An MDA®-based framework for model-driven product derivation

Øystein Haugen‡, Birger Møller-Pedersen‡, Jon Oldevik#, Arnor Solberg#

‡University of Oslo, #SINTEF
{oysteinh|birger}@ifi.uio.no, {jon.oldevik|arnor.solberg}@sintef.no

ABSTRACT

In this paper, we present a flexible conceptual model for product family engineering. The conceptual model takes advantage of the new expressiveness and precision of UML 2.0. We also present some simple additions to UML to provide efficient modeling of system families. The conceptual model is used as basis for modeling system families at different abstraction levels and for performing semi automatic product derivation in alignment with the structure and philosophy of the Model Driven Architecture (MDA). The paper also presents how a tool can facilitate the product derivation. In summary, we describe how the MDA approach might be concretized in a system family engineering context.

KEYWORDS

Product families, conceptual models, model transformation

1 Introduction

Model-driven engineering and system family engineering has gained much attention in both academia and the industry. For the model-driven engineering area the introduction of the Unified Modeling Language (UML) and the Model Driven Architecture (MDA) has been important to broaden its popularity and employment.

In many domains and market areas it is more and more critical for product vendors to provide many different products to the market with only slight differences in functionality, quality and design (e.g. the mobile phone domain). This has led to a lot of attention to system family engineering.

Both model-driven engineering and system family engineering focus on improving efficiency and quality of both the system lifecycle processes, such as development and maintenance, and the delivered products. Improvement of reuse and code generation is important means for both approaches. (In MDA, code generation might be seen as a special case of model transformation).

This paper presents a framework that provides a conceptual model for system family engineering. The conceptual model is defined based on MDA technologies and serves as the basis for both modeling and product derivation. Our goal is to provide a sound framework for modeling system families using UML 2.0 and provide mechanisms and tool support for model-driven system derivation. The remaining parts of this paper are structured as follows: Chapter 2 describes our conceptual model, chapter 3 describes an example, chapter 4 describes how we combine model-driven and family engineering. Chapter 5 describes related work, and chapter 6 concludes our work.

2 The conceptual model

In the following we present a conceptual model for the domain of system modeling and product derivation.

A cardinal point of this model is that we distinguish between the market-oriented concepts and the design-oriented concepts. This distinction is motivated from the experience that what the market sees as products and product lines may not necessarily map one-to-one to how the designers choose to model their realization of the products.

The market-oriented concepts try to capture the concepts needed by organizations that provide product lines to their markets. Product lines define a set of products with various features that are used in marketing processes. The products and their features are realized through design-oriented concepts that may be organized in system families in which common, reusable elements are defined. A system family defines architectural assets (reusable elements) that products can/must use to be part of the product line. The family is a framework that in turn may use other, more general frameworks that are not specific for the particular system family in question.

2.1 Market view of products and product lines

Product lines denote collections of a set of related products defining a market segment. Thus, the set of products within a product line are not necessarily technically related (as by definition is the case for members of system families). A product line typically is a chain of products that are related from a marketing point of view. An example is Microsoft Office, consisting of a set of related products (PowerPoint, Word, Excel etc.), which jointly supports the needs for, and in effect defines, a market segment. Interestingly enough Microsoft Office is an example of how the relationship between the product line and the system family has changed over time. In the early versions of Microsoft Office there was little reuse between the individual products, while later versions exploit what could be termed the Microsoft Office system
family. On the market side the increased reuse between MS Office modules is of little interest.

The product line conceptual model describes the market-oriented terms associated with products and product lines. The conceptual model is shown in Figure 1.

2.2  System family conceptual model

The system family conceptual model describes the essential terms for reasoning about and specifying system families and systems (Figure 2). A system family defines technical and architectural commonalities, variabilities, and constraints for its members.

A system family constitutes a domain-specific framework to be used both when deriving and developing individual members. The architecture of a system family may contain variability through a set of variation points. Some of these must be resolved in order to produce a system configuration that implements a product.

The system family architecture is realized on one or more platforms on which systems execute. Thus, our concept of framework goes beyond the case where families are just represented by a collection of domain specific classes or components, and where a given system is made by composition of these. We consider the case where the structure and the behavior of the systems are given by a system family.

![Figure 1 Market Concepts – Product Line](image1)

**Figure 1 Market Concepts – Product Line**

Product lines are characterized by the set of features it provides as well as the tailored set of member products. The features might be of two kinds; functional features and quality features. The quality features describe the quality of service (QoS) and address how well the functionality is (or should be) performed if it is realized.

The features of a product are derived from the features of the product line. Product line variability is reduced in a product, i.e. it is a selection of the set of features from the product line. Variability in this model is represented in terms of features that are optional. Furthermore, features may have relations to other features constraining variability, such as ‘requires’, ‘conflicting’ and ‘alternative’. These terms are defined in existing product family literature, such as [5].

![Figure 2 System Family Concepts](image2)

**Figure 2 System Family Concepts**

The executable characteristic of a system family also implies that analysis can be performed on the system family itself and not only on the system members.

As mentioned, a product line defines a market segment that is provided with a set of products. However, a product line does not require that the member products are technically related. Thus, a given product line is not necessarily associated with a system family. On the other hand, a system family is defined to always correspond to at least one product line. A product line has zero or more members, which opens for defining a market segment before any concrete products are provided. Systems are derived from system families to realize products. A system realizes one or more products, whereas a product is a member of zero or more product lines.

2.3  System Derivation

System derivation is the combined process of specifying a product from a product line and realizing it through a system that is derived from a system family.

The goal of the system derivation process is to reach a configuration of the system family in which necessary variabilities have been bound, to define a more specific system. In the general case the derivation of a system from a product is a non-trivial derivation, but in the following we will concentrate on a subset of such situations where the product features and the variation points of the system family map one-to-one. In this simplified situation the product specification goes through these subprocesses:

1. From a product line model choose a consistent set of features which then fully specifies the product.
2. From the product specification and an appropriate system family model derive the realized system as a configuration of the system family.
In the simplified case finding the appropriate system family is considered obvious. The system derivation conceptual model describes the relationship between system families and systems (Figure 3).

**Figure 3 System Derivation**

The system family has two distinct aspects. One aspect is the set of concepts that are potentially used by the systems of this family. Another aspect is the architecture that is considered common among all systems of the family.

In the simplified case we assume that the system family has an optional variation point corresponding to each feature of the product line. An instantiation of the common architecture is called a configuration and it specifies an architecture in an extended form of the UML 2.0 composite structure that conforms to the commonalities, but still defines specifics. This process should be supported by a transformation tool. The rationale for the system derivation conceptual model is to combine the traditional decision model mechanism used in many system family approaches with the transformation and code generation techniques of the MDA. The next section addresses this in more detail.

3 Conceptual model example - the watch family

In this section we describe a simple system family to illustrate the usage of the conceptual model and how the product derivation is performed. The watch family [10] is a system family from which digital watches are derived. It is deliberately chosen a small example for the purpose of efficient illustration of the approach.

3.1 The Product Line Model

According to the conceptual model we have a market view defined by a product line which is characterized by features and its member products. We have found the UML use case diagram as a convenient vehicle for modeling this view. The main features of the Watch family are: Time, Timer, Stopwatch, Alarm, and WorldTime (Figure 4).

**Figure 4 Products and features of the product line**

The use cases in Figure 4 represent the functional features of the product line. The quality features are UML classes stereotyped with QoS. The product line itself is represented with a UML package and stereotyped with Product Line. The products are represented by actors. Thus, the diagram defines the set of features provided by a product. Note that the quality feature Precision is associated with the product line, implying that each product of the product line has that feature. The attribute indicates that the allowed values for the Precision feature are either 1/10, 1/100, or 1/1000 of a second. The interpretation of the dependency association between the Waterproof quality feature and the “Sporty Watch” product is that the “Sporty Watch” product must be waterproof, and consequently due to the inheritance, also the “Advanced Watch” product must be waterproof.

3.2 The System Family Model

The system family view describes the scope of the system family at the architecture level, the general architecture of the family and the variation configuration model.

UML 2.0 composite structures are suitable for describing the detailed architecture of systems and system families. The recursive nature of the parts, and the ability to refine composites, makes this a powerful instrument. A composite structure specification can be detailed with behavior specifications, such as state machines or sequence charts.

The common architecture of the watch family are defined as a UML 2.0 composite structure as shown in Figure 5, together with the class definitions in Figure 6, which defines e.g. the class WatchApplication used in the
The composite structure and different subclasses. (Its behavior in terms of an associated state machine is not shown.)

![Figure 5 Watch System Family Architecture View](image)

The composite structure of the system family defines the connections between parts of the architecture. In effect, it defines the rules for every configuration of the watch. For example, it defines that any Watch has a single Clock that sends ticks to a Watch Application. UML2.0 defines that such a part (here being a set with multiplicity 1..n) can have objects of any subclass of WatchApplication. In effect, we have defined a family of Watch systems where this part of the Watch is constrained to have objects of these four subclasses. We might also allow more subclasses, resulting in a larger system family. In the extreme case we will even have categories for subclasses implementing watch applications that were not foreseen when the watch architecture was defined.

![Figure 6 Watch System Family Classes](image)

Note that by restricting to just making new subclasses of WatchApplication, the watch architecture in Figure 5 is still valid, as the rest of the architecture only assumes the applications to have properties as defined in class WatchApplication. Thus, the system family class view is a view on the architecture that complements the composite structure and is used to constraining the system family architecture.

An important aspect of the system family class view is to specify the variation points in the system family architecture that must be resolved when deriving a system. The variation points specified by using the variation point stereotype <<vp>> implies that this is a variation that need to be resolved at configuration time. This approach is similar to that used in KobrA[1], except that in our approach, a class cannot be a variation point in itself. Variation points are specified on the properties of classes, such as associations, attributes and operations. Variation on classes is done by means of configuration.

In the example, the Watch class is a component that represents the watch family. It is stereotyped <<SystemFamily>>. We show variation points (<<vp>>) specified for associations between components and some on attributes and operations.

The next section elaborates how these models can be used and supported in a model-driven family derivation process.

4 Model-driven family engineering

4.1 Family engineering and MDA

Traditionally within product family engineering, the process of deriving products has been a manual process, where the transformation from system family models to specific product models has been done manually, based on decisions and resolutions described textually.

The approach described here integrates family engineering with an MDA-philosophy that leverages the transformation processes from products line towards running system. This framework utilizes the system family architecture, the architecture variabilities, and the feature specifications to provide tool support for the system derivation.

System family engineering is a discipline which differs from traditional system engineering specifically by the explicit identification and description of common and variable parts of a set of products. The main idea is that the common parts advocate reuse (in the traditional sense), whilst the variable parts advocate an explicit decision process to determine the concrete shape and size of a product, based on a set of assets.

One of our goals has been to integrate model-based thinking to improve this process and to leverage MDA for the system family engineering domain. When looking at the MDA Guide, it is clear that OMGs current visions of model-driven architecture are quite open for interpretation. In principle, these can be models at any abstraction level, depending on your definition of platform. For the sake of this paper, we will consider platform-independent models to be any models that still have variability with respect to the target execution platform.
Transformation between abstraction levels is one of the key elements of MDA [8]. Indeed, there are need for mechanisms for handling the transitions and relations between model abstraction levels. Figure 7 depicts how we see that product family models and the transition between them relates to MDA.

**Figure 7 Transformation between system family levels**

The Product Line Model defines the scope of the product family in terms of features that are required or optional. It resembles a requirements model, and as such, it is a CIM. The System Family Model defines the architectural elements of the system family. It defines the design of the system family realizing product line features, including architectural variation points. In MDA terms, it is a PIM. The relation between the product line model and the system family model can be represented as a QVT transformation, where the relationships between market-oriented product line terms and system-oriented system family terms are defined.

The System Model is defined through a decision process, where some variations from the system family are resolved. This too, is a PIM. The transformation from the System Family Model to the System Model is a mapping which is partly automated and partly supported by human intervention. A System Model may itself be subject to refinement into more specific systems, again supported by transformation. Finally, the running product/system can be generated from a system model, supported by a code generating transformation process, as described in the next section.

### 4.2 Tool support for product derivation

To improve product development and maintain traceability between the system family and the different systems, we have developed a tool prototype that supports model-driven system derivation. The tool is based on user interaction to achieve correct product derivation. Based on the variation points defined in the product line and system family models, a graphical decision models is generated, from which the user can configure a product.

The tool maintains the constraints defined by the product line models, such as conflicts between certain features, or requirements-relationships between features. If a certain feature choice implicates another feature, the tool automatically detects this. Figure 8 gives a snapshot of the tool after an *AlarmWatch* product has been derived from the *Watch* family, showing the Composite Structure of the product.

Our goal with the tool is to generate the new models based on the decisions made by the user. In practice, the generated models will serve as suggestions to a system specification, which later can be modified by the user. The environment within which the product derivation tool works should be integrated with the overall product development processes and tools. Ideally, it should be integrated with UML 2.0 tools, which provide the modeling capabilities needed to support system family modeling. The source and target specifications in a product derivation are then artifacts in a productive, model-driven process.

**Figure 8 Feature selection tool and a derived *Alarm Watch***

### 5 Related work

The work described in this article is based on ongoing work in the FAMILIES project [2] concerning product family modeling and model transformation and work in the QuA project [3] on model transformation and quality features.

In the Kobra method [1], they have a conceptual model with some similarities to ours. They use the term *System* to denote generic description at the level of product line and *Application* as an instantiation of the generic system. These are in essence the same as our concepts of System Family and System, except that they do not see the
System Family as an executable asset. In KobrA, there is no separation and relation between system families, systems, product lines and products. For product derivation, KobrA describes a decision model and a decision resolution model in tabular form, where variation points are resolved. They do not provide any semi automatic product derivation and do not use UML-models and transformation techniques in order to accomplish this task.

Becker[4] gives an extensive description of variability in product families. Our core system family concepts are more or less in alignment with his general model for product family variability. The model we present contains the core concepts and is simpler than Beckers. It is however possible to extend our model using concepts he defines. Our derivation approach is related to Becker’s abstraction levels, although ours is based on MDA-related model abstractions. Also, we utilize mechanisms of UML 2.0 composite structures and apply transformation perform product derivation. Becker does not address the separation of market and system concepts.

There is currently no uniform and consistent notation for modeling variabilities and commonalities in system families. In our approach we use UML as the modeling language. The KobrA approach is hybrid and uses both UML and natural language to describe their models and variabilities (e.g., functional model and decision model). Clauß[5] describes an approach for modeling variability with UML. He defines a set of UML extension by means of UML stereotypes and constraints. In our approach we have tried to minimize the use of stereotypes (only variation point (<vp>)) and instead utilized UML own build in mechanisms for modeling variability. (E.g instead of Clauß[5]'s <<optional>> stereotype we use UML association multiplicity 0..1).

The work on UML 2.0 composite structures and configurations are based on the experience from the UML 2.0 standardization effort, through the work of Haugen and Møller-Pedersen[6]. The work on model-driven derivation is related to the ongoing standardization on Query, View and Transformation (QVT)[8] and the FAMILIES work on this topic. In [7], a method using variability patterns in UML and OCL is used as a basis to automate product derivation. [9] gives a brief overview of the relationship between MDA and product family engineering.

6 Conclusions and future work

In this paper we have presented a framework for product family engineering that utilizes MDA technologies and techniques both to define the framework with its conceptual model and to automate the product derivation process. In the framework we define a conceptual model that supports the modeling of product lines, products, system families and systems. We also provide a precise definition of these concepts as well as their relationships, and show how they are used for modeling using UML 2.0. The main contributions of this paper, apart from the conceptual model, are the illustrated techniques for describing system families and system configurations using UML 2.0, and the tool-supported product derivation process. The framework shows how to define the product lines in terms of features and variabilities, and the corresponding system family architecture. Further, it shows how the derivation tool provides support for interpreting these models, providing the end user with assistance in the product derivation process, and automatically derives specific product configurations.

We will continue to develop this framework within projects like FAMILIES and QuA. In particular we will continue our research in:

- The specification of variation and variation points within family architectures, and the relation to model driven derivation process.
- UML 2.0 and configuration of behavior.
- Product derivation prototype integrated with UML tools and tools that are emerging in the wake of the OMG QVT [8] standardization process.

7 References