

Adopting model driven development in a large IT consultancy organization – opportunities, challenges and lessons learnt

Vinay Kulkarni and Sreedhar Reddy

Tata Research Development and Design Centre,
54, Industrial estate, Hadapsar,
Pune, 411 013, INDIA
{vinay.vkulkarni, sreedhar.reddy} @ tcs.com

1. Introduction

Back in 1994, we were developing a core banking product that was to be delivered on state-of-the-art technologies of the day viz., object oriented technology, three-tier client server architecture with OLTP monitor, rich GUI etc. Being a customizable product offering, it was expected to cater to a choice of RDBMSs, OLTP monitors and GUI platforms. Object orientation was still in the early stages of industrial adoption with a shortage of experienced OO programmers and a lack of proven industrial-scale OO development method backed by robust development tools. The business logic of a typical business application is low on algorithmic complexity. Implementation of three-tier architecture has a high recurrence of a set of design and architectural patterns. We wanted to exploit these facts to simplify and automate development. An architectural prototype exercise by a small team of expert architects, revealed several reusable design and architectural patterns that can be separated from business logic. This led us to design a simplified domain-specific OO language for specifying business logic and a set of models to capture the design and architectural patterns. We developed a toolset that takes these models and business logic specifications as input and generates the various layers of the three-tier architecture completely. Coordinated development with a large team was facilitated through a repository-centric development process. This model driven development approach supported by the toolset shifted the focus of software development from coding to modeling resulting in enhanced productivity, uniformly high code quality and platform retargetability. However, the approach incurred some initial costs as the modeling standards were still nascent and the developers needed to be trained on modeling. In spite of these initial learning costs, we pushed for organization-wide adoption of this approach as the benefits far outweighed the costs. Large development projects and enterprise product lines found it easier to adopt the approach as they could afford these initial costs. However, small to medium sized development projects, of which there is a large number in our organization, found it difficult to adopt to the approach. They appreciated the code generation capabilities but found the initial learning cost too prohibitive due to their shorter timeframes. We devised a code-centric metadata-driven aspect-oriented development approach to address their

needs. However, many a times a project grows into a large project and / or a product line over time. We extended the code-centric development approach with modeling so as to make available the benefits of model driven developments to such projects. It is now possible to begin with a code-centric development and move up to model-driven development when the need arises.

We begin with a description of our model-driven development approach. We discuss our experience of using model-based techniques for developing product-lines, several large enterprise applications and a few small-to-medium sized projects. We argue for the need of a lightweight and flexible approach to develop small-to-medium sized applications and present a metadata-driven aspect-oriented approach for the same. We introduce a hybrid approach that combines the benefits of the two. We conclude with an analysis of our experience.

2. Our model-driven development approach

The development of an application starts with an abstract specification that is to be transformed into a concrete implementation on a given target architecture [3]. The target architecture is usually layered with each layer representing one view of the system e.g. Graphical User Interface (GUI) layer, application logic layer and database layer. The modeling approach constructs the *Application specification* using different abstract views - *GUI layer model*, *App layer model* and *Db layer model* each defining a set of properties corresponding to the layer it models. Corresponding to these specifications are the three meta models - *GUI layer meta model*, *App layer meta model* and *Db layer meta model* which are views of a single *Unified meta model*. Having a single meta model allows us to specify integrity constraints to be satisfied by the instances of related model elements within and across different layers. This enables independent transformation of *GUI layer model*, *App layer model* and *DB layer model* into their corresponding implementations namely *GUI layer code*, *App layer code* and *Db layer code* with assurance of integration of these code fragments into a consistent whole. These transformations can be performed either manually or using code generators. The transformations are specified at meta model level and hence are applicable for all model instances. If each individual transformation implements the corresponding specification and its relationships with other specifications correctly then the resulting implementations will glue together giving a consistent implementation of the specification. Models can be kept independent of implementation technology and the application specifications can be targeted to multiple technology platforms through code generation. Construction of application specification in terms of independent models helps divide and conquer. Modeling helps in early detection of errors in application development cycle. Associated with every model are a set of rules and constraints that define validity of its instances. These rules and constraints could include rules for type checking and for consistency between specifications of different layers. Automated code generation results in higher productivity and uniformly high code quality.

3. Experience of adopting our MDD approach

This approach is realized in our model driven development environment [7]. Over the last 12 years, it has been used to deliver enterprise product lines, several large development projects and a few small to medium sized projects [5].

All the project teams initially found it hard to switch over from the conventional code-centric development approach to model-centric approach. The difficulties were essentially due to the steep learning curve and perceived loss of control over development artifacts. Some of the problems encountered were due to the state of the tool support available. In model-driven development approach, part of the specification is in model form and part in code form. However, debugging support was available only at the code level leading to difficulties in debugging. Also, the cycle-time required to introduce a small change in the model and test it was found to be significantly greater for the model-driven approach than traditional code-centric approach. However, the fact that a model-level change gets automatically reflected at multiple places in a consistent manner was appreciated.

Several projects had a product-family nature wherein a product-variant needed to be quickly put together and customized to meet the specific requirements of a customer. Model-driven development approach helped in quickly retargeting the application functionality on multiple technology platforms with purpose-specific choice of architecture and design strategies. This was achieved using a relatively unskilled workforce as the technology and architecture concerns were largely taken care of by the tools [8]. The tool-assisted component-based development process helped in early detection of errors that would otherwise have led to late-stage integration problems. Also, all the projects reported significant improvements in productivity and quality. Large development projects found the approach attractive due to the assurances of consistent application of chosen design strategies, guidelines and best practices across the application. Promise of productivity enhancements even with a relatively unskilled workforce made it worthwhile for product-lines and large projects to invest in learning/training and to cope with the tool irritants.

However, small-to-medium sized projects found the model-driven approach too heavyweight and restrictive. Shorter project durations didn't allow for investments in learning/training. Need to deliver quickly made the teams intolerant to tooling irritants. The perceived loss of control of the development process and artifacts was the principal reason for these projects not taking to model-driven approach.

4. Metadata-driven aspect-oriented development approach

Our metadata-driven aspect-oriented approach treats code itself as the model thus bringing the advantages of model-driven development to small and medium sized projects, and removing model-code dichotomy and associated problems. Implementation code of an enterprise application can be broadly classified into business logic and code for solution architecture that addresses non-functional concerns such as design strategies, architecture and technology platform. The central

idea is to annotate business logic, specified using a popular programming language such as Java, with *tags* where a tag encodes one of the possible choices of a non-functional concern so as to generate the code for solution architecture from its declarative specification.

We fall back on model-to-model transformation [9] and model-to-text transformation [8] techniques for code generation. The basic UML [10] class model can be easily extracted from the business logic specified using a popular programming language like Java. The code generator for the required solution architecture can be specified as a composition hierarchy of tags. We build on the ideas of separation of concerns [11] and aspect weaving [2] to develop the building block abstraction [6] that specifies contribution of a tag towards code generation. Building block is the unifying abstraction to specify a tag in terms of i) model corresponding to the specific concern, ii) transformation of the concern-specific model into UML class model, and iii) transformation of the resultant model to the desired code fragments. Ability to compose tags into a hierarchy leads to two types of building blocks, namely, *leaf building block* and *composite building block*. A leaf building block specifies the code fragment to be generated for a tag and a composite building block specifies how to compose the code fragments its child building blocks have generated.

The approach was supported through an open extensible Eclipse-based [1] toolset resulting in a development process that is flexible and lightweight as compared to model-driven development process. Separation of architect role (to specify tags) from developer role (to specify business logic and use tags) led to effective utilization of expertise. The reusable nature of building blocks enabled reuse even across projects. Template based code generation strategy enabled easy adherence to customer-specific standards, guidelines, best practices etc. The composable nature of building blocks, the ability to define purposespecific meta models and plug-in architecture due to Eclipse resulted in easy customizability, extensibility and improved maintainability of the toolset. The approach facilitated creation of a repository of reusable artifacts. Meta tools driven by a library of reusable, easy to adapt solution accelerators enabled even inexperienced teams to deliver high quality code on schedule. Low or no learning curve, adherence to industry standards and dependence on freeware further accelerated acceptance of the approach by the developer community.

5. Merging the two approaches

A *tag* essentially is a mechanism to augment the basic fixed model supported by the underlying programming language. In the modeling world, the information captured by a tag can be specified in terms of first-class constructs like *Objects*, *Properties* and *Associations*. Thus, by providing a bridge from a tag to its corresponding meta model one can avail the power of model driven development. We evolved an approach that merges the code-centric metadata-driven aspect-oriented approach with model driven development approach in a seamless manner enabling one to switch between the two as required.

This approach supports the following scenarios of software development:

- Models are created using standard modeling tools during analysis phase. Skeletal class definitions are generated from these models. The metadata-driven aspect-oriented development approach is used thereafter.
- Project starts with metadata-driven aspect-oriented approach and moves up to model-driven approach when the scope and size of the project demands it.
- An iterative development approach wherein analysis and design phases use model-driven approach while coding and testing phases use metadata-driven aspect-oriented approach.
- Model-driven development approach is used during development stage and metadata-driven aspect-oriented approach is used for emergency maintenance in the production stage.

The focus of model driven development community so far has been on standardizing modeling notations and transformations in order to facilitate tool interoperability. From our experience, we feel, the focus now needs to shift towards evolving suitable development methods that allow model driven development to be adopted in a more flexible and incremental manner, rather than an all-or-nothing top-down manner.

References

- [1] Eclipse – a universal tool platform. <http://www.eclipse.org/>
- [2] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Longtier and John Irwin. Aspect oriented programming. ECOOP'97 LNCS 1241, pp 220-242. Springer-Verlag. June 1997.
- [3] Vinay Kulkarni, R. Venkatesh and Sreedhar Reddy. Generating enterprise applications from models. OOIS'02, LNCS 2426, pp 270-279. 2002.
- [4] Vinay Kulkarni and Sreedhar Reddy. Generating enterprise applications from models – experience and best practices. Workshop on Best Practices of Model Driven Software Development at OOPSLA 2005 (<http://www.softmetaware.com/oopsla2005/kulkarni.pdf>)
- [5] Vinay Kulkarni and Sreedhar Reddy. Model-driven development of enterprise applications. UML Satellite Activities 2004: 118-128
- [6] Vinay Kulkarni and Sreedhar Reddy. Separation of Concerns in Model-Driven Development. IEEE Software 20(5): 64-69 (2003)
- [7] MasterCraft – Component-based Development Environment. Technical Documents. Tata Research Development and Design Centre. <http://www.tata-mastercraft.com>
- [8] MOF Models to Text Transformations <http://www.omg.org/cgi-bin/doc?ad/05-05-15>
- [9] MOF Query / View / Transformations <http://www.omg.org/cgi-bin/doc?ad/05-09-01>
- [10] OMG, "UML Infrastructure 2.0 Draft Adopted Specification", 2003, <http://www.omg.org/uml/>
- [11] Peri Tarr, Harold Ossher, William Harrison and Stanley M. Sutton Jr. N Degrees of separation: Multi-dimensional separation of concerns. Proceedings of the International Conference on Software Engineering (ICSE'99) pp 107-119.