Adding variability management to UML-based software product lines[1]

Edson Alves de Oliveira Junior [a], Itana M. S. Gimenes [a,*], Elisa H. M. Huzita [a], José C. Maldonado [b], Paulo Alencar [c]

[a] Departamento de Informática, Universidade Estadual de Maringá (UEM), Maringá, Paraná, Brazil

[b] Departamento de Ciências de Computação e Estatística, Universidade de São Paulo (USP), São Carlos, São Paulo, Brazil

[c] Computer System Group, School of Computer Science, University of Waterloo, Waterloo, Canada

Abstract

The software product line (PL) approach promotes the generation of specific products from a set of core assets for a given domain. This approach is applicable to domains in which products have well-defined commonalities and variation points. Variability management is concerned with the management of the differences between products throughout the PL lifecycle. This paper presents a UML-based process for variability management that allows identification, representation and delimitation of variabilities; and, identification of mechanisms for variability implementation. The evaluation of the process was carried out as a case study within the context of an existing PL for the Workflow Management System (WfMS) domain. The results have shown that the proposed process includes well-defined control mechanisms that increase the possibility of identifying and tracing variabilities.

Keywords: variability management, software product line, component-based development, software architecture.

[1] This paper in an enhancement of a previous paper as referred to in (Oliveira Junior et al., 2005).

1. Introduction

The software product line (PL) approach aims at promoting the generation of specific products of a domain that forms a product family. The approach is based on the reuse of a well-defined infrastructure, called the *core asset* (SEI, 2005).

The benefits obtained with a PL approach include (SEI, 2005; Heymans and Trigaux, 2003): better understanding of the domains, more artifact reuse, and less time to market. Practical evidence of these benefits can be seen in organization reports such as Nokia (Bass et al., 2003) and Bosch GmbH (Steger et al., 2004). As a result, there are several efforts; both academic and industrial, to reduce the difficulties in the adoption of the PL approach (van Der Linden, 2002).

According to the Product Line Practice of the Software Engineering Institute (PLP/SEI) (SEI, 2005) there are three main processes in the product line context: core asset development (domain engineering), product development (application engineering) and management.

The core asset, generated in the core asset development activity, is the main part of the PL; it contains the architecture of the PL, and its components are represented in a way that makes clear the common and variable aspects of the potential products of the PL. The simplicity of producing products from a PL depends on how well its core asset is designed. The more generic are the artifacts of the core asset, the more products can be generated from a PL. This generality requires the postponing of design decisions (SEI, 2005; van Gurp et al., 2001) that allow the discrimination of products. These design decision issues are treated as *variabilities*.

Variability management is one the main activities of the PL management. It is recognized as an important issue for the success of PLs. However, each existing solution is applied to only specific PL approaches (Heymans and Trigaux, 2003). Thus, there is a lack of an overall reasoning about variability management.

This paper presents a process for variability management which includes tasks for identification, representation, and delimitation of variabilities; and identification of variability implementation mechanisms. The process supports the whole PL lifecycle. In addition, it supports variability tracing and analysis of the configuration of specific products.

The process was conceived within the context of an existing component-based product line for workflow management systems (WfMS) (Gimenes et al., 2003; WfMC, 2005). The evaluation of the process was carried out as a case study which followed the concepts of experimental software engineering.

This paper is organized as follows: Section 2 presents a discussion about important issues of variability management; Section 3 describes the PL for WfMS; Section 4 presents the proposed process; Sections 5 and 6 present lessons learned and related work. Finally, Section 6 presents the conclusions.

2. Variability Management in Software Product Line

This section introduces the main concepts and issues involved in variability management.

Variability is the general term used to refer to the variable aspects of the products of a PL. It is described through variation points and variants. A variation point is the specific place in a PL artifact to which a design decision is connected. Each variation point is associated with a set of variants that corresponds to design alternatives to resolve the variability (Heymans and Trigaux, 2003).

Variability management includes issues such as: (i) variability identification and representation; (ii) variability binding, and (iii) variability control.

According to van Gurp et al. (2001), variability can be initially identified based on the concept of feature. They define a feature as a logical unit of behavior that corresponds to a set of functional and quality requirements. This concept comes from domain engineering (Kang, 1990) and has been constantly improved to fit the demands of the product line approach (Simons et al., 1996; van Gurp et al., 2001; Sochos et al., 2004; Czarnecki et al., 2005).

*
Corresponding author. Address: Departmento de Informática, Universidade Estadual de Maringá, Av. Colombo 5790, 87020-900, Maringá-PR, Brazil. Tel.: +55 44 3261-4324; fax: +55 44 3263-5874.
e-mail address: itana@din.uem.br (Itana Gimenes).

Features are usually represented through feature diagrams which are decorated trees that contain the features identified for a system family (Sochos et al., 2004; Czarnecki et al., 2005). Important attributes usually represented in a feature diagram are: the relationship between features and variation points, the relationship amongst features, and the feature binding time.

There are PL approaches that are purely based on the feature model and Domain Specific Languages (DSL) (Czarnecki and Eisenecker, 2000; Batory, 2004). Another important set of PL approaches maps the features to UML models (Atkinson et al., 2001; ClauB, 2001a; Gomaa and Weber, 2004) and provides UML extensions to represent variability along the PL lifecycle. The variability management process proposed in this paper is based on the latter approaches.

Jacobson et al. (1997) propose the currently most popular notation to represent variability in use cases. A blob ("•") is associated to the use case indicating that there are variable aspects associated with it. However, variabilities are not treated at the actor level. Morisio et al. (2000) introduce a notation for class diagram; in this case, the stereotypes <<V>> and <<xorV>> are associated to classes and methods. In our work, variability is represented by introducing stereotypes to the UML models: use case, class, and component. In addition, UML notes are used to represent information regarding variation points and variants.

The variability binding time indicates the PL lifecycle milestone in which zero or more of the variants associated with the variation point will be chosen depending on the kind of the relationship between the variants and the variation point (van Gurp and Bosch, 2000). Variability binding constrains also the choice of implementation mechanisms (Fritsch et al., 2002). We adopt the classification of binding time proposed by Anastasopoulos (2000) which is: (i) compile time – the variability is resolved either before the program is compiled or at compile time; (ii) link time – the variability is resolved during module or library linking, by selecting different library with different versions of exported operations; (iii) runtime – the variability is resolved during program execution; (iv) update time or post runtime – the variability is resolved during the program updates or after its execution.

According to Fritsch et al. (2002) and Becker (2003), variability management is related to every activity of a PL core asset development process. van Gurp and Bosch (2001) suggest that the variability management process is composed of the following activities:

- *Variability identification*: consists of identifying the product differences and their location within the PL artifacts.
- *Variability delimitation*: define the binding time and multiplicity.
- *Variability implementation*: is the selection of implementation mechanisms.
- *Variant management*: controls the variants and variation points.

These activities were used as a basis to conceive our process.

3. The PL for WfMS

This section presents the PL for WfMS used as a basis to design the proposed process for variability management.

The design of the product line for WfMS (Gimenes et al., 2003) was mainly based on the Catalysis method (D'Souza and Wills, 1999). Catalysis was used, as it is a general purpose component-based development method, based on UML (Rumbaugh et al., 1999) that encompasses important concepts such as the central role of software architecture, frameworks and patterns. The fact that we represent all artifacts of the PL in UML makes it easier for designers of traditional approaches to understand our specification and also enable us to take profit of current support tools such as IBM Rational Rose (IBM, 2005).

The core asset development process of the PL is composed of the following phases: (i) requirement analysis; (ii) system specification; (iii) architectural design, and (iv) component internal design.

In the requirement analysis, the domain model representing the objects and actions of the domain were developed. The domain analysis was based on the generic architecture and reference models for WfMS of the WfMC (WfMC, 1995). The similarities and variabilities amongst product line members were identified and represented at the level of use case model following the notation proposed in Jacobson et al. (1997).

In the system specification phase, the analysis of the system's actions led to the identification of the types and related actions. Sequence diagrams were designed for each use case repre-

senting the interactions between objects. The correspondent variation points were represented in the system specification phase based on the notation proposed in Morisio et al. (2000) which extends UML with a variability stereotype.

From the system specification several refinements were made to reach the level of architecture of components. The product line architecture is presented in Figure 1.

The components of the architecture and their respective variation points are described as follows:

- `GraphicalInterfaceMgr`: responsible for user interface management. One variation point is the user interface being via web browser or conventional.
- `WorkflowArchitectureMgr`: supports the definition and maintenance of workflow architectures. This component makes the workflow definition more flexible as the workflow types are not static. Variation points include the resource type, the tool type, and the process language supported. Resource can be specialized into actor, tool and material types. Tool type can be either internal or external. Different process programming languages can be supported depending on the interpreter.
- `WorkflowMgr`: responsible for the instantiation and management of projects that are associated with a workflow. A project includes an instantiation of workflow architecture. For each workflow element in the architecture there is an object in the workflow instance. No variation points were defined for this component.
- `WorkflowExecutionMgr`: responsible for the control and management of workflows. The main variation point in this component is the possibility of executing different scheduling algorithms.
- `TaskScheduler`: responsible for the scheduling of tasks. It allows the interaction between the users and the tasks. Variation points include resources to be used: types of resources and tools to be used (external or internal).
- `ResourceAllocationMgr`: responsible for resource allocation (e.g. actors, tools and material). In addition to the resource type and tool type, variation points include resource allocation policies.
- `ExternalApplicationMgr`: responsible for the management of external applications during the workflow definition and task execution. Variation points include different mechanisms to adapt external applications to the workflow.
- `ObjectMgr`: responsible for the object management support. It maintains workflow data such as: control data, information data and even a whole workflow. All other components of the architecture use its services. This makes the architecture independent from the object management system. Variation points include adapters for different database management systems.
- `Interpreter`: responsible for the execution of a workflow script written in a process programming language (WfMC, 1995).

The specification of invariants, preconditions and post-conditions for the components was also carried out based on OCL (OMG, 2005). The product line architecture was evaluated based on Rapide and its support tools (Computer Science Lab, 1997).

Currently we are populating the architecture by either developing novel components or reengineering components developed previously not following the PL approach. Each component internal design also follows the Catalysis method.

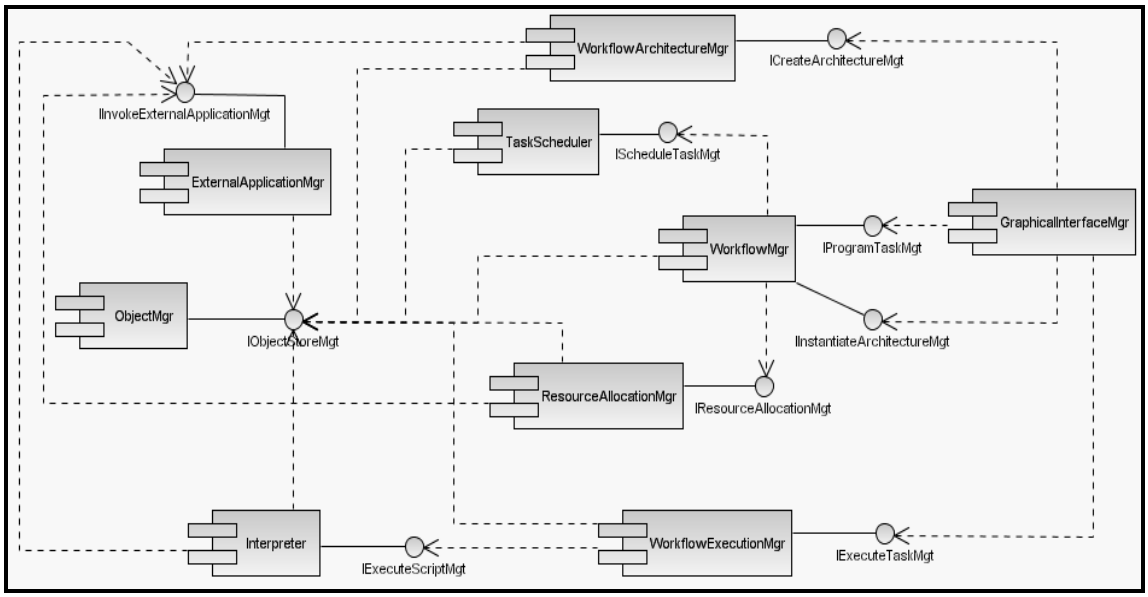

Fig. 1. Architecture of the PL for WfMS.

As far as we progress in their design, the architecture is revaluated. The implementation of the core product line components was carried out based on open source tools which include: Java, the Swing (Java 2) toolkit (Sun, 2005) and the JHotDraw framework (JHotDraw, 2005); CORBA JacORB (JacORB, 2005), an Object Request Broker (ORB) implemented in Java that has many functionalities not identified in similar products; and, MySQL (MySQL, 2005) together with the ObjectBridge framework (Apache, 2005).

4. The Variability Management Process

This section presents the overall variability management process (Oliveira Junior et al., 2005), its activities and its relationship with the PL development process.

Figure 2 presents the interaction between the core asset development process, represented by the activities vertically aligned on the left, and the variability management process, represented by the activities defined inside the right rectangle. The variability management process is iterative and incremental, and runs in parallel with the core asset development. After the execution of each activity of the core asset development, the variability management process is executed, thus progressively taking as input the output artifacts of the core asset development. As the activities are executed the number of variabilities tends to increase. Given that the process is iterative, variability updates are allowed from any activity.

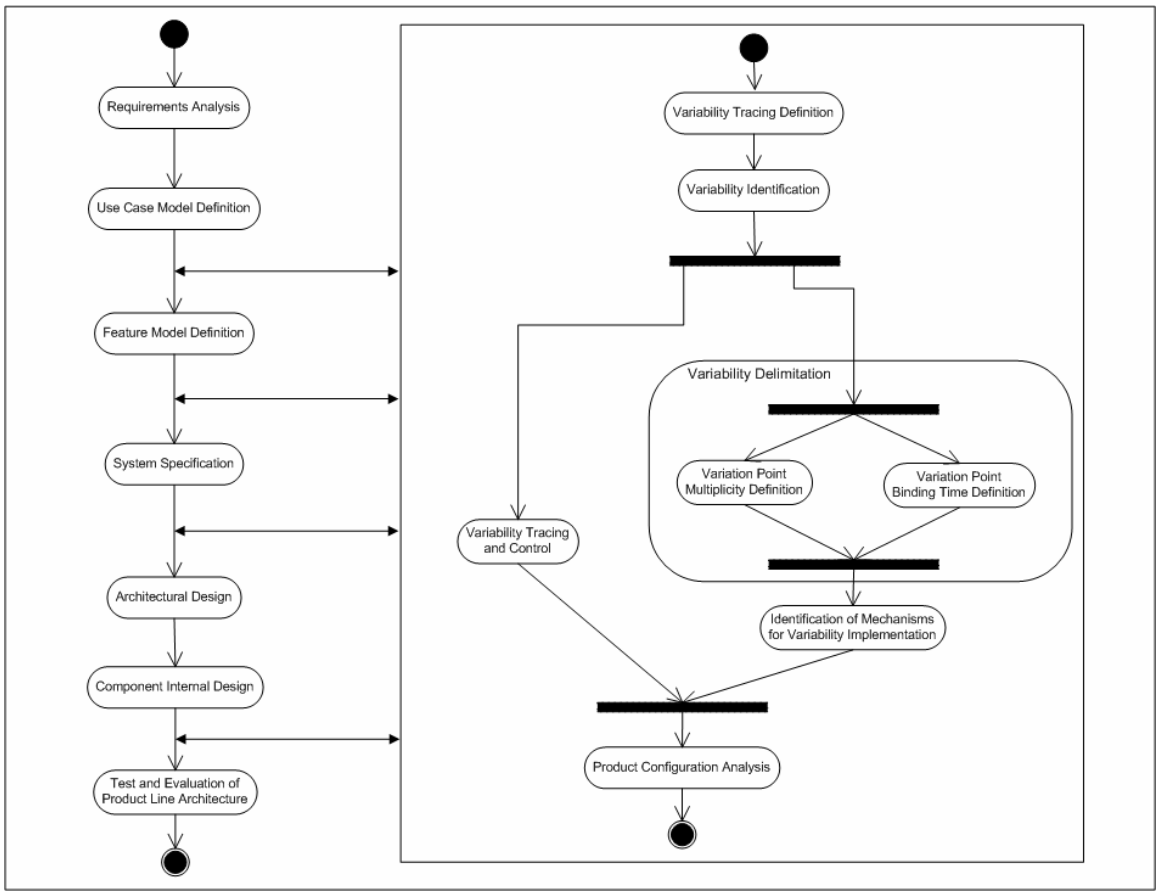

Fig. 2. The variability management process and its interaction with the PL development process.

The activities of the proposed process and their respective input and output artifacts are presented in Table 1. However, note that the input artifacts are made available according to the progress of the core asset development activities. The process is supported by a metadata model which describes the relationships among the PL artifacts. This model is described in Section 4.6.

The process consumes artifacts from the PL core asset as well as producing information for it. An example is the use case model and the static type model. They feed the variability management process and return to the core asset with the variabilities identified and limited. However, there are models, such as the variability tracing and implementation models that are originated in the variability management process.

Activity	Input Artifacts	Output Artifacts
Variability tracing definition	Use case and feature models	Variability tracing model
Variability identification	Use case, static type, feature and component models	Use case, static type, feature and component models with variabilities identified
Variability delimitation	Use case, static type, feature and component models with variabilities identified	Use case, static type, feature and component models with variabilities limited
Identification of mechanisms for variability implementation	Static type and component models with variabilities limited	Variability implementation model

Tab. 1. Input and output artifacts of the variability management process activities.

The following subsections provide a summary of the case study undertaken to evaluate the proposed process and describe the process activities.

4.1 Case Study

The case study was conceived based on the concepts of experimental software engineering and the evaluation of software engineering methods and tools (Kitchenham et al., 1995; Pfleeger, 1995; Kitchenham, 1996). An evaluation plan was developed consisting of the following activities: identification of the case study context, definition of hypotheses, selection of the pilot project, identification of comparison methods, planning of the case study, execution of the case study and analysis of results.

The context of the case study was the PL for WfMS described in Section 3. Its objective was to re-specify the existing product line with the introduction of the proposed variability management process in order to observe the impact on the accuracy of variability identification and support for variability tracing. Thus, the PL development activities were carried out interacting with the variability management activities, as established in the proposed process. The number of variabilities identified in the following models was measured: use case model, static type model, and component model. These numbers were compared to the number of variabilities identified in the previous process.

The results confirmed that there was an improvement on the variability representation after the introduction of the variability management process. This led to a more explicit representation of the potential products that can be developed from the PL.

As support tool we used IBM/Rational Rose and Requisite Pro (IBM, 2005).

The activities described in the following sections are illustrated with an excerpt of the case study. For space reasons, the activities of identification and delimitation of variabilities are illustrated through the use case model. The representation of variabilities for other artifacts is similar.

4.2 Variability Tracing Definition

Variability tracing definition receives as input the use case model and the feature model built in the PL development process. The variability tracing model is built as follows:

- the captured features of the PL are listed;
- the captured use cases are listed;
- the relationship between features and use cases are re-analyzed;
- crossing relations between use cases and features are marked with a blob.

This model is represented in tabular form. An example of a row in the model is for the feature `User Communication`. It is related to the use cases (see Figure 3): `Communicate with Users`, `Communicate Via E-mail`, `Communicate Via Chat` and `Communicate Via Teleconference`, so the corresponding columns of the table are marked. The model will support the tracing from the features to all UML models that are related to variable aspects throughout the lifecycle. Thus, if a certain feature needs to be removed from a given product, it is necessary to know the impact of the removal over related artifacts. The tracing is possible because the features are related to the use case model that is related to the static type model and the component model. For instance, if the feature `User Communication` is not selected for a certain product, the related use cases, static type and component models have to be updated to reflect this.

The variability tracing model is based on the representation used for feature and use case tracing in the tool IBM/Rational Requisite Pro (IBM, 2005). This tool allows indicating which use cases are related to the features and vice-versa.

4.3 Variability Identification

Variability identification receives as input the use case, feature, static type, and component models from the development activity in each interaction between the PL development and the variability management process. It aims at progressively identifying the variability associated with the models.

Table 2 presents the stereotypes used to represent variability on the artifacts of the PL. For each model there are two columns. The first indicates the UML relation used to represent the

type of variability between the variation point and its variants. The second indicates the stereotype used to indicate variability.

	Use Case Diagram		Class Diagram		Component Diagram	
	UML Relation	Element Stereotype	UML Relation	Element Stereotype	UML Relation	Element Stereotype
Variation Point	--------	<<variationPoint>>	--------	<<variationPoint>>	Dependency	<<variable>>
Mandatory Variant	Association Aggregation Composition Dependency	<<mandatory>>	Association Aggregation Composition Dependency	<<mandatory>>	Dependency	<<mandatory>>
Optional Variant	Association Aggregation Composition Dependency	<<optional>>	Association Aggregation Composition Dependency	<<optional>>	Dependency	<<optional>>
Inclusive Alternative Variant	Spec./ General. with the stereotype <<extend>>	<<alternative_OR>>	Inheritance	<<alternative_OR>>	Dependency	<<alternative_OR>>
Exclusive Alternative Variant	Spec./ General. with the stereotype <<extend>>	<<alternative_XOR>>	Inheritance	<<alternative_XOR>>	Dependency	<<alternative_XOR>>
Mutually Exclusive Variant	Dependency	<<mutex>>	Dependency	<<mutex>>	Dependency	<<mutex>>
Inclusive Variant	Dependency	<<requires>>	Dependency	<<requires>>	Dependency	<<requires>>

Tab. 2. UML relations and stereotypes of the graphical representation of the variation points and variants.

The stereotype `<<variationPoint>>` indicates that a model element represents a variation point and has associated variants. These variants can be represented with one of the following stereotypes: `<<mandatory>>`, indicating a compulsory variant; `<<optional>>`, indicating a variant that need not be chosen; `<<alternative_OR>>`, indicating that one or more variants can be chosen; and, `<<alternative_XOR>>`, indicating that only one of the variants can be chosen.

The stereotypes `<<requires>>` and `<<mutex>>` are used to indicate dependency relationship between variants. The former indicates that once a source variant is chosen, also the target variants have to be chosen. In contrast, the latter indicates that once a source variant is chosen, the target variants cannot be chosen.

UML notes are also used to support graphical representation of variabilities. A UML note has essential information that allows answering questions such as:

- Which variants are related to a certain variation point?
- What is the binding time of the variants associated to a variation point?
- Is it possible to add new variants to the variant set associated with a given variation point?

The UML note associated with a variation point defines:

- the type of the relationship between the variation point and its variants, which are: {} indicating a mandatory or optional relation, {or} indicating an inclusive relation and {xor} indicating an exclusive relation;
- the name of the variation point;
- the multiplicity of the variation point, indicating the minimum number of variants to be chosen to resolve such variation point;
- the binding time of the variation point; and,
- true or false indicating whether the variation point supports the addition of new variants to its associated set.

Figure 3 presents an example of variability identification in a use case model. In this figure, the UML notes has three "?" because the variabilities were not limited yet.

Two variabilities were identified in this figure. One is concerned with different forms of workflow execution, and the other is related to the kind of communication between workflow users. The former is represented by the use case `Execute Workflow`. The associated variants include `Execute Workflow with Priority Control` and `Execute Workflow Serial`. The kind of variability relationship is `<<alternative_XOR>>` because only one of the execution algorithms can be selected. The latter is represented by the use case `Communicate With Users`. The associated variants are `Communicate Via Chat`, `Communicate Via E-mail` and `Communicate Via Teleconference`. The kind of variability relationship is `<<alternative_OR>>` as more than one kind of communication can be used.

Variability identification is a domain dependent activity which requires abilities of the PL managers and analysts. However, some guidelines may be offered, such as:

- elements of the use case model related with the stereotype `<<extend>>` or elements of the static type model related by inheritance suggest variation points with associated variants which are inclusive or exclusive alternative. For instance, in Figure 3, the use cases `Execute Workflow with Priority Control` and `Execute Workflow Serial` represent exclusive alternatives associated with the variation point `Execute Workflow`.
- elements of the artifacts related to the stereotype `<<include>>` in the use case model or association in the static type model suggest either mandatory or optional variation points. For instance, in Figure 3, the use case `Execute Script` represents a compulsory variant.

4.4 Variability Delimitation

Variability delimitation aims at defining the following attributes of a variation point: (i) multiplicity; (ii) binding time, and (iii) possibility, or not, of adding new elements to the associated variant set.

The multiplicity of a variation point indicates the minimum number of elements of the associated variant set that must be chosen to resolve the variability. The following rules are applied:

- a variation point with relation type optional has multiplicity 0 (zero), indicating that a variant can be chosen or not;
- a variation point with relation type mandatory has multiplicity 1 (one), indicating that the associated variant must be selected;
- a variation point with relation type exclusive alternative has multiplicity 1 (one), indicating that only one element of the possible set of variants must be selected; and,
- a variation point with relation type inclusive alternative has multiplicity ranging from 0 (zero) to the maximum number of variants associated with the variation point, indicating that any number of variants of this interval can be chosen.

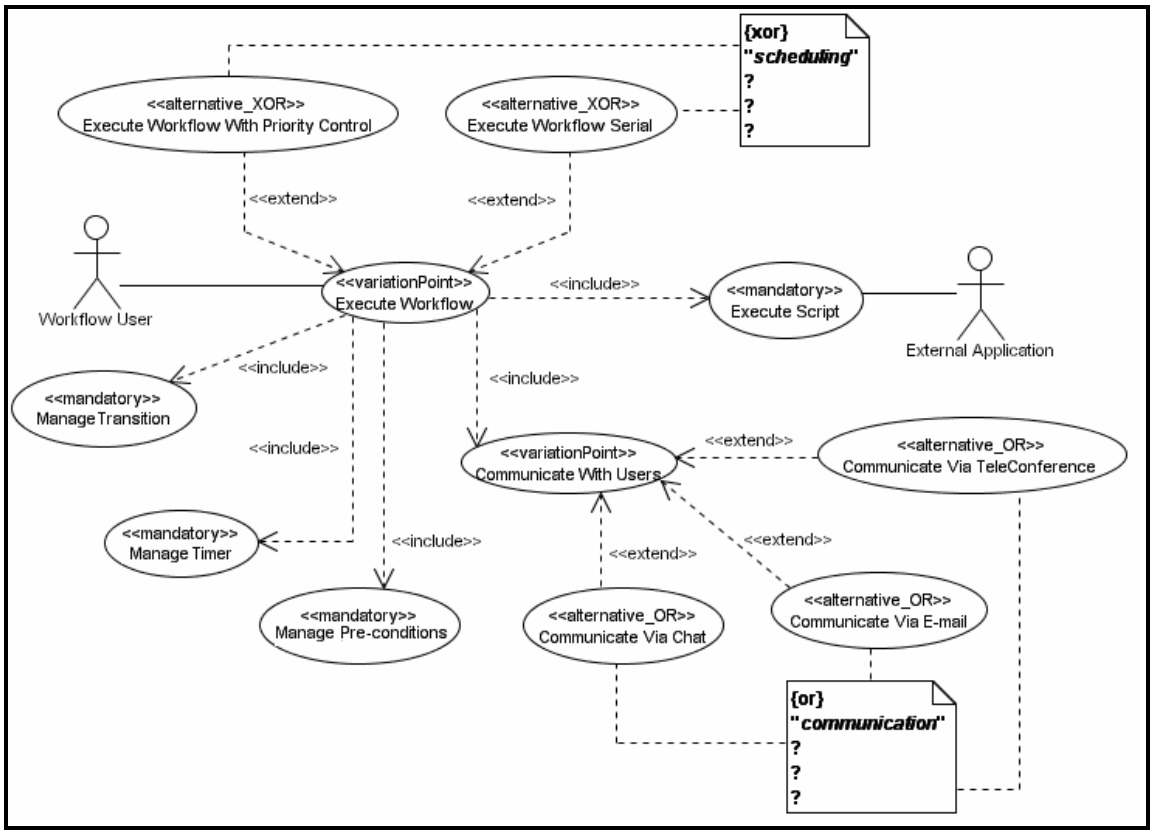

Fig. 3. Variability identification in a use case model.

We have extended the Anastasopoulos and Gracek (2001) binding time classification by adding the *design* and *implementation* steps. Thus, the binding time of a variation point were defined as follows:

- *Design*: the variation point is resolved during the specification of the PL or of its products.
- *Implementation*: the variation point is resolved at programming time, before compilation.
- *Compiling*: the variation point is resolved during the compilation process.
- *Linking*: the variation point is resolved during the module or library linkage process.
- *Runtime*: the variation point is resolved at the PL product execution time.
- *Updating*: the variation point is resolved during the PL product update. An example is the inclusion of a new module in a PL product when it is already running.

The definition of the binding time is essential to determine the choice of implementation mechanisms, which are described in the next section. Figure 4 presents an example of variability delimitation in a use case model, for the variation points `Execute Workflow` and `Communicate With Users`, identified in Figure 3. The variability `"scheduling"` has multiplicity 1 as it requires the selection of only one of the variants: `Execute Workflow with Priority Control` or `Execute Workflow Serial`. The fourth line indicates that the binding time is `"compiling"` whereas the fifth line is set to false to indicate that addition of new variants is not allowed. The note with the variability `"communication"` indicates that 0 or more variants can be selected; the binding time is `"runtime"`, and the addition of new variants is not allowed.

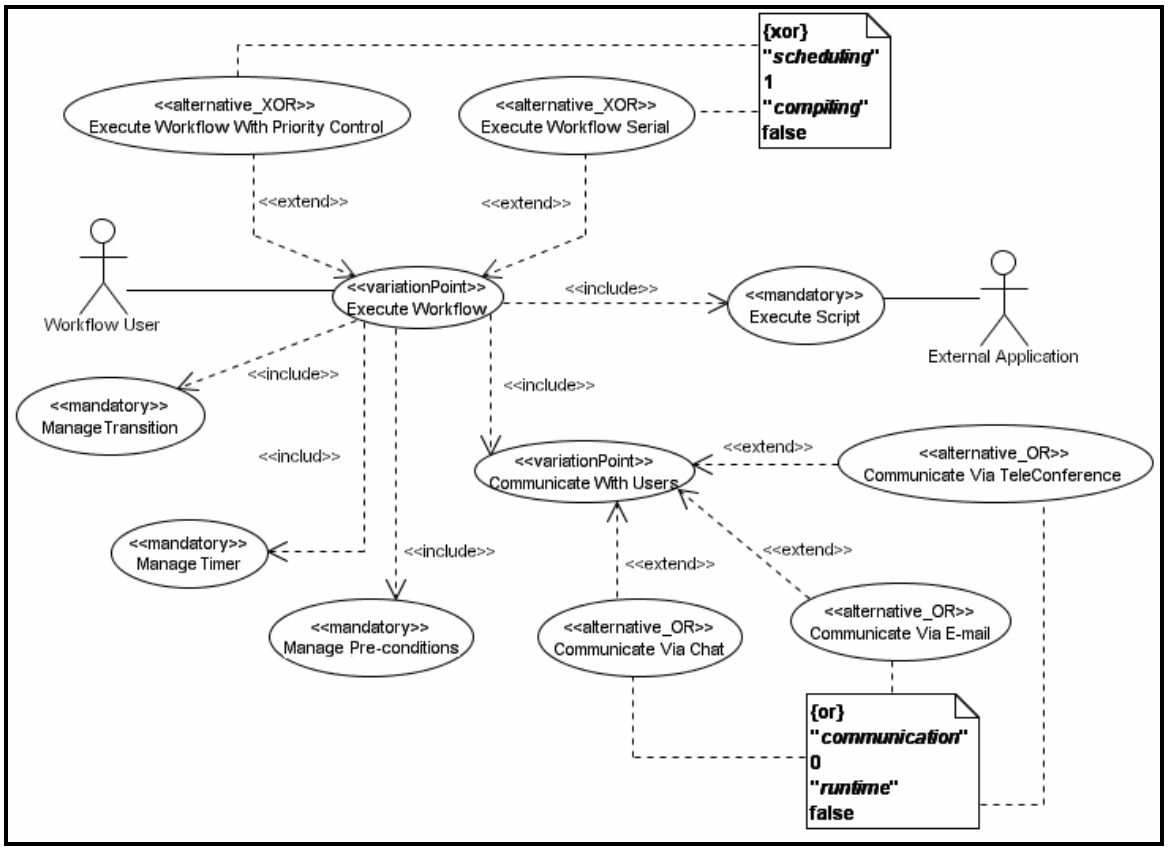

Fig. 4. Representation of multiplicity and binding time of variation points in use case models.

4.5 Identification of Mechanisms for Variability Implementation

Identification of mechanisms for variability implementation aims at selecting the mechanisms to be used to implement the variabilities. The input artifacts are static type and component models with their respective variabilities represented and limited, as a result of the previous activity.

The output artifact of this activity is an implementation model which is represented as a table. Each row of the table indicates the name of a variability, the artifact element in which it occurs, the binding time, the implementation mechanism and the implementation strategy. The model was built based on the variability implementation techniques proposed by Svahnberg et al. (2002), Jacobson (1997), and Anastasopoulos and Gracek (2001). These techniques include inheritance, extension and parameterization.

The implementation mechanism and strategy are chosen based on the binding time and the class or component in which it occurs.

As an example, Figure 5 presents the static type model for the WfMS PL. The class `TypeTool` represents a variation point and the classes `TypeInternalTool` and `TypeExternalTool` are the associated variants. The variability `"tool class type"` occurs at the class level and it is bound at compiling time. Thus, a possible implementation mechanism (Svahnberg et al., 2002) is `"Code Fragment Superposition"`. The strategy `"override generic source code with specific one using aspect-oriented programming"` guides the variability implementation.

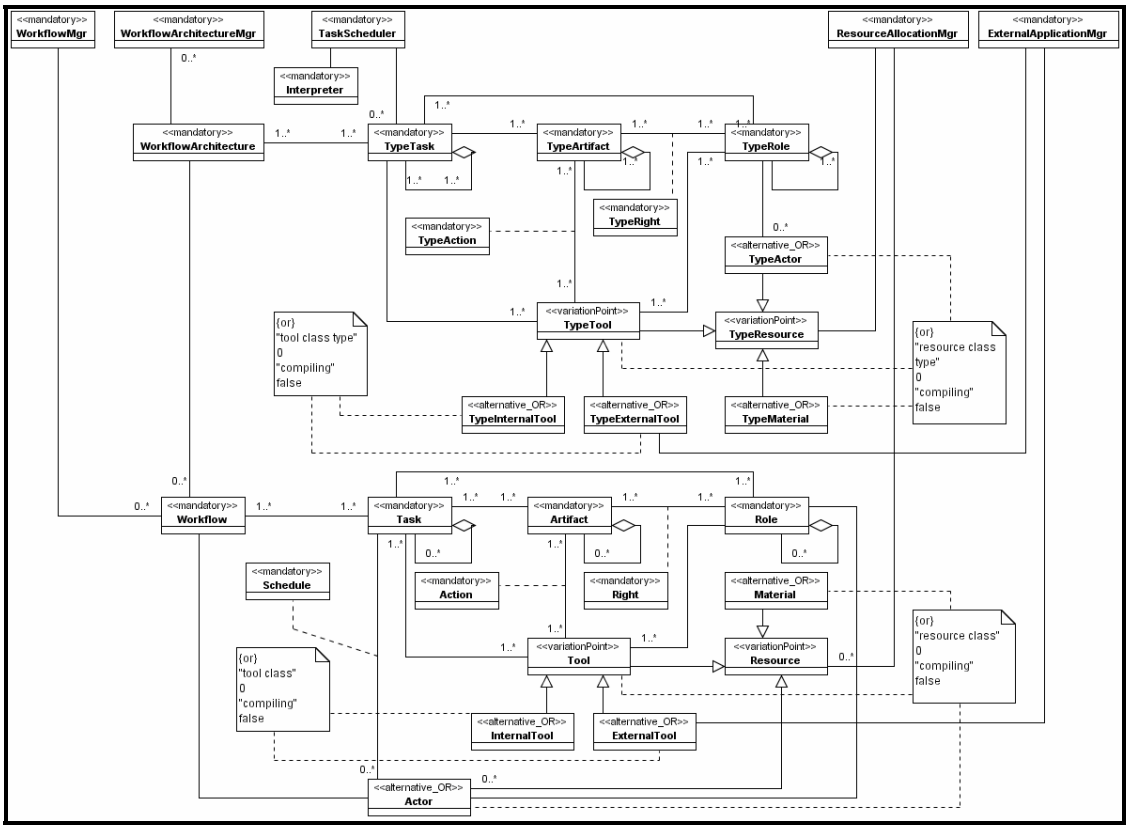

Fig. 5. Static Type Model for WfMS.

4.6 Variability Tracing and Control

Variability tracing and control aims at tracing and controlling variabilities. It is supported by the process metadata model, presented in Figure 6. This model describes the relationships among the PL artifacts. It was conceived based on the generic variability model proposed by Becker (2003) to support variability management tools.

The metadata model defines the relationships between variant artifacts of a PL and their variation points, variants, binding time, and implementation mechanisms. This metadata model together with the variability tracing model allows the association of a feature to the related use cases and therefore to the elements of the static type and component model. The execution of this activity consists of the instantiation of the metadata model for the PL.

4.7 Configuration Analysis of Specific Products

Configuration analysis of specific products aims at investigating the impact of selecting possible features of the PL artifacts in order to analyze the feasibility of the production of a specific product which is a PL member. The selection of a feature may imply the selection of a product configuration according to the variation points described. Moreover, if the product requires an additional feature there has to be an analysis of the introduction of this feature in the PL artifacts.

The fact that the metadata model makes explicit the relationships among artifacts enables the execution of algorithms of analysis to inspect the PL artifacts. However, we still lack a tool that makes this analysis possible.

The completion of this activity represents the end of an iteration of the variability management process, and thus a return to the core asset development activity which triggered the beginning of the iteration.

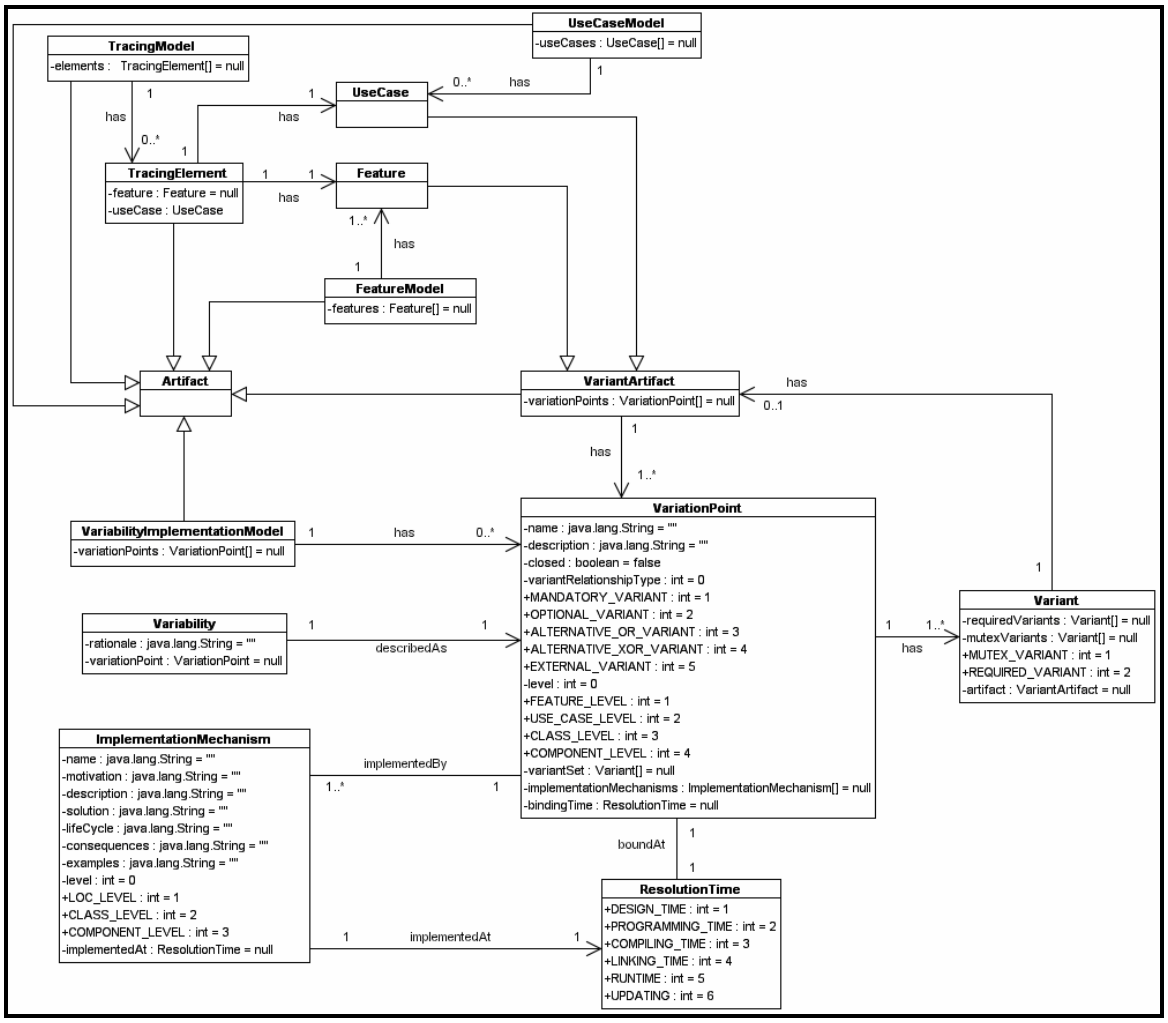

Fig. 6. Metadata model to support the variability management process.

5. Lessons Learned

In this section, we present the lessons learned from the development and the exercising of the variability management process.

Variability Management Process: the key issue of a PL is the way it articulates and manages its variable aspects. A variability management process must coexist with any PL core asset development and product development in order to support the clear specification, tracing, and control of variabilities. Our studies provided evidence that the variability management process enables better representation, and therefore better control of variabilities, compared to the existing PL. Moreover, it makes explicit the most important decisions, such as the number of variants associated with a variation point and the type of choices allowed; the binding time; and the implementation mechanisms.

Empirical Evaluation of a PL: the evaluation of the proposed variability management process was carried out based on an empirical study in the form of a quantitative case study (Kitchenham et al., 1995; Kitchenham, 1996) which enabled the demonstration of the need of a variability management process. The case study controlled variables such as: (i) the number of artifacts in each activity of the process; (ii) number of variabilities in each artifact, given by the sum of variable elements in each artifact; and, (iii) variability types associated with each varia-

tion points. The proposed variability representation allowed a more precise estimation of the number of products that can be derived from a PL, thus offering a better support for PL configuration analysis. This case study together with the results of previous work (Gimenes et al., 2003) made it possible to create an experimental baseline for a PL. However, investigation has to proceed to increase the volume of data available for experiments. On going work aims at building an experimental basis for PL that allows inference of quality attributes of the PL itself and of potential products.

UML Models: the proposed process is based on the UML models. It extends these models by adding stereotypes to represent variability in the feature, use case, static type, and component models. Most of the stereotypes used in our process were inherited from previous work (ClauB, 2001a; ClauB, 2001b; Gomaa and Shin, 2002; Gomaa, 2005). In our process, we introduced the idea of using UML notes to make explicit the multiplicity and binding time associated with variation points. One of the advantages of using notes is that, because the notes belong to the UML meta-models, they can be read from any commercial tool that supports UML modeling.

Feature Modeling: in previous work (Gimenes et al., 2003), we did not use the feature model because we gave priority to modify a traditional software process to reduce the impact of a PL adoption. The feature model is not, as yet, a familiar artifact from the software engineers point of view. However, we introduced the feature model in our variability management process because it proved to be important from both the reuse and variability tracing perspectives. Moreover, the introduction of the feature model made our PL approach compatible with most existing approaches (Atkinson, 2001; Gomaa, 2005; PURE-SYSTEMS, 2004). In particular, it is important for a PL to keep a clear relationship between the feature model and the PL's architecture (Sochos et al., 2004).

Metadata Model: the metadata model was an important result of our process design. It provides information about the relationships between artifacts, thus allowing navigation throughout them. Moreover, it forms a basis for building a tool to automate the variability management process. We note that there is still few tools to support the PL approach, e.g. Ménage (van Der Hoek, 2000), Holmes (Succi et al., 2001) e pure:variants (PURE-SYSTEMS, 2004). The proposed metadata model can be represented in XML to facilitate exchange of data with other PL support tools. IBM/Rational Requisite Pro and Rose have shown to be potentially feasible to support variability management. Requisite Pro supports the specification of the relationship between use cases and features. However, it still does not allow graphical representation of the feature model and it is not well integrated with Rose models. Rose allows the manipulation of the UML models to represent variability, and it already provides some mechanisms to support artifact tracing. We believe that the harmonization of tools such as the IBM/Rational suite with PL principles can be fruitful and should facilitate PL adoption.

6. Related Work

The PL variability management process proposed in this paper took into account previous work on PL approaches mainly related to the following issues: management activities, artifact notation, variability attributes, metadata modeling and experimental software engineering.

The management activities defined for the proposed process is inline with the essential activities of the SEI PLP (SEI, 2005). Variability management is considered a subprocess of the PL management activity. Thus, it has a close interaction with the core asset development and the product development activities.

The activities defined in our process were initially based on van Gurp et al. (2001) variability management activities. However, in their work the activities were listed, but not described within the context of a well-defined process with roles, input and output artifacts. Our activities and their associated roles and artifacts are fully specified.

The proposed process uses UML as the notation in which to represent the PL artifacts. It took into consideration similar approaches such as those of Gomaa and Weber (2004), Kobra (Atkinson et al., 2001), and Clauß (2001a, 2001b). These approaches extend the UML notations to represent variability aspects. Our process focused on the use case, feature, static type, and component models because they were considered the most important in variability tracing. However, on going work is evaluating the inclusion

of the representation of variability in sequence and collaboration diagrams. They will allow the analysis of the impact of variability on interactions as proposed in general methods such as Gomaa (2005).

In order to define the activities of the process, it was important to consider previous attempts to define variability attributes such as (van Gurp et al., 2001; van Gurp and Bosch, 2000; Anastasopoulos, 2000; Becker, 2003; Clauß, 2001). On the end, we define the following attributes for variability: type, binding time and implementation mechanisms. Recent work presented by Czarnecki (2005) and Cechticky (2004) contains well-defined feature attributes and representation. However, they are not based on UML.

The metadata model proposed to support our process was based on that of Becker (2003). However, we have eliminated elements that were not compatible with our approach like those which represent the product family such as `ProductFamily` and `FamilyMember`; and those which represent the variability implementation mechanism such as `Selection`, `Generation`, and `Substitution`. Morever, we added the following elements to support the variability tracing over all the PL artifacts: `VariantArtifact`, `FeatureModel`, `Feature`, `UseCaseModel`, `UseCase`, `TracingModel`, and `TracingElement`.

Existing experimental works were used as the basis on which to undertake the case study. In particular, Basili (1986) discusses experiment planning and Kitchenham (1995, 1996) proposes a terminology, steps for an experiment, and how to analyze the results.

7. Conclusions

The variability management process is one of the most important activities of PL development and evolution. It is the variability management that enables the clear identification and tracing of the differences amongst products of a PL.

This paper proposes a variability management process which: (i) defines the activities, artifacts, and roles necessary to control variability in UML-based PL approaches; (ii) makes explicit important decisions, such as the alternatives for variation points, binding times and implementation mechanisms; (iii) introduces UML notes to make explicit the multiplicity and binding time associated with variation points; (iv) allows a clear relationship between features and the PL architecture; and (v) defines a metadata model that forms the basis to design a tool to support variability management. As the variability management process is triggered from the PL development activities, it can be easily adapted to also PL approaches that are based on UML.

The proposed process was evaluated within a case study that compares a PL approach to the same with the introduction of variability management. The quantitative results show that more variability elements were represented throughout the PL artifacts. However, meanwhile the most significant results are qualitative. The experiments show that the variability management process guides the PL developers to better represent and control variability issues. Thus, the process offers support for PL configuration analysis. The case study enabled the definition of a baseline for empirical PL experiments. On going work aims at enhance the volume of data of the experimental basis so that more quantitative measures can be obtained when different PL aspects are introduced.

Further work includes the definition of support to make automated product configuration analysis possible, thus, providing the organizations with mechanisms to evaluate PL adoption and evolution.

References

Anastasopoulos, M.; Gacek, C., Implementing Product Line Variabilities, in: ACM SIGSOFT Software Engineering Notes, New York, v. 26, n. 3, pp. 109-117, May. 2001.

Apache DB Project: ObJectRelationalBridge - http://db.apache.org/ojb - 2005.

Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthing, D.; Paech, B.; Wurst, J.; Zeitel, J. Component-Based Product-Line Engineering with UML. [S.l.]: Addison-Wesley, 2001.

Basili, V. R.; Selby, R. W.; Hutchens, D. H., Experimentation in software engineering. IEEE Transactions on Software Engineering, Piscataway, v. 12, n. 7, p. 733-743, 1986.

Bass, L.; Clements, P.; Kazman, R., Software architecture in practice. 2. ed. Boston: Addison-Wesley, 2003. 560 p.

Batory, D. The Road to Utopia: A Future for Generative Programming, in: Domain Specific Generation, Lengauer et al. (eds.), LNCS 3016, p1-18, 2004.

Becker, M., Towards a general model of variability in product families. in: Proceedings of Software Variability Management Workshop. Portland, USA. 2003. pp. 19-27.

Cechticky, V.; Passetti, A.; Rohlik, O.; Schaufelberger, W., XML-based feature modelling. in: Proceedings of International Conference on Software Reuse. Madrid. pp. 101-114, LNCS 3107, Jul. 2004.

Clauß, M. Generic modeling using UML extensions for variability. in: Proceedings of Workshop on Domain Specific Visual Languages. Tampa Bay, 2001a. p. 11-18.

Clauß, M. Modeling variability with UML. In Proceedings of Young Researches Workshop, 2001, Erfurt, 2001b.

Computer Science Lab. DRAFT Guide to Rapide 1.0 – Language Reference Manuals, Rapide Design Team – Program Analysis and Verification Group. Stanford University, 1997.

Czarnecki, K., Eisenecker, U., Generative Programming. Methods, Tools, and Applications. Addison-Wesley, 2000. 832 p.

Czarnecki, K.; Helsen, S.; Eisenecker, U., Staged configuration through specialization and multi-level configuration of feature models. To appear in special issue on "Software Variability: Process and Management," Software Process Improvement and Practice, 10(2), 2005

D'Souza, D. F.; Wills, A. C. Objects, Components and Frameworks with UML – The Catalysis Approach. Addison Wesley Publishing Company, 1999.

Fritsch, C.; Lehn, A.; Strohm, T., Evaluating variability implementation mechanisms. in: Proceedings of International Workshop on Product Line Engineering Seattle, 2002. p. 59-64.

Gimenes, I. M. S.; Oliveira Junior, E. A.; Lazilha, F. R.; Barroca, L. M. A Product Line Architecture for Workflow Management Systems with Component-based Development, in: 2003 Proc. The IEEE Conference on Information Reuse and Integration, pp. 112-119.

Gomaa, H.; Shin, M. E., Multiple-view meta-modeling of software product lines. in: 8th International Conference on Engineering of Complex Computer Systems (ICECCS 2002), IEEE Computer Society 2002, ISBN 0-7695-1757-9, pp. 238-246, 2002.

Gomaa, H.; Webber, D., Modeling adaptive and evolvable software product lines using the variation point model. in: Proceedings of Hawaii International Conference on System Sciences, Hawaii, 2004. p. 01-10.

Gomaa, H. Designing software product lines with UML: from use cases to pattern-based software architectures. Addison-Wesley, 2005, 736 p.

Heymans, P.; Trigaux, J. C., Software product line: state of the art. Technical report for PLENTY project, Institut d'Informatique FUNDP, Namur, 2003.

IBM Rational Software – http://www-306.ibm.com/software/rational/ - 2005.

Jacobson, I.; Griss, M.; Jonsson, P. Software Reuse – Architecture Process and Organization for Business Success, New York: Addison-Wesley, 1997. 528 p.

JacORB - http://www.jacorb.org - 2005.

JHotDraw - http://www.jhotdraw.org - 2005.

Kang, K. Feature-oriented domain analysis (FODA) - feasibility study. Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh, 1990.

Kitchenham, B.; Pickard, L.; Pfleeger, S. L., Case studies for method and tool evaluation. IEEE Software, v.11, p. 52-62, 1995.

Kitchenham, B., DESMET: a method for evaluating software engineering methods and tools. Technical Report TR96-09, Keele, United Kingdom, 1996. 49 p.

Morisio, M.; Travassos, G. H.; Stark, M. Extending UML to Support Domain Analysis, in: 2000 Proc. IEEE International Conference on Automated Software Engineering, Grenoble, France. pp. 321-324.

MySQL - http://dev.mysql.com - 2005.

Object Management Group. OMG Document: UML 2.0 OCL 2nd Revised submission – http://www.omg.org/cgi-bin/doc?ptc/2003-10-14 - 2005.

Oliveira Junior, E. A., Gimenes, I. M. S., Huzita, E. H. M., Maldonado, J. C. A Variability Management Process for Software Product Lines. To appear in Proc. of CASCON 2005. Toronto, Canada, October 2005.

Pfleeger, S. L., Experimental design and analysis in software engineering – how to set up an experiment. ACM SIGSOFT – software Engineering Notes. v. 20, n. 1, p. 22-26, 1995.

PURE-SYSTEMS - pure-variants: Variant Management – <http://web.pure- systems.com/Variant_Management.49.0.html> - Access: Nov. 2004.

Rumbaugh, J., Jacobson, I., Booch, G. The Unified Modeling Language Reference Manual, Addison-Wesley Pub. Company, 1999.

SEI - Software Engineering Institute. A framework for software product line practice 4.2. Pittsburgh. <http://www.sei.cmu.edu/productlines/framework.html >. Access: June, 01 2005.

Simons, M.; Creps, D.; Klingler, C.; Levine, L.; Allemang, D., Organization domain modeling (ODM) guidebook, version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defence Systems, 1996.

Sochos, P.; Philippow, I.; Riebish, M., Feature-oriented development of software product lines: mapping feature models to the architecture. Springer, LNCS 3263, 2004, p. 138-152.

Succi, G.; Yip, J.; Pedrycz, W. Holmes: an intelligent system to support software product line Development. in: International Conference on Software

Engineering, 23., 2001, Toronto. Proceedings of the International Conference on Software Engineering, 2001, pp.829-832.

Sun Microsystems. Java Technology - http://www.java.sun.com - 2005.

Svahnberg, M.; Van Gurp, J.; Bosch, J., A taxonomy of variability realization techniques. Technical report, Blekinge Institute of Technology, Sweden, 2002.

Steger, M., Tischer, C., Boss, B., Müller, A., Pertler, O., Stolz, W., Ferber, S. Introducing PLA in Bosch Gasoline System: Experiences and Practices. In: Software Product Lines: Third International Conference, SPLC 2004. Nord, R.. (ed.), LNCS 3154, p34-50, 2004.

van der Hoek, A. Capturing product line architectures. in: International Software Architecture Workshop. 4., 2000, Limerick. in: Proceedings of the 4th International Software Architecture Workshop, 2000. pp. 95-99.

van der Linden, F., Software Product families in Europe: The Esaps and Café Projects, IEEE Software, July/August 2002, pp. 41-49.

van Gurp, J.; Bosch, J., Managing variability in software product lines. in: Proceedings of the Landelijk Architectuur Congres. Amsterdam, 2000.

van Gurp, J., Bosch, J., Svahnberg, M. On the notion of variability in software product lines, in: Proc. The Working IEEE/IFIP Conference on Software Architecture (WICSA), Amsterdam, The Netherlands, 2001. pp. 45-54.

Workflow Management Coalition. Workflow Reference Model. Document number TC00-1003, January, 1995.

Workflow Management Coalition (WfMC) – http://www.wfmc.org - 2005

Edson Alves de Oliveira Junior has a bachelor and master degree in computer science from Universidade Estadual de Maringá, Brazil. He is currently taking a doctoral degree in computer science at Universidade de São Paulo (ICMC-USP), Brazil.

Itana M. S. Gimenes is full professor of Software Engineering at Universidade Estadual de Maringá (DIN-UEM), Brazil, and Ph.D. from The University of York, Department of Computer Science, UK. She is currently in a sabbatical license at CSG/SCS/University of Waterloo.

José C. Maldonado is full professor at the Instituto de Ciências Matemáticas e de Computação (ICMC-USP) of the Universidade de São Paulo, Brazil. He is vice-president of the Brazilian Computer Society, and member of the ACM and IEEE.

Elisa H. M. Huzita is associate professor at the Departamento de Informática of the Universidade Estadual de Maringá (DIN-UEM), Brazil. She is member of IEEE and ACM.

Paulo Alencar is a Research Associate Professor in the School of Computer Science at the University of Waterloo. He is member of IEEE and ACM.

Acknowledgements: the authors would like to thank Prof. Daniel Berry for his review of this paper.