Acta Univ. Sapientiae, Informatica **10,** 1 (2018) 43–57

DOI: 10.2478/ausi-2018-0003

On the use of model transformation for the automation of product derivation process in SPL

Nesrine LAHIANI
LRDSI Laboratory
C.S. Department
Saad Dahlab University
Blida, Algeria
email:
lahiani.nesrine@gmail.com

Djamal BENNOUAR
LIMPAF Laboratory
C.S. Department
Akli Mohand Oulhadj University
Bouira, Algeria
email:
djamal.bennouar@univ-bouira.dz

Abstract. Product Derivation represents one of the main challenges that Software Product Line (SPL) faces. Deriving individual products from shared software assets is a time-consuming and an expensive activity. In this paper, we (1) present an MDE approach for engineering SPL and (2) propose to leverage model-to-model transformations (MMT) and model-to-text (MTT) transformations for supporting both domain engineering and application engineering processes. In this work, we use ATL as a model-to-model transformation language and Acceleo as a model-to-text transformation language.The proposed approach is discussed with e-Health product line applications.

1 Introduction

Companies are more and more forced to customize their software products for completely different customers. In practice they often clone an existing system

Computing Classification System 1998: C4, D.2.11 D.2.13
Mathematics Subject Classification 2010: 68N99
Key words and phrases: product derivation, software product lines, model-driven engineering, domain specific language

and adapt it to the customer's needs. In such scenarios software product lines promise benefits, for example, reduced maintenance effort, improved quality, and customizability. However, introducing new development processes into a company is risky and might not pay off . The other advantage is that this fairly recent software development paradigm allows companies to create efficiently a variety of complicated products with a short lead-time.

In a software product line context, software products are developed in two phases, i.e. a domain engineering process and an application engineering process. Domain engineering a basis is provided for the actual development of the individual products. During application engineering individual products are derived from the product line, i.e. constructed using a subset of the shared software artifacts. If necessary, additional or replacement product-specific assets may be created.

The key activity in application engineering is Product Derivation. It addresses the construction of a concrete product from the product line core assets, which includes the derivation of application artifacts from domain artifacts, for instance the derivation of Application Requirements from Domain Requirements, the derivation of the Application Architecture from the Domain Architecture and the derivation of Application Components from Domain Components.

In this context, this paper proposes a model-driven product derivation approach based on Model-Driven Engineering principles [20, 21] . We use (1) metamodels to represent domain concerns such as application, architectural, or technological; (2) models that conform to metamodels to designate particular products of product lines; (3) model transformation programs to derivate members of the line from an initial model. Transformation programs are composed by transformation rules. Each transformation rule is responsible for producing a part of the final product. To derive a complete product, we have to assemble the rules in a precise ordering that determines the order in which the individual parts are produced and assembled. To express configurable variability we use feature models. Our feature model represents variation points and variants according to user needs. From the feature model, a product Designer defines a Configuration with his choices. Consequently, our big challenge is to produce adapted transformation programs to contain rules able to derive products with the desired user choices.

The remainder of this paper is structured as follows: Section 2 discusses related work, while Section 3 introduces the terminology and concepts used in this work.In Section 4 we present an overview of our approach for product

derivation. Section 5 illustrates the application of our approach on a case study. Finally, Section 6 presents the conclusions.

2 Related work

In this section, we cite the state-of-the-art related to product derivation approaches. Perovich et al. in [19] employ model-driven techniques to transform a feature model to specific product architectures. However, the domain design is specified in terms of ATL transformation rules, therefore the transformation processes is not completely automated. Such an approach is complex and makes the SPL architecture design process difficult.

An approach is proposed in [4, 5] to derive the architecture of a product by selectively copying elements from the SPL architecture based on a product-specific feature configuration. The SPL architecture model contains variability to cover all products aspects. This approach deals only with the derivation of the high level product architecture. The mapping between features and the components realizing their implementation is done through an implementation model. A prototype that implements the derivation as a model transformation is described in the Atlas Transformation language.

Tawhid et al. in [22] proposed to derive an UML model of a specific product from the UML model of a product line based on a given feature configuration is enabled through the mapping between features from the feature model and their realizations in the design model. The mapping technique proposed aims to minimize the amount of explicit feature annotations in the UML design model of SPL. Implicit feature mapping is inferred during product derivation from the relationships between annotated and non-annotated model elements as defined in the UML metamodel and well-formed rules. The transformation is realized in ATL.

Gonzlez-Huerta et al. in [11] presented a set of guidelines for the definition of pattern-based quality-driven architectural transformations in a Model-Driven SPL development environment. These guidelines rely both on a multimodel that represents the product line from multiple viewpoints as well as on a derivation process that makes use of this multimodel to derive a product architecture that meets the quality requirements.

Parra et al.[18] propose an approach for feature-based architecture composition in component-based software product lines. To fill the gap between features and software components, authors rely on the definition of aspect-like composition models that link every particular feature with several software

components. The approach detects that one feature requires a second feature when the pointcut that define the variation point for the first feature references source code elements referred to by the aspect that defines the second feature. The approach can detect when one feature excludes a second feature, when the pointcuts (that define the variation points) for both features refer to the same source code elements.

All the preceding approaches constitute good effort to provide a smooth transition from feature models to product architectures. In addition, some approaches such as [19, 22] use model-driven techniques to transform a feature model into product architectures. However, as the domain design is specified in terms of ATL transformation rules [3], the transformation processes cannot be fully automated. Other similar efforts that rely on aspect-oriented techniques[18] to derive product architectures from feature selections .Although, the derivation at higher levels of abstraction, that is from generic to concrete product line architectures, is poorly addressed.

3 Terminology and basic concepts

In this Section we describe the main terminology and basic concepts of the different areas involved in our approach.

3.1 Product lines

Definition (Product lines). *A Software Product Lines can be defined as is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way [6].*

Definition (Feature modeling). Feature modeling is the activity of identifying externally visible characteristics of products in a domain and organizing them into a model called a feature model. The feature modeling described in this section is based on that of [7].

Definition (Product derivation) We focus in this paper at application engineering known also as product derivation (PD). PD has been defined in many different ways. McGregor in [16] describes the process as *"Product derivation is the focus of a software product line organization and its exact form contributes heavily to the achievement of targeted goals"*.

Deelstra et al. in [8] define product derivation by *"A product is said to be derived from a product family if it is developed using shared product family artifacts. The term product derivation therefore refers to the complete process of constructing a product from product family software assets"*.

3.2 Model-driven engineering

Definition (MDE) Kent defines Model Driven Engineering (MDE) by extending MDA with the notion of software development process (i.e., MDE emerged later as a generalization of the MDA for software development) [12]. MDE refers to the systematicuse of models as primary engineering artifacts throughout the engineering lifecycle. All the definition of MDE are based on the concept of model, meta-model, and model transformation.

Definition (Meta-Model) A model is frequently considered an instance conforming a meta-model. Based on [14] *"a meta-model is a model of a modeling language where the languageis specified"*.

Definition (Model transformation) Performing a model transformation by taking one or more models as the input and producing one or more models as the output requires a clear understanding of the abstract syntax and the semantics of the source and the target models. Metamodelling is a key concept in MDA that defines the abstract syntax of the models and the inter-relationships between the model elements [13].

The common setting for all transformation languages is such that the model to be transformed (source model) is supplied as a set of class and association instances conforming to the source metamodel. The result of transformation is the target model - the set of instances conforming to the target metamodel. Therefore the transformation has to operate on instance sets specified by a class diagram.

4 Model-driven product derivation approach

In this section, we present an overview of our approach for product derivation. It is founded on the principles and techniques of software product lines and model driven engineering. Figure 1 illustrates the main elements of our approach and their respective relationships.

4.1 Domain engineering

Domain analysis. Domain analysis [17] or Feature modelling is the first activity to define the commonality and variability that can be expected to occur among the SPL members identified in the product line's scope. We use feature model to present the similarities and variations among the products identified in the product line's scope that can be expected to occur.

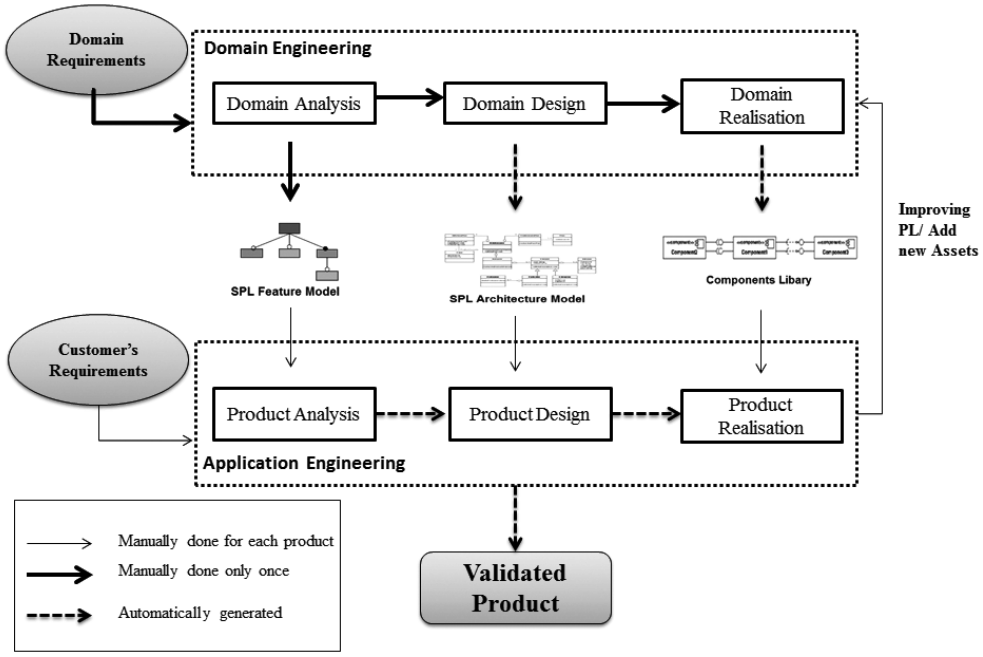

Figure 1: Overview of our approach

To build our metamodel we modify and simplify the metamodel proposed by Czarnecki et al. [7]; we depict it in Figure 2. All Features in the Feature Model have different names and may be composed of several members.

Domain design. At this stage the proposed derivation approach uses the mapping technique [15] in aim to map features to architecture model. After that, feature model is considered as an input parameter and then is processed by a model-to-model (M2M) transformation written in ATL (Atlas Transformation Language)[2] that creates an Architecture Model which is composed of a set of rules and helpers. The rules define the mapping between the source and target model. The helpers are methods that can be called from different points in the ATL transformation. This model describes all components that have to be included to implement this particular Application Feature Model. We need to create in the target model all the model element types that compose a component model as its shown in Figure 3.

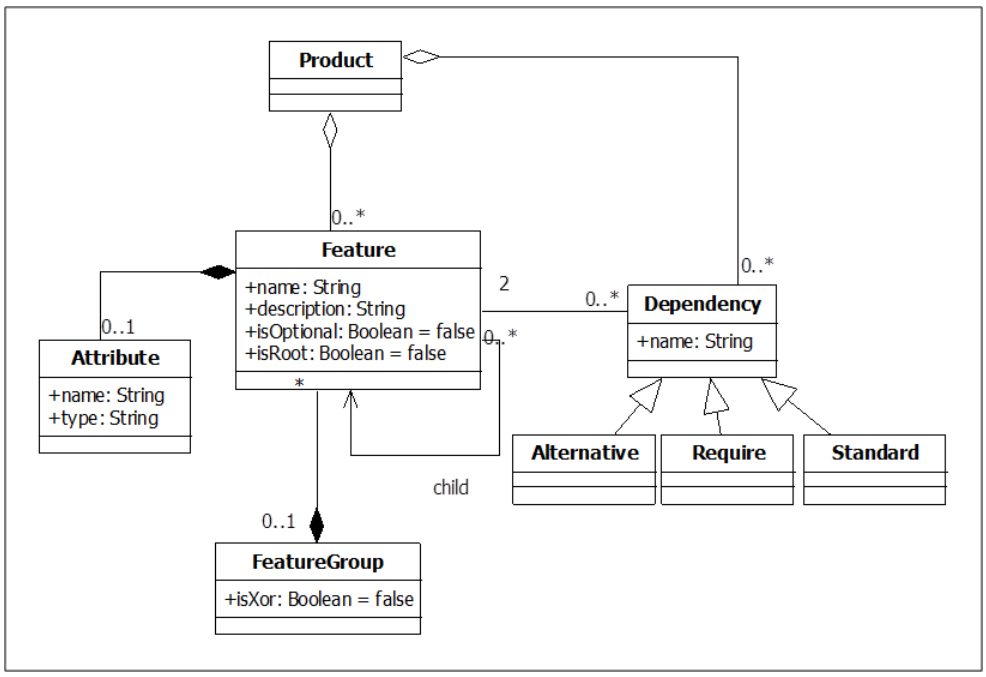

Figure 2: UML metamodel for feature models.

Domain realization. The goals of the domain realization sub-process are to provide the detailed design and the implementation of reusable software assets, based on the Architecture Model obtained in the domain design. In addition, domain realization incorporates configuration mechanisms that enable application realization to select variants and build an application with the reusable artifacts. The model obtained in Domain Design is then processed by a model-to-text (M2T) transformation which generates an equivalent textual configuration implemented using Acceleo language [1] to promote the generation of Java. This tool specializes in the generation of text files (code, XML, documentation) starting from models. Using Acceleo we can generate the source code based on templates and models expressed with EMF [9].

4.2 Application engineering

Product analysis. The main goal of product analysis is to document the requirements artifacts for a particular application and at the same time reuse, as much as possible, the domain requirements artefacts. A feature configura-

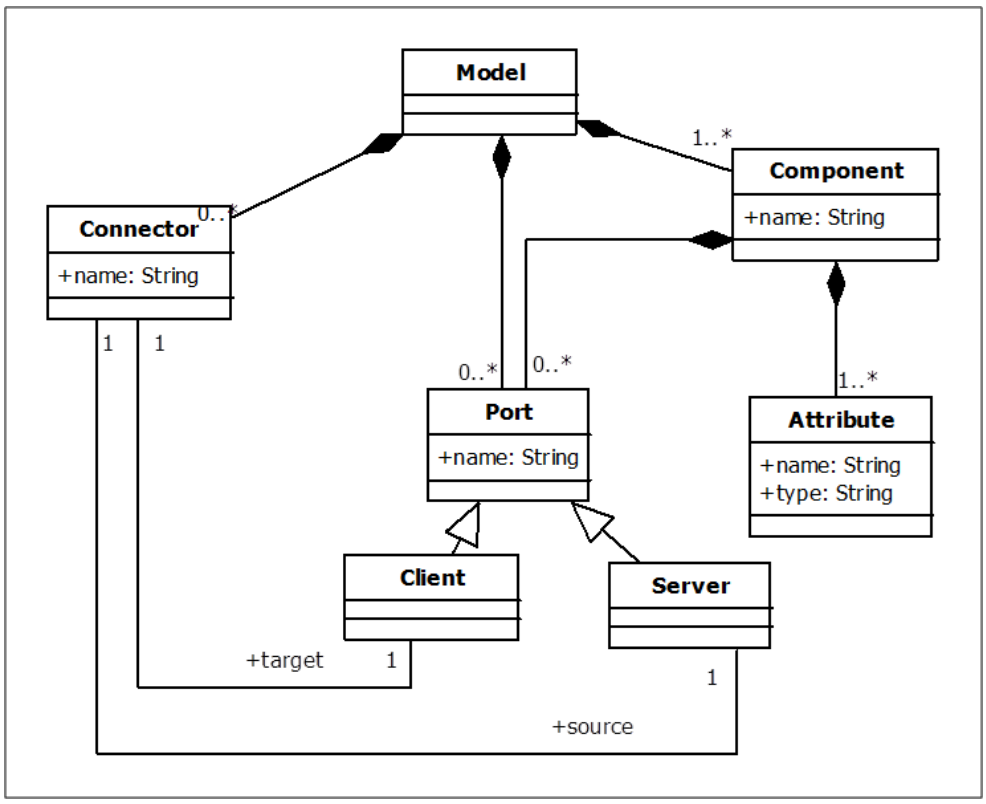

Figure 3: UML metamodel for Component models.

tion is the production of this activity which is a legal combination of features that specifies a particular product. This activity uses feature models as input to select the feature relevant for customers requirements to build the product and identify the specific-assets of the product. Once the selection is checked and validated by the product designer the output at this stage is a specialized version of feature model (application feature model).

Product design. The main goal of the product design activity is to produce the product architecture model. The product architecture model is defined for the particular product being developed, considering its desired features defined in the feature configuration model.

Product realization. The main goal of product realization is to build the actual product, taking in consideration the product architecture defined in the previous activity. The corresponding component implementations developed

during the domain implementation must be used to obtain the implementation of the product.

5 Case study

Health-related Internet technology applications delivering a range of clinical care, content, and connectivity, are referred to collectively as e-health. The most remarkable attribute of e-Health is that it is enabling the transformation of the health system from one that is barely focused on curing diseases in hospitals by health professionals, to a system focused on keeping citizens healthy by affording them with information to take care of their health whenever the need arises, and wherever they may be. E-health is promoted as a mechanism to bring growth, gain, cost savings, and process improvement to health care.

In this context of an e-Health application, we present in this section a simple case study to illustrate the overall process, from the feature model to the final product.

5.1 Domain engineering

The first activity is domain analysis where we define the feature model for e-Health Product Line, as Figure 4 (part A) shown doctors could connect via the application to follow up (1) remote consultation (via phone/message) and (2) manage patients accounts. Patient also must do (3) a registration so that he/she can consult and (4) pay using its own credit card or just by bank transfer which are alternative features only one could be chosen. Drug refill and offline consultation are two optional features that could be chosen or just left.

Second activity, the domain design where we build the feature-to-architecture transformation rule artifact is built. We use a model-to-model transformation we developed to create an initial version of this model from the feature model, only containing all defined features and their member relationship. We present a fragment of one of the rules using the ATL specialization of our metamodel illustrated in Figure 3, using textual notation. Final activity in domain engineering is the domain realization. The architecture-to-components transformation rule artifact is built. We use a model-to-text transformation we developed to create an initial version of this model from the architecture model, only containing all defined features and their member relationship

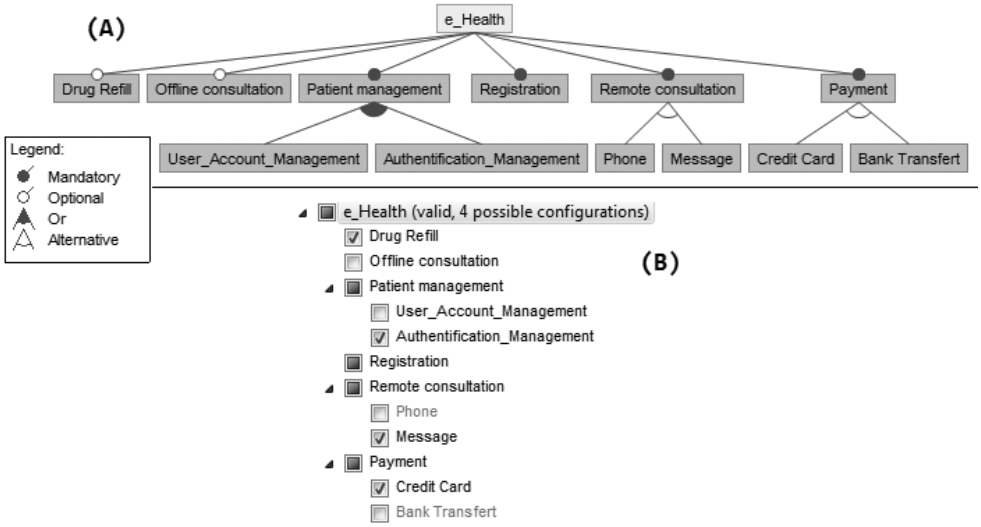

Figure 4: e-Health product line: (a) Feature Model tree for e-Health applications (b) Feature Configuration Model for e-Health.

5.2 Application engineering

During product analysis we use FeatureIDE [10] an Eclipse plug-in for Feature-Oriented Software Development to create the feature configuration model defining the desired features in the new product being built. A feature configuration is a legal combination of features that specifies a particular product. We use a text-to-model transformation to obtain the model shown in Figure **??** (part B),that illustrates the selected features as an instance of the metamodel shown in Figure 2.

During product design, the meta-transformation is used to generate from the feature-to-architecture transformation rule the Model Architecture artifact. The proposed model transformation approach takes as input the SPL source model and generates a product target model. Our transformation generating a concrete product model from a SPL model is implemented in (ATL). An ATL transformation is composed of a set of rules and helpers. The rules define the mapping between the source and target model, while the helpers are methods that can be called from different points in the ATL transformation. This transformation is then applied to the feature configuration model to automatically generate the product architecture. Here we show an example of an ATL helper and rules:

```
module MyRules; -- Module Template
create OUT: Components from IN: Features;
rule Componnent{
from
 e : Feature!Feature
to
 out : Component!Component (
           name <- e.name, )
}
rule Association{
from
 e : Feature!Dependency
to
 out : Component!Connector (
           name <- e.name, )
}
rule Attribute {
from
 e : Feature!Attribute
to
 out : Component!Attribute (
           name <- e.name,
           type <- e.type )
}
rule Port {
from
 e : Feature!Operation
to
 out : Component!Port (
           name <- e.name,
           type <- e.parameter->select(x|x.kind=#pdk_return)->
            asSequence()->first().type,
           parameters <- e.parameter->select(x|x.kind<>#pdk_return)->
           asSequence()
     )
}
```

As Figure 5 illustrates a fragment of the resulting PRODUCT ARCHITECTURE model generated by the rules, applied to the FEATURE CONFIGURATION MODEL shown in Figure 4. The e-Health product line application component is composed by the subcomponents generated by the rules. Final activity in application engineering is the application realization. At this stage,

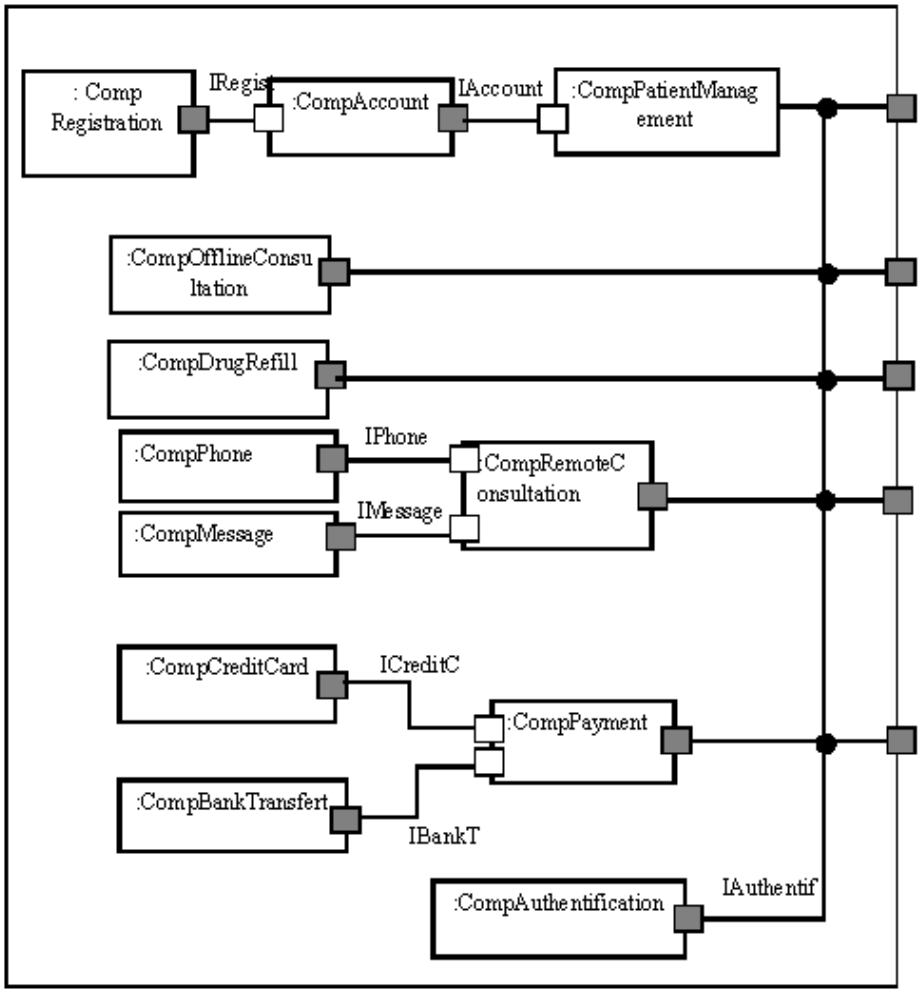

Figure 5: Component Model for e-Health product Line applications.

we write a program that generates Java code from our previously created architecture model using Acceleo, which navigates the model and creates the source code ($*$.java files for Java). Our goal is to transform the features into java classes and Attribute into class properties, and finally generate set and get methods for class properties. here is the code used to create a bean for each of the classes defined in our target model:

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/3.0.0/UML')/]

[template public generate(aClass : Class)]
[file (aClass.name.concat('.java'), false)]
  public class [aClass.name.toUpperFirst()/] {
  [for (p: Property | aClass.attribute) separator('\n')]
    private [p.type.name/] [p.name/];
  [/for]

  [for (p: Property | aClass.attribute) separator('\n')]
    public [p.type.name/] get[p.name.toUpperFirst()/]() {
      return this.[p.name/];
    }
  [/for]

  [for (o: Operation | aClass.ownedOperation) separator('\n')]
    public [o.type.name/] [o.name/]() {

    }
  [/for]
  }
[/file]
[/template]
```

6 Conclusion

The main objective of a product line is reusability.Various assets are being used in software product lines. These assets have different values. Also, the values of them differ from the value of the profit obtained through using these assets is different. Derivation of a product from an SPL seems to be an easy step since its relied on reuse. Actually the product derivation represents one of the main challenges that SPL faces due to time-consuming. In this paper, we intended to reduce the development time of a product by automating the derivation by generating some java code using Acceleo in conjunction with ATL. The proposed transformation uses Feature-architecture mapping technique by instantiating the initial feature model, an instance of feature model is constructed according to customers requirements. Then, separate features into two kinds: common and variable. The main idea is to create for each feature a component or a set of components combined in a specific way. Linking

these created components together based on the relationships among features in the feature model is the last step of our process. Although testing is of main importance in the context of product lines due to high reuse, in this paper we do not cover testing activities and it is one of the limitations of our proposed approach. This paper has illustrated by means of e-Health application the overall process, from the feature model to the final product. As future work, we will add more features to e-Health product line application and also intend to build a new set of components. A possibility is to apply our approach on other product line applications as e-Vote and also add testing activity to the approach.

References

[1] Acceleo Project, [Online]. Available:https: //eclipse.org/acceleo. ⇒49
[2] ATL Project, [Online]. Available: http: //www.eclipse.org/atl/. ⇒48
[3] J. Bèzivin, G. Dup, F. Jouault, G. Pitette, & J. E.Rougui, First experiments with the ATL model transformation language: Transforming XSLT into XQuery. *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture* Vol. 37. 2003 ⇒46
[4] G. Botterweck, K. Lee, S. Thiel, Automating product derivation in software product line engineering, *Software Engineering*, Kaiserslautern, 2009, pp. 177–182. ⇒45
[5] G.Botterweck, L. OBrien, & S. Thiel. Model-driven derivation of product architectures. *Proc. of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 469–472. ⇒45
[6] P. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. The SEI series in software engineering. Addison–Wesley, Boston, 2002. ⇒46
[7] K. Czarnecki, S. Helsen, U. Eisenecker. Staged configuration using feature models. *Int. Conf. on Software Product Lines*, Springer Berlin Heidelberg, 2004, pp.266–283. ⇒46, 48
[8] S. Deelstra, M. Sinnema, J. Bosch, Product derivation in software product families: a case study. The Journal of Systems and Software, **74** (2005) 173–194. ⇒46
[9] Eclipse Modeling Framework, [Online]. Available: http://www.eclipse.org/emf/. ⇒49
[10] FeatureIDE,[Online] Available: `http://wwwiti.cs.unimagdeburg.de/iti_db/research/featureide/`. ⇒52
[11] J. Gonzàlez-Huerta, E. Insfran, S. Abrahao, J. D. McGregor, Architecture derivation in product line development through model transformations, *22nd Int. Conf. on Information Systems Development*, 2013, pp. 371–384. ⇒45
[12] S. Kent, Model driven engineering, *Int. Conf. on Integrated Formal Methods, Lecture Notes in Comp. Sci.*, **2335** (2002) pp. 286–298. ⇒47

[13] A. G. Kleppe, J. B. Warmer, W. Bast, *MDA Explained: the Model Driven Architecture: Practice and Promise*, Addison-Wesley Professional. 2003. ⇒47

[14] I. Kurtev. *Adaptability of model transformations*, PhD Thesis, University of Twente Research Information. ⇒47

[15] N. Lahiani, D. Bennouar, A software product line derivation process based on mapping features to architecture, *Proc. of the Int. Conf. on Advanced Communication Systems and Signal Processing ICOSI*, 2015. ⇒48

[16] J. McGregor, Goal-driven product derivation, Journal of Object Technology, **8** 5 (2009) 7–19. ⇒46

[17] L. Northrop, P. Clements, With F. Bachmann, J. Bergey, G. Chastek, S. Cohen, P. Donohoe, L. Jones, R. Krut, R. Little, J. McGregor, L. OBrien, *A framework for software product line practice, version 5.0*, Software Enginering Institut, 2012. ⇒47

[18] C. Parra, A. Cleve, X.Blanc, L. Duchien, Feature-based composition of software architectures. *European Conf. on Software Architecture*, Springer, Berlin, Heidelberg, 2010, pp. 230–245. ⇒45, 46

[19] D. Perovich , P. O. Rossel, M. C.Bastarrica, Feature model to product architectures: Applying MDE to software product lines, *Software Architecture, & European Conf. on Software Architecture*, WICSA/ECSA 2009, Joint Working IEEE/IFIP Conf., IEEE 2009, pp. 201–210. ⇒45, 46

[20] D. C. Schmidt. Guest editors introduction: model-driven engineering. *IEEE Comput.* **39** 2 (2006) 25–31. ⇒44

[21] T. Stahl, M. Voelter, K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*, John Wiley & Sons 2006. ⇒44

[22] R. Tawhid , D. C. Petriu. Product model derivation by model transformation in software product lines. *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 14th IEEE International Symposium*, IEEE 2011, pp. 72–79. ⇒45, 46

Received: January 14, 2018 • Revised: April 5, 2018