

# Model-Driven Engineering with Large Models in the Context of Product Line Engineering with IBYKUS AP

Steffen Skatulla, Detlef Hornbostel  
*IBYKUS AG for Information Technology, Erfurt, Germany*

Christian Erfurth, Wilhelm Rossak  
*Department of Computer Science, Friedrich Schiller University Jena, Germany*

## Abstract

*In software engineering practice models of software systems become more and more important and play a decisive role. Different model-driven approaches are followed by enterprises to achieve more efficient development techniques and higher flexibility. Practice has also shown that models reach a size and complexity that is hard to handle.*

*Over the last years, IBYKUS, a mid-sized software vendor with focus on product line engineering in the domain of business, administration and governmental processes, has put an effort into the development environment IBYKUS AP which supports Model-Driven Engineering. This paper describes the state of the used techniques on a conceptual level in the first part. The idea of Incremental Modeling and editable Model Views is presented in the second part accompanied with research questions to be tackled in future.*

## 1. Introduction

In recent years the way software is developed in practice has begun to change due to the need of shorter development cycles and a fast changing technology landscape. Complexity is tackled by abstract models of software systems which are transformed systematically to concrete implementations. Such software development approaches are known as Model-Driven Engineering (MDE). Stahl and Völter present a good overview of the field in [7]. As almost every new idea, MDE embraces new challenges in applying. France and Rumpe gave in [4] a classification of challenges: Model language challenges, separation of concerns

challenges, and model manipulation and management challenges.

These challenges have to be tackled especially when software systems become more complex. In the context of product line engineering, models of software systems may become huge and complex. For an effective processing, methods have to be found or adapted to handle issues like configurations, versions of a model, and dependencies between model elements.

Model partitioning and integration are key instruments to support the building of complex software product lines, using the MDE approach. First the application model has to be partitioned to enable variations among single products:

- basic version with missing components and
- extended versions with additional components or features,
- individual variants with selected predefined and individually developed customer specific features and components.

Second the model integration is obviously needed to combine a number of those model components into a consistent application model to form an individual product version.

## 2. MDE with IBYKUS AP

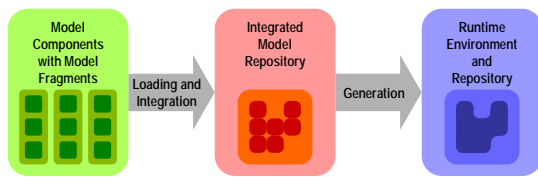
The paper presents an approach to model partitioning and integration that has been developed and successfully used for a large number of minor as well as major customer projects and continuously improved in the last 5 years at the IBYKUS AG [5]. IBYKUS is a mid-sized software vendor that is focused on the development of large individual client-server application systems in the domain of business,

administration and governmental processes using the IBYKUS AP development platform. AP follows the Model-Driven Software Development (MDS) that has been successfully used to develop a number of complex software systems during the last 7 years, e. g.

- systems for the management of agricultural promotion funds of the European Union, the controlling of financial resources and other governmental administration processes in several German federal states,
- applications to support business processes like claims and contract handling, project and innovation management as well as other industrial projects.

In those fields, IBYKUS builds product lines with versions that enhance over the years. Customer specific variants are built, that combine common components and features besides specifically adapted and individually developed parts.

With respect to MDE IBYKUS uses two basic strategies: a model partitioning and an integration concept supported by according tools and techniques. To describe these concepts, the overall structure of the IBYKUS AP development environment, as shown in Figure 1, needs to be explained first.



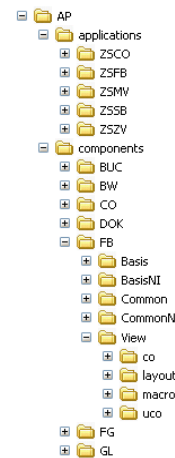
**Figure 1: Structure of the IBYKUS AP environment**

For each project there are a number of model fragments stored in partial models in a hierarchical file structure. These model fragments are loaded and thereby integrated into a model repository, which is held in a database. From this repository the application is generated and deployed to the IBYKUS AP runtime environment. The runtime environment is needed to run AP applications. On the one hand it provides components and libraries used by the application. But, more important, it contains an interpretative infrastructure. As described in [3] IBYKUS AP uses low level DSL (Domain Specific Language) configuration data that is interpreted by the user interface, the application container and the data store to form each single application. This interpretative runtime environment is fixed and does not need to be altered for different applications.

## 2.1. Model partitioning

There are a number of aspects according to which models are partitioned:

- *Software architecture*: The model is split accordingly into architectural components.
- *Application variance*: Customer specific model fragments are separated to represent different parallel variants of applications and its components.
- *Reuse*: Additionally partitioning is used to establish reusable model fragments and thereby reusable application components.
- *Development aspects*: Finally model partitioning provides the separation of different development concerns: structure, behavior, interface layout and others. Furthermore, teamwork support is provided.



**Figure 2: Model partitioning and integration**

Each subcomponent holds a specific subset or aspect of its component: Subcomponents with model elements that are shared among different versions are usually called ‘Common’. Subcomponents, modeling infrastructure basics are usually called ‘Basis’ if they are commonly shared (see Figure 2). The name ‘View’ indicates that a subcomponent provides some external interface to other components. Subcomponents modeling customer specific requirements are usually identified by a mnemonic denoting the customer (see ‘NI’ for the German federal state of Niedersachsen). This naming is not predefined but a consistent convention is established for each product line.

Technically each subcomponent contains a number of model files. On the one hand each of these sub

models is versioned for its own by checking the files into a CVS repository. On the other hand the applications assembled from the (sub)components, as described in the next section, is versioned independently too.

## 2.2. Model integration

The composition of applications is done by consecutive loading of the (sub)component models into an consolidated repository. The resulting integrated model is used to generate the application code and the low level DSL runtime configurations. Thereto each application or more exactly each single application version is defined as a list of (sub)components to be consecutively integrated to form the finally integrated application model.

Such a definition is shown in Figure 3: Therein the model for the application ZFSB called “Finanzbuchhaltung” is defined to be assembled from several components and subcomponents. It is composed of the component FB with the commonly shared subcomponents Basis and Common and the customer specific subcomponent NI. Additionally it comprises an interface to the external component CO, that is composed of the commonly shared subcomponent Basis and the interface subcomponent View. Furthermore it contains the components BUC and GL each with the subcomponents Common and NI.

```
<apapp_r name="ZFSB" version="1.05.006"
  label="Finanzbuchhaltung">
  <apsubcmpref parent="FB" name="Basis" />
  <apsubcmpref parent="FB" name="Common" />
  <apsubcmpref parent="FB" name="NI" />
  <apsubcmpref parent="CO" name="Basis" />
  <apsubcmpref parent="CO" name="View" />
  <apsubcmpref parent="BUC" name="Common" />
  <apsubcmpref parent="BUC" name="NI" />
  <apsubcmpref parent="GL" name="Common" />
  <apsubcmpref parent="GL" name="NI" />
</apapp_r>
```

Figure 3: Defining model integration

**2.2.1. Definition of merging possibilities.** As basis for the selection process of components, feature models [6] are a good choice to describe all merging possibilities - variants of our system. In product line engineering this method is popular to specify system variability. Czarnecki et al. presents a notation of feature models in [1], which can also be specified with a context free grammar. Figure 4 presents a sample section of components in a feature model notation.

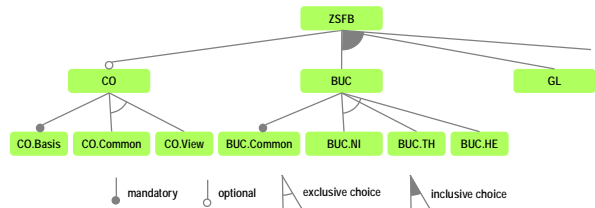


Figure 4: Feature Model with application components

**2.2.1. Merging techniques.** Different merging techniques are used in IBYKUS AP to integrate the partial models:

- adding,
- overwriting,
- hierarchical overwriting and
- model modifications.

To explain these techniques we need to mention that model elements in IBYKUS AP are organized hierarchically. There are top level elements like domain classes, ranges of values, central commands, user groups etc. and subordinate model elements like attributes and methods. Figure 5 shows the top level element “Domain class” and its most important subordinates.

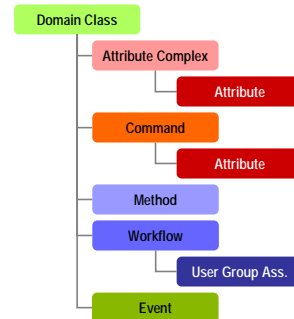


Figure 5: Hierarchical model structures

The *adding* technique is simply the possibility to add new model elements of a later loaded fragment to elements of the existing model repository. The position at which to add each new element is defined by a path along the hierarchy of model elements. This technique is used to add new subordinate elements as attributes and methods to existing domain classes as well as to complete new top level elements as domain classes with their subordinate elements. Using this technique a partially defined element can be completed or “umbrella” aspects like security can be woven into the

system model following aspect orient programming [2].

To replace top level or subordinate elements of a given model, the *overwriting* is used. Therewith an element can be replaced by a newly loaded model fragment. The overwriting can be performed even *hierarchically*, whereas the overwritten element and all its subordinates and relations are removed first. Following this way, model elements can be even removed completely from the integrated model. This technique is important to build customer specific variants from pre-existing models.

The last technique to mention is the model *modification*. Although modification could be done by overwriting it is useful in most cases to change single properties of existing model elements very selectively.

### 2.3. Practical experiences

Using the model driven generative software development techniques with IBYKUS AP, we experienced a boost of overall project performance of 20%-50%. Though it is rather difficult to number the increase in efficiency, the records of more than 20 major projects in the last 7 years show this significant reduction of project duration or team size or increase of accomplishable project size or complexity.

One example shows the typical performance boost quite well: In 2006 a legacy accounting solution of the German federal state of Niedersachsen processing about 1.2 millions booking records per year with about 15 sub-records each and an overall transaction volume of about 800 million Euros was replaced with a new solution. The legacy application had been developed over the last 10 years with 5-10 team members. In contrast the replacement project required 7 team members, 11 months and surpassed the functionality.

Especially the development of product lines is very effective. Although IBYKUS develops individual applications for its customers, in most cases there are common requirements among certain customers. For example the mentioned accounting application not only the requirements in that specific federal state but offers solutions to the other states. In fact this and most of the other IBYKUS applications are deployed to a number of customers. The point is that in spite of the common requirements each customer usually has its own needs ranging from different business processes and domain rules, varying data elements, to differing system infrastructures, environments and interfaces and many others. Especially for complex organizations as governmental organizations and major industrial companies this needs to be considered.

The continuation for the above example illustrates this pretty good: In 2007 the accounting solution was deployed to two other federal states and was adapted individually to both of them, parallel to the normal product line feature extension. The required adaptations were rather substantial: besides numerous minor changes, a complete subsystem, dealing with governmental procedures was completely replaced by an existing solution, financial handling rules and processes were tailored to the new customers and interfaces to the application environment of the new customers needed to be implemented.

### 2.4. Open issues

In the described MDS approach, which is currently in productive use at IBYKUS, there are certain issues we seek to improve.

Currently the practical reuse of existing application models is encountered mainly with respect to self-contained components and subcomponents. Here we strive for better support of *parametrizable template techniques* for modeling. For example we would like to be able to add historiography functionality to existing domain classes with a fixed set of new attributes and methods but also with the ability to specify, which attributes are kept in history records and to define the domain actions at which history is captured.

Furthermore we are looking for advanced *techniques for DSL building*. On the one hand we intend to build higher level DSLs based on the existing AP-DSL, used by IBYKUS to model business applications and processes. This could benefit customer projects, where we currently realize configurable applications 'manually'. One Example is a contract management and handling system for a major German telecommunications company to support the configuration and controlling of arbitrary service contracts. On the other hand we believe that IBYKUS could reach other markets, once we were able to fragment the existing IBYKUS AP DSL to strip down or replace subcomponents that do not fit actual projects. Examples may be the unplugging of integrated components like document management, workflow processing, mass printing service, message service etc. These could be individually assembled and/or replaced by third party components for each project.

Finally there is the clear demand for new techniques to handle model size and more important model complexity. Meanwhile the models in our product lines reach dimensions that are rather difficult to survey and to work with efficiently. To cope with this we propose,

that *techniques to separate concerns*, where a model engineer, designer or developer could focus on certain single aspects of a product line model at times, would be helpful.

### 3. Challenges

Right now IBYKUS is developing the next revision of its MDSD platform called APx. In this context new concepts of MDE, model partitioning and integration are explored. Two key issues are presented in the following subsections.

#### 3.1. Incremental Modeling

The idea behind the so-called *Incremental Modeling* or *Delta Modeling* is to describe models in terms of modifications. Conventionally models are defined and stored in some kind of enumeration or sets of interrelated model elements like domain classes, relations, attributes, methods and so on. Now models are to be described in terms of modification operations, like the adding of new or the removing or altering of existent model elements. For instance such an operation could describe the addition of some new attributes and methods to existing domain classes in order to add historiography functionality. These model modification operations can be interpreted as functions that transform a given model  $M_1$  into a new model  $M_2$ :

$$M_1 = t(M_2)$$

Following this approach, a model can be constituted as a nesting of modification operations starting with some given (potentially empty) model:

$$M = t_n(\dots t_2(t_1(M_0)))$$

Each function can have a number of parameters to control the model transformation:

$$M_j = t(M_i, p_1, \dots, p_n)$$

Then likewise the merging of two models  $M_i$  and  $M_j$  into  $M_k$  a resulting model is a special case of this:

$$M_k = m(M_i, M_j)$$

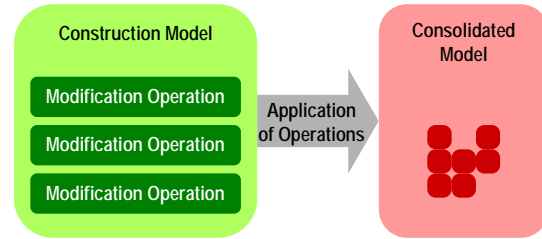
We can distinguish the following types of transformations:

- *Constructions*: a template model with placeholders is “filled in” with parameter values or whole parameter models. Like building blocks that are assembled, the template and the parameters are not modified themselves. Usually this type is to be used to build up new application models.
- *Modifications*: parts of an input model are modified, depending on parameters, to form the output model. To stick to the image, in contrast to

the latter type, the building blocks are reshaped to form the result. This type fits best, when existing application models need to be adapted to new requirements.

Whereas constructions may be realized similar to template engines modifications need techniques as FLWOR-expressions of XQuery [8] and XSLT [9] to locate, construct, change or remove model elements and parts.

Following this concept we get two model spaces as depicted in Figure 6. First the Construction Model as composed formula of modification operations and second the Consolidated Model as set of interrelated domain model elements, which are built up by the operations of the construction model.



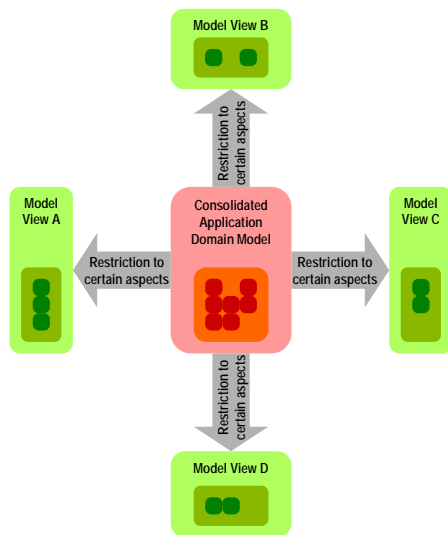
**Figure 6: Incremental Modeling with two model spaces**

The main advantage of this approach is that different application variations can be easily represented in the Construction Model as modifications in some part of the composed formula. New variations to meet individual needs of new customers are constructed as modifications or additions of the construction model. Another advantage is that these modifications can be grouped into compound sub-constructions and reused in other projects. Additionally this mechanism provides a powerful template technique.

In this area we are investigating practical issues like techniques to extract construction formulas from editing operations on the Consolidated Model, representation techniques for the Construction Model, techniques for incremental updating of the Consolidated Model on changes in the Construction Model and others. Simultaneously a number of theoretical issues are examined, e.g. equivalence, normalization and minimization of the Construction Model.

#### 3.2. Model Views

As the practical experience at IBYKUS shows, even domain models rapidly get very complex, although hiding lots of technical implementation details. Likewise the design and editing are usually restricted to certain aspects of these models at a time. Configuration management views the model in terms of components, variants and configuration parameters. Data(base) management has its view on the model with respect to domain entities, storage structures and transactions. Module implementation focuses on algorithms, source code and the like, in contrast to module integration, focusing on interfaces and so on.



**Figure 7: Model Views representing different aspects of a model**

To support these different aspects of application engineering IBYKUS researches into a technique we call *Model Views* or *Model Aspects*. As displayed in Figure 7 a number of different views are associated to the application domain model representing different aspects of it.

Such a Model View can be implemented as a restricted editor, which displays only defined aspects of the model and constrains modifications to these. Based on a view the editor provides the possibility to modify the system even by customers themselves.

As an alternative to the editor, we are studying means to extract sub-models into distinct storage structures containing only model elements relevant for the respective aspect and their reintegration into the Consolidated Model after modifications.

### 3. Summary

MDE concepts and approaches have been successfully applied in recent years in the development practice at IBYKUS. Thereby partitioning is necessary to handle large models and is a prerequisite for further integration to build product lines of software systems. An outline on techniques currently used in the development and runtime environment IBYKUS AP was presented in this paper.

In the area of product line engineering, the usage of models is essential for an efficient development. For a more holistic usage of models, investigations in further concepts are needed. As presented in this paper, for instance model transformations have to be formalized to support software product lines effectively. From an industry point of view, research effort should be spent on model support for:

- *description of system variants* based on models,
- *modeling model changes* like the presented Incremental Modeling with constructive and modifying model transformations,
- *separation of concerns* in models like the presented concept of Model Views,
- underlying basis techniques as *parametrizable template models* and technologies for *effective building of new DSLs*.

### 10. References

[1] K. Czarnecki, S. Helsen and U.W. Eisenecker, "Staged Configuration Using Feature Models", *Software Product Line Conference (SPLC) 2004*, pp. 266-283.

[2] T. Elrad, R.E. Filman, and A. Bader, "Aspect-oriented programming: Introduction." *Commun. ACM* 44, 10 (Oct. 2001), 29-32.

[3] C. Erfurth, W. Rossak, C. Schachtzabel, D. Hornbostel, S. Skatulla, "Concepts of Model Driven Software Development in Practice - Generic Model Representation and DSL Interpretation", *International Conference on Enterprise Information Systems 2007*, Funchal, Madeira, Portugal.

[4] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap", *Future of Software Engineering, International Conference on Software Engineering 2007*, IEEE Computer Society, Washington, DC, pp37-54.

[5] IBYKUS AG, <http://www.ibykus.com>, 2007.

[6] K. Kang, S. Cohen, J. Hess, W. Nowak and S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study", *Technical Report CMU/SEI-90TR-21*, Software

Engineering Institute, Carnegie Mellon University,  
Pittsburgh, PA, 1990.

[7] T. Stahl and M. Völter, *Model-Driven Software  
Development - Technology, Engineering, Management*,  
Wiley, 2006.

[8] W3C, <http://www.w3.org/TR/xquery/>, 2007.

[9] W3C, <http://www.w3.org/TR/xslt20/>, 2007.