, and are used for review meetings as well as product description. Automated document generation naturally saved time and improved consistency and standard compliance.
- Training time was reduced significantly. Because the method fitted the domain, the developers found it easy to adopt: it already included the concepts and vocabulary they were familiar with. Similarly new employees only needed training in the domain, not coding: "Earlier it took 6 months for a new worker to become productive. Now it takes 2 weeks"

5 Conclusion

Domain-specific visual modelling languages represent a powerful way to leverage the abilities of expert developers to enable other developers in a team. Previously largely only applicable to a small area, recent advances in components and metaCASE technology significantly increase the range of applicability of this approach. In particular, a modern metaCASE environment can markedly reduce the time and effort required to develop a domain-specific method, along with its tool support and code generation.

Industrial application of this approach in a case study shows remarkable improvements in productivity and training time: of the order of a factor of ten times faster, in both cases. The benefits to the industry of such an increase in productivity are clear. Even clearer are the commercial benefits to be gained in areas of fast technological development and short product lifespan, in terms of reduced time to market. In an era of rapid change and fast employee turnover, some of the most important improvements are related to training costs and introduction of new team members.

6 References

Burst, A., Spitzer, B., Wolff, M., Müller–Glaser, K.D., On Code Generation for Rapid Prototyping Using CDIF, *OOPSLA'98 Workshop on Model Engineering, Methods and Tools Integration with CDIF, 1998.*

Alderson, A., Cartmell, J.W., Elliott, A., *ToolBuilder: From CASE Tool Components to Method Engineering,* Peer Logic white paper, 1999.

Bézivin, J., Who's Afraid of Ontologies?, OOPSLA'98 Workshop on Model Engineering, Methods and Tools Integration with CDIF, 1998.

Burnett, M., Visual Programming Bibliography, www.cs.orst.edu/~burnett/vpl.html, 1999.

EIA/CDIF Technical Committee: *CDIF / CASE Data Interchange Format*. EIA Interim Std. EIA/IS– 106–112, 1994.

Fitzgerald, B., *The use of system development methods: a survey*. Paper ref 9/95, Univ. College Cork, 1995.

Hardy, C., Thompson, J., Edwards, H., The use, limitations and customisation of structured systems development methods in the UK. *Information and Software Technology*, 37 (9), 1995.

Harel, D., Statecharts: *A Visual Formalism for Complex Systems*. Science of Computer Programming, 1987.

Harel, D., Gery, E., Executable Object Modeling with Statecharts. *IEEE Computer*, July, 1997.

Hillegersberg van, J., Kumar, K., Welke, R.J.,: Using metamodeling to analyze the fit of object-oriented methods to languages. *Proceedings of the 31st Hawaii International Conference on System Sciences,* Volume V, (eds. R. Blanning, D. King) IEEE Computer Society, 1998.

IDC, *The Rose Family Grows*, IDC report #20089, 1999.

Isazadeh, H., Lamb, D.A., CASE Environments and MetaCASE Tools, Technical Report 1997-403, Queen's University, Canada, February 1997.

Kelly, S., Lyytinen, K., Rossi, M., *MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment,* Advanced Information Systems Engineering, proceedings of the 8th International Conference CAISE'96, Constapoulos et al (Ed.), Springer-Verlag, 1996

Jackson, M.A., *Software requirement & Specifications* A lexicon of practice, principles and prejudices Addison Wesley, ACM Press, 1995.

Merunka, V., Polak, J., BORM, http://omega.pef.czu.cz/pef/kii/oo/orm/borm_html/

Necco, C.R., Gordon, C.L., Tsai, N.W. Systems Analysis and Design: Current Practices, *MIS Quarterly,* December, 1987.

Nokia Mobile Phones, Press Release: *Nokia expects increased mobile growth and raises subscriber estimates*, Dec. 3 1999.

Rossi, M., Brinkkemper, S., Complexity Metrics For Systems-Development Methods And Techniques, *Information Systems*, 21, 2, 1996.

Russo, N., Wynekoop, J., Walz, D., The use and adaptation of system development methodologies. *Procs of International Conference of IRMA*, Atlanta, May 21-14, 1995.

Selic, B., G. Gullekson and P. T. Ward, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, New York, 1994.

Seppänen, V., Kähkönen, A. -M., Oivo, M., Perunka, H., Isomursu, P., Pulli, P., (1996) *Strategic Needs and Future Trends of Embedded Software*. Technology Development Centre, Technology review 48/96, Sipoo, Finland.

Tolvanen, J.-P., Lyytinen, K., Flexible method adaptation in CASE - the metamodeling approach. *Scandinavian Journal of Information Systems*, Vol. 5, 1993.

Weiss, D., Lai, C. T. R., *Software Product-line Engineering*, Addison Wesley Longman (a Pearson Education company), 1999.