Model Driven Architecture as Approach to Manage Variability in Software Product Families

Sybren Deelstra, Marco Sinnema, Jilles van Gurp, Jan Bosch
Department of Mathematics and Computer Science, University of Groningen,
PO Box 800, 9700AV Groningen, The Netherlands,
{s.deelstra|m.sinnema|jilles|j.bosch}@cs.rug.nl, http://segroup.cs.rug.nl

Abstract

In this paper we portrait Model Driven Architecture (MDA) as an approach to derive products in a particular class of software product families, i.e. a configurable product family. The main contribution of this paper is that we relate MDA to a configurable software product family and discuss the mutual benefits of this relation. With respect to variability management, we identify two main benefits of applying MDA to product family engineering, i.e. postponing binding time and mechanism selection to application engineering, and the independent evolution of domain concepts, product family assets and the transformation technique. In addition, we identify variability management as a solution to the problem of round-trip transformation in MDA.

Keyword(s): Model Driven Architecture, Software Product Families, Variability Management

1. Introduction

The notion of software product families has received substantial attention during the 1990s, and has proven itself in a large number of organizations. Software product families are organized around a product family architecture that specifies the common and variable characteristics of the individual product family members, as well as a set of reusable components that implement those characteristics. Products within a product family are typically developed in a two stage process, i.e. a domain engineering stage and a concurrently running application engineering stage.

Domain engineering involves, amongst others, identifying commonalities and differences between product family members and implementing a set of shared software artifacts (e.g. components or classes) in such a way that the commonalities can be exploited economically, while at the same time the ability to vary the products is preserved. During application engineering individual products are derived from the product family, viz. constructed using a subset of the shared software artifacts. If necessary, additional or replacement product-specific assets may be created.

The ability to derive various products from the product family is referred to as variability. Managing the ability to handle the differences between products at various stages of development is regarded as a key success factor in software product families. Variability is realized through variation points, i.e. places in the design or implementation that are necessary to make functionality variable. Two important aspects related to variation points are binding time and realization mechanism. The term 'binding time' refers to the point in a product's lifecycle at which a particular alternative for a variation point is bound to the system, e.g. pre- or post-deployment. The term 'mechanism' refers to the technique that is used to realize the variation point (from an implementation point of view). Several of these realization techniques have been identified in the recent years, such as aggregation, inheritance, parameterization, conditional compilation (see e.g. [5] [1]).

Model Driven Architecture (MDA), on the other hand, was introduced by the Object Management Group only a few years ago. MDA is organized around a so-called Platform Independent Model (PIM). The PIM is a specification of a system in terms of domain concepts. These domain concepts exhibit a specified degree of independence of different platforms of similar type, (e.g. CORBA, .NET, and J2EE) [11]. The system can then be compiled towards any of those platforms by transforming the PIM to a platform specific model (PSM). The PSM specifies how the system uses a particular type of platform [11].

The main contribution of this paper is that we relate MDA to a particular class of software product families, i.e. a configurable software product family, and that we discuss the mutual benefits of this relation. With respect to variability management, we identify two main benefits of applying MDA to product family engineering, i.e. postponing binding time and mechanism selection to application engineering, and the independent evolution of domain concepts, product family assets and the transformation technique. In addition, we identify variability management as a solution to the problem of round-trip transformation in MDA.

The remainder of this paper is organized as follows. In the next section, we discuss a classification of different types of software product families. In section 3 we relate MDA to on of these classes, i.e. a configurable product family. In addition, we provide our view on variability management in this context, as well as how variability management and MDA benefit from such an approach. Related work is presented in section 4 and the paper is concluded in section 5.

2. Product Family Classification and Model Driven Architectures

The basic philosophy of software product families is intra-organizational reuse through the explicitly planned exploitation of commonalities between related products. In our experience with industry we have identified that organizations employ widely different approaches to product family based development. Based on [3], we organize these approaches according to the s*cope of reuse*, i.e. the extent to which the commonalities between related products are exploited.

- **Standardized infrastructure:** Starting from independent development of each product, the first step to exploit commonalities between products is to reuse the way products are built. Reuse of development methodologies is achieved by standardizing the infrastructure with which the individual applications are built. The infrastructure consists of typical aspects such as the operating system, components such as database management and graphical user interface, as well as other aspects of the development environment, such as the use of specific development tools.

- **Platform:** With a standardized infrastructure in place, the next increase in scope of reuse is when the organization maintains a platform on top of which the products are built. A platform consists of the infrastructure discussed above, as well as artifacts that capture the domain specific functionality that is common to all products. These artifacts are usually constructed during domain engineering. Any other functionality is implemented in product specific artifacts during application engineering. Typically, a platform is treated as if it was an externally bought infrastructure.

- **Software Product Line:** The next scope of reuse is when not only the functionality common to all products is reusable, but also the functionality that is shared by a sufficiently large subset of product family members. As a consequence, individual products may sacrifice aspects such as resource efficiency or development effort in order to benefit from being part of the product line, or in order to provide benefits to others. Functionality specific to one or a few products is still developed in product specific artifacts. All other functionality is designed and implemented in such a way that it may be used in more than one product. Variation points are added to accommodate the different needs of the various products.

- **Configurable Product Family:** Finally, the configurable product family is the situation where the organization possesses a collection of shared artifacts that captures almost all common and different characteristics of the product family members, i.e. a configurable asset base. In general,

new products are constructed from a subset of those artifacts and require no product specific deviations. Therefore, product derivation is typically automated once this level is reached (i.e. application engineers specify a configuration of the shared assets, which is subsequently transformed into an application).

In the next section, we discuss how MDA relates to these product family classes and briefly discuss how products are derived in this context. In addition, we discuss the mutual benefits of combining MDA and variability management.

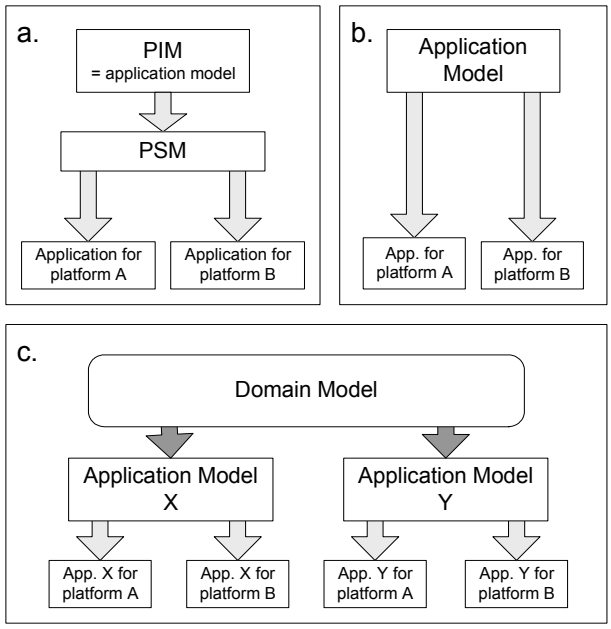

Figure 1 – How the MDA approach relates to software product family engineering. *Figure a. shows the traditional MDA approach. Figure b. shows the platform as a variation point that is resolved during the transformation step. Figure c. shows how MDA can be extended to enable model driven product family engineering.*

3. Variability Management in an MDA approach to Software Product Families

Conceptually, a software system that is specified according to the MDA approach specifies an application model for a family consisting of products that implement the same functionality on different platforms (see Figure 1a). The choice for the alternative platforms is a variation point in such a family. This variation point is separated from the application model in the sense that it is not visible in the specification anymore. Instead, it is provided in the infrastructure (consisting of the different platforms), and managed in the transformation definition (see Figure 1b).

The main benefit of MDA compared to traditional SPF development, is that the management of the platform variation point is handled automatically by the transformation step and is not a concern for the product developer anymore. The underlying platform, however, is not the only variation point that needs to be managed in a product family. The various product family members differ on both a conceptual level, i.e. their functional characteristics, as well as on the infrastructure level, i.e. which shared software assets are used to implement the alternative concepts.

MDA can easily be extended to accommodate this need, by adding a domain model that specifies places where alternative concepts can be selected. Selection of different concepts from this domain model then results in different application models, which, provided that the right transformation definitions are implemented, can already be automatically transformed within to the existing MDA

approach (see Figure 1c). This presumes that the MDA transformations are powerfull enough to deal with both platform and other technical variability, and that all variations in the domain models can be implemented or provided by the infrastructure (consisting of the asset base). Relating the latter aspect to the classification of product families described in section 2, MDA thus only presents a real benefit for a configurable product family.

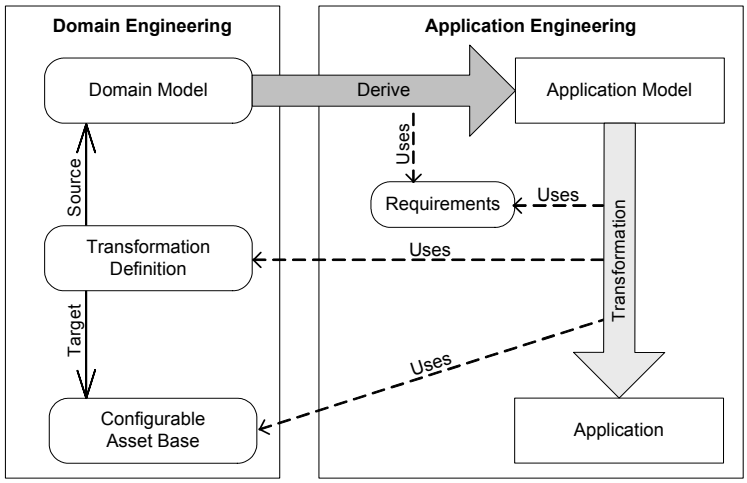

Figure 2 – Product family engineering in an MDA world. *Domain concepts specify the functionality of the product family independent from the implementation details of the asset base. The transformation definition then defines how the domain concepts are mapped on the asset base. Based on the product requirements, an application model is derived by selecting alternative concepts specified in the domain model. The application model is then compiled to a working application using the transformation definition, the asset base and the product requirements.*

3.1 Product derivation in an MDA world

Assuming that there is a configurable software product family, the two main processes can now be described as follows (see also Figure 2):

Domain engineering: Domain engineering involves the following three main activities. The first activity involves the creation and maintenance of the asset base, which consists of reusable and/or variable assets. The second activity involves the specification of the domain model, which consists of the domain concepts. The third activity involves the definition of the transformation definition, which specifies how instances of the domain model are converted into a working application using the asset base. This transformation definition could be a compiler for a domain specific language or an elaborate environment such as KOALA [10], where applications are specified in an ADL and some glue scripts.

Application engineering: Application engineering then boils down to creating an application model by selecting alternative domain concepts from the domain model. In this process alternative concepts are selected based on customer requirements. The application model is then compiled to a working application using the transformation definition, the asset base and the customer requirements. Essentially, the transformation process binds infrastructure level variability that is not visible in the domain model anymore, as well as variation points that are selected on the conceptual level, to the realization of those variation points in the infrastructure.

3.2 Benefits

The main advantages for variability management in this approach to product family engineering are as follows.

- **Postponing binding time and mechanism selection to application engineering.** Rather than polluting the application model with implementation details of the variation points on the conceptual level, such as binding time and mechanism, the mapping to the realization in the infrastructure is handled in the transformation. The main advantage of this separation in comparison to traditional product family development is that the choice for a particular binding time and realization mechanism can now be postponed to the stage at which the application model is compiled. In other words, provided that the appropriate transformations are defined, it is possible to select a binding time and mechanism for a variation point that is most suited to the application requirements during application engineering.
- **Independent evolution of domain concepts, transformations and infrastructure.** The separation of specification and implementation furthermore allows for the independent evolution of the variation points and variants specified in the domain concepts, and the variability realized in the asset base. Managing the evolution of variability in both domain concepts and infrastructure is effectively moved to the transformation definition, as the transformation definition is responsible for the mapping between domain concepts and the infrastructure. A main advantage in comparison with traditional product family development is that applications can benefit from infrastructure or transformation evolution with a simple re-compilation, rather then requiring adaptations to the application model.

In addition to the advantages that MDA presents with respect to variability management, variability management in turn presents a possible solution to the problem of round-trip transformation in MDA. We elaborate on this below.

- **Using variability management to eliminate the need for round-trip transformation.** In cases where the transformations turn out to be not powerful enough, systems that are specified in platform independent models require adaptations at the level of the platform specific model. The problem with such a situation is that once changes have been made at the platform specific level, the PIM cannot be changed and re-compiled or all changes to the PSM will be lost. The MDA community therefore identified the need for round-trip transformation, i.e. a reversible transformation between the PIM and the PSM. This need can be removed however, by specifying the platform specific variations as variation points in the PIM. These variation points then explicitly specify where those variations are allowed, and during the transformation these platform specific variants are inserted in the PSM. This approach is similar to programming assembly within a higher level language such as C.

4. Related Work

In the past few years, a number of books on software product families emerged, such as, [14] [6] [4] and our co-author's book [2]. In a paper by the same co-authour [3] a maturity classification of product families is presented that consists of the following levels: standardized infrastructure, platform, software product line, configurable product base, programme of product lines, and product populations. In this paper we extracted four of those levels according to the scope of reuse.

A number of common techniques for realizing variability are discussed in [5] and [1]. Other techniques are focused around some form of infrastructure-centered architecture where variability is realized in the component platform, e.g. CORBA, COM/DCOM or Java-Beans [13], or around code fragment superimposition, e.g. Aspect-, Feature- and Subject-oriented Programming, [8], [12] and [7], respectively.

A paper related to deriving different application models is [9], which proposes extending UML with architectural constraints to enable automatic derivation of product models from the product family architecture.

5. Conclusions

While product families have received wide adoption in industry during the 1990s, MDA was proposed by the OMG only a few years ago. In this paper, we portrayed MDA as an approach for a particular class of product families, i.e. a configurable product family. We discussed how products are derived in such an approach and how MDA and variability management benefit from each other.

The main contribution of this paper is that we related MDA to a configurable software product family and discussed the mutual benefits of this relation. With respect to variability management, we identified two main benefits of applying MDA to product family engineering, i.e. postponing binding time and mechanism selection to application engineering, and the independent evolution of domain concepts, product family assets and the transformation technique. These aspects are not tackled by traditional product family development. In addition, we identified variability management as a solution to the problem of round-trip transformation in MDA.

Acknowledgements

This research has been partially sponsored by the EU under contract no. IST-2001-34438.

6. References

[**1]** M. Anastasopoulos, C. Gacek, *Implementing Product Line Variabilities*, Symposium on Software Reusability (SSR'01), Toronto, Canada, Software Engineering Notes, Vol. 26, No. 3, pp. 109-117, May 2001.

[2] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Pearson Education (Addison-Wesley & ACM Press), ISBN 0-201-67494-7, 2000.

[3] J. Bosch., *Maturity and Evolution in Software Product Lines; Approaches, Artifacts and Organization*, in Proceedings of the SPLC2, August 2002.

[4] P. Clements, L. Northrop, *Software Product Lines – Practices and Patterns*, Addison-Wesley, 2002.

[5] I. Jacobson, M. Griss, P. Jonsson, *Software Reuse – Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.

[6] M. Jazayeri, A. Ran, F. Van Der Linden, *Software Architecture for Product Families: Principles and Practice*, Addison-Wesley, 2000.

[7] M. Kaplan, H. Ossher, W. Harrisson, V. Kruskal, *Subject-Oriented Design and the Watson Subject Compiler*, position paper for OOPSLA'96 Subjectivity Workshop, 1996.

[8] G. Kiczalez, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, J. Irwin, *Aspect Oriented Programming*, in Proceedings of 11th European Conference on Object Oriented Programming, pp. 220-242, Springer Verlag, 1997.

[9] L. Monestel, T. Ziadi, J. Jézéquel, *Product line engineering: Product derivation*, Workshop on Model Driven Architecture and Product Line Engineering, associated to the SPLC2 conference, San Diego, August 2002.

[10] R. van Ommering, F. van der Linden, J. Kramer, J. Magee, *The Koala Component Model for Consumer Electronics Software*, IEEE Computer, vol. 33-3, pp 78-85, March 2000.

[11] MDA Guide v1.0, OMG, 2003.

[12] C. Prehofer, *Feature Oriented Programming: A fresh look on objects*, in Proceedings of ECOOP'97, Lecture Notes in Computer Science 1241, Springer Verlag, 1997.

[13] C. Szyperski, *Component Software – Beyond Object-Oriented Programming*, Pearson Education Limited, 1997.

[14] D. M. Weiss, C.T.R. Lai, *Software Product-Line Engineering: A Family Based Software Development Process*, Addison-Wesley, 1999.