Abstract

We would like to understand the interests of our target audience. Please register at **www.softmetaware.com/whitepapers.html** to provide us with some information about yourself, and to obtain access to the full content of all SoftMetaWare white papers.

Process Implications of Model-Driven Software Development

Author: Jorn Bettin

Version 1.0

September 2004

SoftMetaWare

Revision History

Version	Author	Description
1.0	Jorn Bettin	Translation of the corresponding German article in OBJEKTspektrum, Sept. 2004

REVISION HISTORY 2

1 FRAMEWORKS 3

2 DEVELOPMENT OF DOMAIN-SPECIFIC FRAMEWORKS 3

3 WHY GENERATION, IF FRAMEWORKS DO MOST OF THE WORK? 3

4 DOMAIN ENGINEERING 4

5 LANGUAGE DESIGN 5

6 PLATFORM AND INFRASTRUCTURE DEVELOPMENT 5

7 MODEL-DRIVEN APPLICATIONS 6

8 APPLICATION ENGINEERING 6

9 CODING OF BUSINESS LOGIC 8

10 COMPILING AND BUILD 8

11 CONCLUSIONS 8

12 REFERENCES 9

On the surface, Model-Driven Software Development (MDSD) appears to be a fundamentally new paradigm compared to traditional software development. Upon closer examination however, MDSD turns out to mainly shift the focus of iterative development to a higher level of abstraction. Just as with the transition from assemblers to compilers, the main barrier to acceptance is not technological but mainly cultural. Hence, it should not come as a surprise that model-driven techniques do not gain broad acceptance over night.

Changes to software development processes need to be introduced carefully, since all participants in the software development process are affected, especially if the changes are not simply about learning new APIs or the syntax of a new programming language. This article explains the main differences between model-driven development and traditional iterative software development.

1 Frameworks

Because frameworks play a critical role in MDSD, it is a good idea to investigate the framework usage and framework development. The lesson of *small is beautiful* also applies to frameworks in the model-driven context. The MDSD paradigm starts with manual development of a reference implementation that covers all architectural layers and that can be held very narrow in terms of coverage of application functionality. The reference implementation should already include the binding to appropriate commercial or Open Source frameworks. The next step consists of abstracting framework integration code from the reference implementation, and capturing the results in the form of code templates. This step only increases the costs of developing a proof-of-concept prototype by about 15%-25%. Considering the amount of repetitive framework integration code contained in typical J2EE applications, it is easy to see that using a model-driven approach can save a lot of work.

2 Development of domain-specific frameworks

The term *domain* is typically used in conjunction with expert knowledge in specific industries, such as banking, manufacturing, etc. In MDSD the term domains is also used to refer to sub-domains of the software engineering domain. Especially if a software development team starts without deep industry-specific domain knowledge, the team should at least use all its technical knowledge to automate repetitive tasks in the software development process.

MDSD follows the agile mantra of avoiding speculation about the requirements of future applications, and every investment in framework development needs to be underpinned by the requirements of a concrete application. This approach prevents a scenario where a framework team proceeds to develop elaborate frameworks in the absence of concrete problems that need to be solved.

Nevertheless, in a larger project, it is recommended to create a dedicated framework to support one or more application development teams - not to shield framework developers from reality, but to simplify project management and release management. The objectives and priorities for the framework team need to be set by an architecture group, which consists of the team leaders/application architects of the application development teams. The framework team consists of technology specialists who develop reference implementations, write the integration code for external frameworks, derive code templates, and develop small frameworks as required. The results are validated by the architecture group, which acts as the customer of the framework team, and which therefore controls budget and priorities.

3 Why generation, if frameworks do most of the work?

Even in small frameworks the source code and additionally available framework documentation is often not sufficient for framework users to quickly gain the necessary insight into "correct" framework usage, i.e. usage as intended by the framework developer. The old *time is money* applies especially to software development projects.

Ensuring correct framework usage is one of the real strengths of MDSD and model-driven generation of framework completion code. After creating a small example