Development of a plug-in for Variability Modeling in Software Product Lines

María Karen Cortés-Verdín*, María Lucía López-Araujo* and Jorge Octavio Ocharán-Hernández*

ABSTRACT

Software Product Lines (SPL) take economic advantage of commonality and variability among a set of software systems that exist within a specific domain. Therefore, Software Product Line Engineering defines a series of processes for the development of a SPL that consider commonality and variability during the software life cycle. Variability modeling is therefore an essential activity in a Software Product Line Engineering approach. There are several techniques for variability modeling nowadays. COVAMOF stands out among them since it allows the modeling of variation points, variants and dependencies as first class elements. COVAMOF, therefore, provides an uniform manner for representing such concepts in different levels of abstraction within a SPL. In order to take advantage of COVAMOF benefits, it is necessary to have a computer aided tool, otherwise variability modeling and management can be a hard tasks for the software engineer. This paper presents the development of a Eclipse plug-in for COVAMOF.

RESUMEN

Las Líneas de Productos de Software (LPS) toman ventaja económica de las similitudes y variación entre un conjunto de sistemas de software dentro de un dominio específico. La Ingeniería de Líneas de Productos de Software por lo tanto, define una serie de procesos para el desarrollo de LPS que consideran las similitudes y variación a lo largo del ciclo de vida. El modelado de variabilidad, en consecuencia, es una actividad esencial en un enfoque de Ingeniería de Líneas de Productos de Software. Existen varias técnicas para modelado de variabilidad. Entre ellas resalta COVAMOF que permite modelar los puntos de variación, variantes y dependencias como entidades de primera clase, proporcionando una manera uniforme de representarlos en los diversos niveles de abstracción de una LPS. Para poder aprovechar los beneficios de COVAMOF es necesario contar con una herramienta, de otra manera el modelado y la administración de la variabilidad pueden resultar una labor ardua para el ingeniero de software. Este trabajo presenta el desarrollo de un plug-in de COVAMOF para Eclipse.

Recibido: 19 de Diciembre del 2011
Aceptado: 14 de Febrero del 2012

INTRODUCTION

A Software Product Line (SPL) is defined as "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 that are developed from a common set of core assets in a prescribed way" [1]. The main benefit of a SPL is to economically exploit the similarities and differences between the software products derived from it.

The SPLs have appeared in Software Engineering as an approach whose goal is to create variations from a common infrastructure. The emergence of this new approach in the software industry brings what is called Software Product Line Engineering. This engineering gives companies productivity gains, increased product quality, decreased time-to-market and more efficient use of resources [1].

Software Product Line Engineering includes two processes [2]: 1) Domain Engineering, which develops the common elements, that is, the creation and establishment of a software platform that considers the similarities and differences to be achieved, in other words, the variability is defined and modeled.

Palabras clave:
COVAMOF; líneas de productos de software; variabilidad; modelado de variabilidad; punto de variación; variante; dependencia; eclipse; plug-in.
Keywords:
COVAMOF; software product lines; variability; variability modeling; variation point; variant; dependency; eclipse; plug-in.

*School of Statistics and Informatics, Universidad Veracruzana, Av. Xalapa esq. Avila Camacho S/N, C.P. 91020, Xalapa, Veracruz, Mexico. E-mail: kcortes@uv.mx, zS10019519@malum1.uv.mx, jocharan@uv.mx

(2) Application Engineering, which has to do with the development of systems that are members of the SPL from the common elements and taking into account the variability model defined. Both processes can be viewed separately but must work together in order to succeed.

Variability is defined as the ability to change or customize a system [3]. It occurs when the similar and varible characteristics of SPL are identified. The variability is found in two groups of systems: customizable systems and SPL [1]. Customizable systems are those in which the variability is mainly due to the selection by the user of the elements that matter most and SPL are series of relatively similar products that allow reuse of the common assets.

The variability modeling is important because it allows the management of the variability and development of a clear and specific documentation of SPL and resolve the differences found in them. The variability models generated in the development of a SPL can have a number of features and a large number of combinations among them. Therefore, it is practically impossible to manage and document the variability of complex models without the aid of tools. On the other hand, there is a wide variety of formalisms for modeling variability at different levels and in various artifacts, making the management and documentation of variability more complicated. It is also important to model variation points, variants and dependencies as first-class elements, considering the concepts of domain regardless of their representation in the specific development artifacts. COVAMOF (ConIPF Variability Modeling Framework) [2] is a framework for variability modeling which is explained in further detail in Section *COVAMOF*. To take advantage of all the features that COVAMOF offers it is desirable to have a computer aided tool. Unfortunately the tool presented in [3] is not available at the moment and its development seems to have stopped. In agreement with one of the authors of COVAMOF, the development of a plug-in for modeling variability under COVAMOF was decided.

The support that a computer aided tool that follows COVAMOF framework to a SPL architect is extremely important. The architect of the SPL is responsible for the development of the architecture, which is the artifact where the quality of the products of SPL is defined [4]. As such, the architecture must also incorporate the modeling of variability at this level. The main motivation of the work presented here is to incorporate the COVAMOF framework to AOPLA [5] which is an approach to design a SPL architecture. This article presents the development of an Eclipse plug-in that supports the COVAMOF framework and is organized as follows: Section *COVAMOF* explains the COVAMOF framework, Section *Development in the Eclipse Platform* explains the details of the Eclipse platform which is the basis for the development of plug-in. Section *COVAMOF Plug-In* explains the work done so far in the development of plug-in. Finally conclusions are given.

COVAMOF

COVAMOF (ConIPF Variability Modeling Framework) is a variability modeling framework [2]. It represents variation points, variants and dependencies at different levels of abstraction of a system as first-class elements. The levels of abstraction in a SPL are features, architecture and component implementation. COVAMOF considers two views: the Variability View and Dependency View.

The COVAMOF Variability View (CVV) is the graphical representation of variation points and dependencies. The main objective of this view is to show to the software engineer the multiple options for features that are available in different abstraction levels and to analyze their relationships at different levels. COVAMOV defines two views on the CVV: the Vatiation Point View and the Dependecy View. The Variation Point View contains the following elements:

1. **A variation point** represents the place where the option is available for a LPS. A variation point is the element or subject that varies within a domain artifact [2]. There are three types of variation points in the CVV [2]:

    - **Optional** which is the result of selecting zero or a variant of one or more variants associated with the variation point.
    - **Alternative** is the selection of a variant between one or more associated with the variation point.
    - **Optional variant** corresponds to the selection (zero or more) between one or more variants associated.
    - **Variant** is the selection of one or more of one or more variants.
    - **Value** is the value that can be chosen from a predefined range.

    For each variation point there are several properties defined: a description and information about its state, the rationale behind the binding, the realization mechanism and its associated binding time. Variation points that do not have associated resolution mechanism are those that are realized by variation points at lower levels of abstraction. This type of relationship between variation points is represented by a realization relation.

2. **Variants** are the entities that represent the options available to realize a variation point. According to [2] a variant is defined as "the representation of an object of variability within the domain artifacts." Variants can represent anything from an object or class, a file to a block of code.

3. **A realization relation** specifies the rules that determine the variants to choose from a variation point in a lower level. As stated, variation points are determined by variation points at lower levels of abstraction are used relations resolution. Thus traceability is achieved within the hierarchical structure of the SPL.

For its part, the Dependency View main objective is to show restrictions affecting the choice of variants of the variation points. The dependencies originate from the domain of application, the platform, the design details and the implementation of quality attributes. These dependencies specify how the binding of a variation point influences the value of the property in the system [2]. On the other hand, a Dependency Interaction occurs when the variation points are part of multiple dependencies.

With the use of the Variation Point View and the Dependency View the software engineer has a broader picture of the variation points and dependencies associated with the products and, thus, has the elements necessary to come up with a development strategy during the early stages of the process of SPL. The graphic elements of the view point and dependence COVAMOF variation shown in fig. 1.

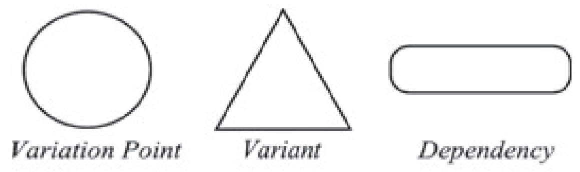


Figure 1 . COVAMOF graphical notation.

In order to exploit the characteristics of the COVAMOF framework it was developed a tool for the automation support as presented below.

DEVELOPMENT IN THE ECLIPSE PLATFORM

Eclipse is a platform that allows the integration of different applications in order to build an Integrated Development Environment (IDE). Eclipse is divided in three subprojects: Eclipse Project, Eclipse Tools Project and Eclipse Technology Project. Unlike other platforms, in order to provide all of its functionality, the Eclipse IDE is formed by plug-ins.

Eclipse Project [6] [7] supports the development of a platform or framework for the implementation of an IDE. Eclipse Tools defines and coordinates the integration of different categories of tools based on the Eclipse platform. For example, the graphical editors employ the Graphical Editing Framework (GEF) and the Eclipse Modeling Framework (EMF). Eclipse Technology Project is a higher level project in the Eclipse Foundation. The Technology projects deal with exploring technology and have a limited life cycle. A project ends when the results of the research are published or when it is incorporated into another project [6] [7].

Eclipse provides several frameworks for the development of graphical, software model manipulation, web and other kinds of applications. Among such frameworks, the following were employed in the work presented here: 1) GMF [6] provides a component for the generation and execution of graphical editors based on GEF and EMF; 2) EMF [6] supports the generation of code for the development of tools and applications based on a structured model; and 3) GEF which is a framework for building visual editors.

A model is an abstract representation of a domain. It captures the important elements of the domain and its relationships. A metamodel models characteristics of the models and the elements that form them [7]. For developing an Eclipse plug-in a metamodel must be developed. Such a metamodel is defined by using EMF by means of a model definition language named Ecore. Ecore constitutes the core of the project. GMF defines a process in order to develop a project or tool [7]. Such a process was followed in the development of the COVAMOF plug-in and it is explained in the following Section.

COVAMOF PLUG-IN

The objective for developing the COVAMOF plug-in for Eclipse is to support variability modeling for a SPL according to COVAMOF framework. The plug-in was developed with the support of GMF and EMF in

Eclipse. The activities carried out to develop the plug-in, are explained next.

First, a use case model was developed in order to guide the development of the plug-in development. The use case model describes the variability modeling related activities that the user will be able to perform with the aid of the COVAMOF plug-in. Such activities must comply with the COVAMOF framework. For the first version of the plug-in, the scope includes creating variation points, variants and dependencies use cases. Next, the plug-in was developed using the Eclipse platform, with the aid of GMF y EMF. According to the process depicted in fig. 3, the plug-in development was as follows:

1. **Domain Model.** The metamodel was defined according to the Ecore domain model, generating next the domain generation model. Such a model is depicted in fig. 4 where it can be seen how the COVAMOF metamodel was translated into the EMF metamodel. Due to space restrictions it is not possible to give a more detailed view of the translation. However, as it can be seen in the figure, the classes of the COVAMOF metamodel were translated into the Ecore. The properties of each one of the classes, as defined in COVAMOF, can also be noticed in fig. 4. It is important to mention that the abstract_layer class allows for indicating the Variation point realizations among abstraction layers.
2. **The second activity.** Corresponded to the definition of the plug-in metamodel. Such a metamodel is based on the COVAMOF metamodel which defines the structure of the COVAMOF framework elements and their relationships as they constitute the CVV. The COVAMOF metamodel is shown in fig. 2.
3. **Domain Gen Model.** The generator model is created and, from this model, the EMF classes of the model are generated. Such a model serves to generate the code of the editor.
4. **Tooling Def Model, Graphic Def Model and Mapping Model.** Tooling, Graphic and Mapping models are combined. With this, the tool bar of the COVAMOF plug-in is created.
5. **Diagram Editor Gen Model.** A configuration file is created in order to generate the diagram code. All previous processes are combined to create the final diagram editor.

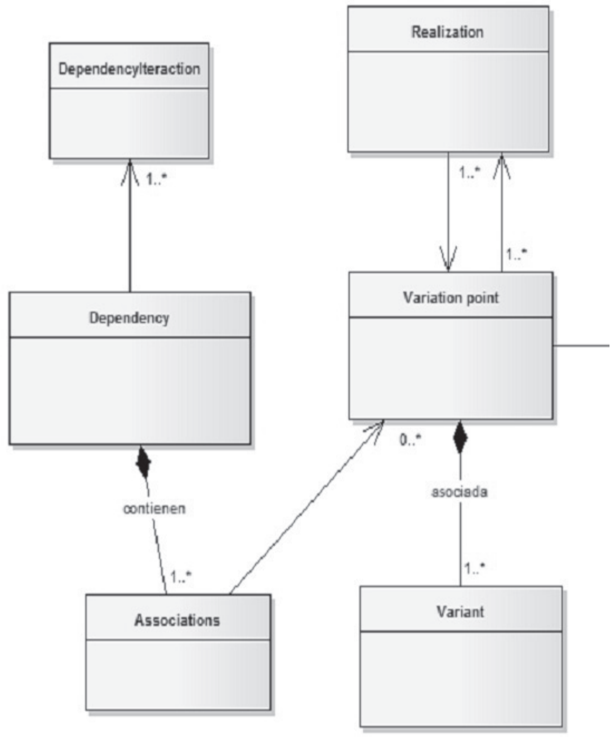


Figure 2 . COVAMOF metamodel.

The display of the COVAMOF plug-in display seen in fig. 5. The variability model that is shown in fig. 5 is an excerpt of a SPL variability model. This SPL corresponds to a tool for academic activities and products and it is still under development. On the right side of the display, the tool bar is shown. The COVAMOF notation can be noticed on the upper side. On the bottom of the tool bar, the arrows that establish relationships among variation points, variants an dependencies, can also be seen.

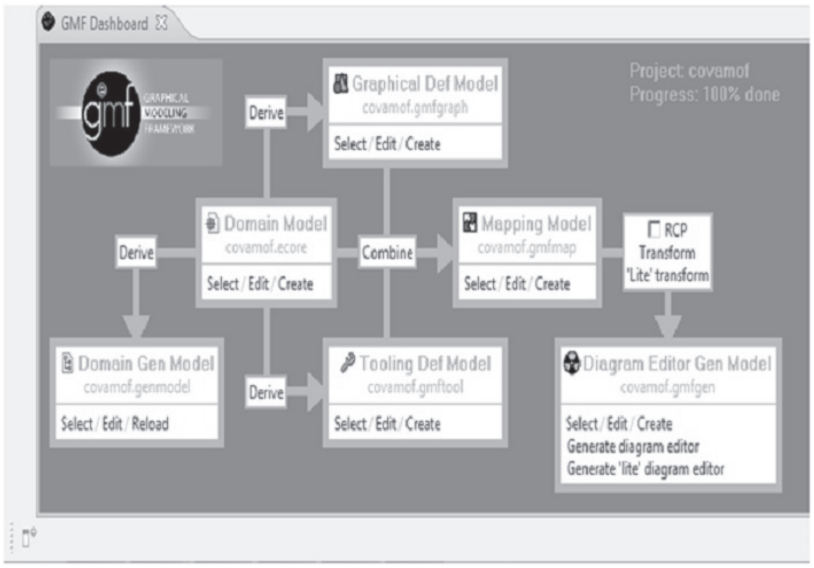


Figure 3 . Production flow for a GMF editor.

The plug-in has been tested with several examples found in [2] [3] and [8], and it is currently being tested in the development of SPL that is employing AOPLA for the design of the architecture. It must be reminded that the main motivation for the development of the plug-in is to support variability modeling in AOPLA [5]. AOPLA is an approach for the design of product line architectures that follows a concern-oriented approach. So far, AOPLA does not include a strong formalism for modeling variability. Due to this, variability management and product derivation are hard to accomplish. The domain analysis and modeling phase of the SPL development has already been finished and the plug-in has been used for modeling variability found in feature, entity-relationship and functional models.

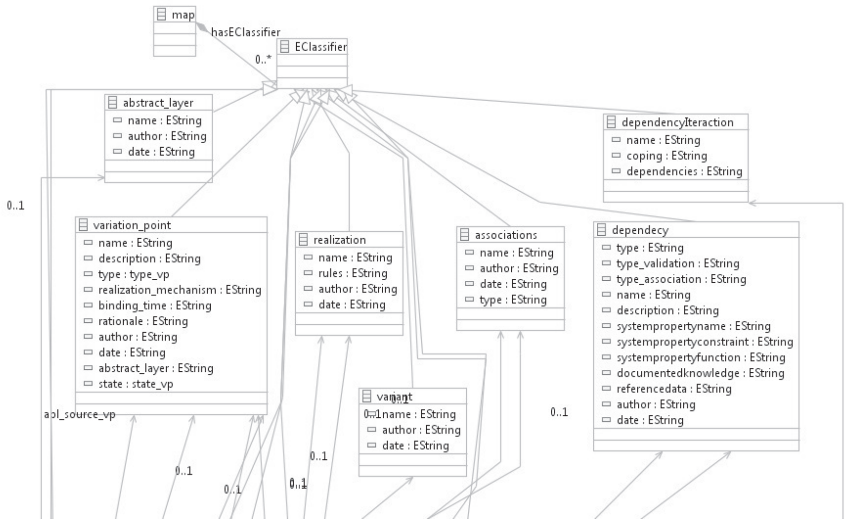


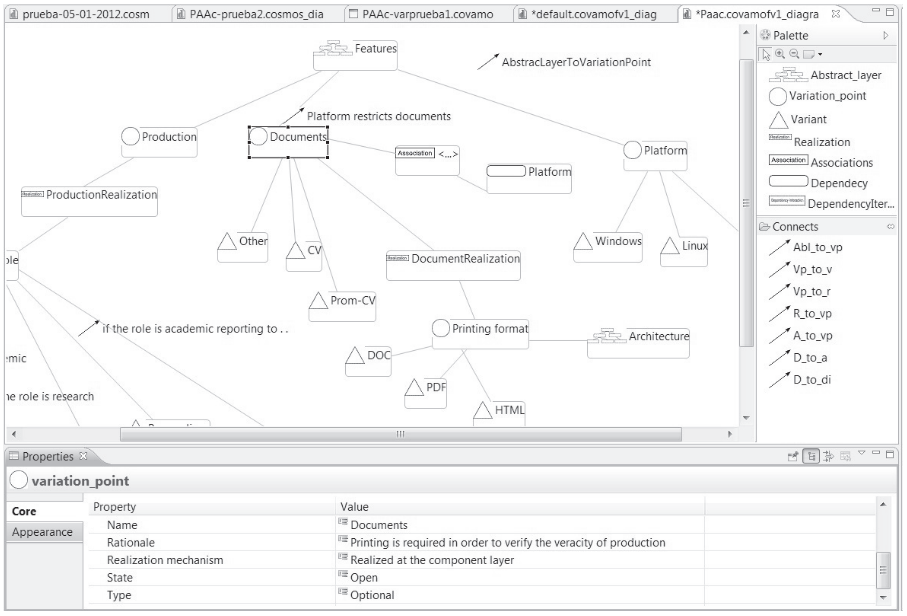
Figure 4 . COVAMOF *Ecore l* model.

The properties shown at the bottom of the figure correspond to properties of the Documents variation point. Another important element of the COVAMOF framework that can be identified in fig. 5: a dependency called Platform. Such a dependency is associated (by means of an Association element) to the Documents variation point and it restricts the selection of the Document's variants depending on the selection of the Platform variation point.

In addition to variation points and variants, two abstraction layers can be noted: features and architecture (a feature is a high level requirement and it is the first formalism for modeling variability during domain analysis and modeling of a SPL). It can also be seen that the Documents variation point in the features layer is realized (by means of a realization relationship named DocumentRealization) by the Printing format variation point in the architecture layer. The Documents variation point encompasses variants: CV, Prom-CV and Other.

The architecture modeling phase for the product line is currently being developed and the results so far indicate the following improvements to the plug-in:

- The differentiation of the types of variation points (as discussed in Section I) by means of a different notation since the arrows and labels employed are not meaningful enough for the users.



variation_point

Property	Value
Name	Documents
Rationale	Printing is required in order to verify the veracity of production
Realization mechanism	Realized at the component layer
State	Open
Type	Optional

Figure 5 . The variability model of the tool.

- The inclusion of 'author', 'date of creation' and 'date of modification' properties for variation points, variants, and dependencies.
- The inclusion of a 'view mode'. That is to say, to give the user the choice of viewing only variation points within the CVV or only dependencies within the CVV.
- The consideration of concerns traceability. AOPLA includes a concern model and it is necessary to achieve traceability of concerns during the process. So far, he inclusion of a 'concern' and 'type of concern' properties for variation points and variants, has been detected as an improvement. However, it is expected to define more support between the concern and variability models as the process develops.

Since the architecture modeling phase of the SPL is still being developed it is expected to find more improvements for the plug-in. As for derivation, the dependencies modeled in the CVV support the derivation process. Dependencies are relationships between variation point that specify a system property whose value is based on the selection of variants in several variation points [8]. During the derivation process, several restrictions on these properties are specified according to SPL products' requirements.. It is very common that multiple dependencies affect one decision in particular. The consequences of this can be that not all of the requirements of a specific product are satisfied. So far AOPLA encompasses a derivation process that is based on concerns and on the architecture model. Concerns are first identified and modeled the domain engineering phase. The concerns model is continuously updated during architecture modeling so that it includes the traceability and mapping into modules and components. However, the dependencies definition and their considerations is entirely left to the architect's knowledge. As a result of this, the derivation process is prone to errors. COVAMOF will therefore alleviate this situation.

CONCLUSIONS

Variability modeling is a key issue for a Software Product Line Engineering approach. A Software Product Line (SPL) takes economic advantage of commonality and variability that already exist within the products. COVAMOF is a framework for variability modeling that allows for modeling variation points, variants and dependencies as first class elements within all of the abstraction levels of a SPL. COVAMOF also encompasses support for the derivation of products as well as for the evolution of the SPL.

This paper presented the development of a Eclipse plug-in for COVAMOF. At the present time, this plug-in encompasses the modeling of variability that corresponds to the Variation Point and Dependency views. The plug-in has been tested with several examples provided in the literature [2] [3] [5] [6].

Currently, the plug-in is being used in the development of a SPL for a tool for academic activities and products whose product line architecture is being developed with AOPLA [5]. The main motivation for the development of the plug-in for COVAMOF was to provide variability modeling support for AOPLA. Although the development of the product line architecture and the SPL have not been finished yet, some important improvements have been detected. Furthermore, important contributions between AOPLA and COVAMOF have been found. This contributions consist mainly in supporting the concern-oriented approach of AOPLA: concern traceability within the variability model and concern-oriented product derivation. A second version of the plug-in is already being developed. The academic activities and products product line will serve to test the new version of the plug-in.

Future work for the plug-in consists in adding COVAMOF derivation process. An important contribution in such a derivation process is that it should be concern-oriented, in order to comply to AOPLA focus on concerns. An additional future work is to include the COVAMOF support for evolution. As has already been mentioned such a support is a method for evaluating the variability needed for including a new product within the SPL. The inclusion of such a method should within AOPLA will help in the development of the variability view in AOPLA.

REFERENCES

[1] Clements, P., Northrop, L. (2001). *Software Product Lines: Practices and Patterns.* Boston: Addison-Wesley.

[2] Pohl, K., Böckle, G., van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques.* Berlin Heidelberg New York: Springer-Verlag.

[3] van Gurp, J., Svahnberg, M., Bosch, J. (2001). On the Notion of Variability in Software Product Lines. *Proc. Of the Working IEEE/IFIP Conference on Software Architecture (WICSA '01),* IEEE Computer Society, pp. 45-54.

[4] Parnas, D. (1976). On the Design and Development of Program Families, *IEEE Transactions on Software Engineering,* 2:1-9.

[5] Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J. (2004). COVAMOF: A Framework for Modeling variability in Software Product Families. *Proc. of the 3rd. Software Product Line Conference (SPLC 2004),* Springer-Verlag, pp. 197-213_12.

[6] Sinnema, M., de Graaf, O., Bosch, J. (2004). Tool Support for COVAMOF, *Proc. of the International Workshop on Software Variability Mangement for Product Derivation - Towards Tool Support,* pp. 12.

[7] Bass, L., Clements, P., Kazman, R. (2003). *Software Architecture in Practice.* Boston: Addison-Wesley.

[8] *The Eclipse Foundation* (November 2011). http://www.eclipse.org/ .