On the Role of Features and Goals Models in the Aspect-Oriented Development of Software Product Line*

Lyrene Silva
Departamento de Informática e Matemática Aplicada
Federal University of Rio Grande do Norte
BR 101, CEP 59078-970, Natal-RN, Brazil
lyrene@dimap.ufrn.br

Thais Batista
Departamento de Informática e Matemática Aplicada
Federal University of Rio Grande do Norte
BR 101, CEP 59078-970, Natal-RN, Brazil
thais@dimap.ufrn.br

Sérgio Soares
Centro de Informática
Federal University of Pernambuco
Av. Luis Freire, CEP 50.740-540, Recife-PE, Brazil
scbs@cin.ufpe.br

Lidiane Santos
Departamento de Informática
State University of Rio Grande do Norte
Av. Ayrton Sena, CEP 59.080-100, Natal-RN, Brazil
diane_lid@hotmail.com

Abstract

Requirements of a Software Product Line (SPL) are usually captured in the form of a feature model, which represents the product line variation model, but this model lets several requirements details aside, such as the specification of functional and non-funcional requirements. Due to the crosscutting nature of SPL variations, researchers are using aspect-oriented techniques, to deal with such crosscutting concerns. In this context, the sooner these aspects can be identified the better, influencing the SPL and products architecture upfront. In this work we propose a Product Line extension to an aspect-oriented intentional model. The extended model provides both variability information and requirements details, promoting a natural blending of SPL and aspect-oriented architectural abstractions. We present the mapping between the SPL and the modeling approach abstractions and discuss two development scenarios: starting with a plain feature model and generating the extended aspect-oriented intentional model and the opposite approach.

*An earlier version of this paper was presented at the Early Aspects 2010 workshop at the 9th International Conference on Aspect-Oriented Software Development. The paper is recommended by Ruzanna Chitchyan and Steffen Zschaler.

Silva, L. et al. On the Role of Features and Goals Models in the Aspect-Oriented Development of Software Product Line. Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Early Aspects, R.Chitchyan, S. Zsachaler (Eds.), Vol. 2, No. 1 (2010) 60-65

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements Specifications

Keywords

Documentation, Design, Languages

1. Introduction

In Software Product Line (SPL) development [3], the product line requirements are usually captured in the form of a feature model [5], which represents the product line variation model, but it lets several requirements details aside, mainly if these details are commonalities, for instance, a mobile phone should make and receive calls, and usually this feature would not be represented in the feature model of a mobile phone. The feature model does not distinguish functional and non-functional requirements.

In software development in general, several requirements, when implemented, will derive concerns that are tangled with and spread over other concerns, the so-called crosscutting concerns [6]. In the SPL context, it is expected that these concerns are part of variation points implementations, therefore affecting several products. This situation will demand these concerns to be plugged either in or out of the SPL' products.

Such demand led to the use of aspect-oriented (AO) techniques [6], modeling included, with SPL development [1]. The use of aspects to implement product variations allows variations to be easily added or removed from a product configuration, without polluting the code with conditional compilation code, an alternative strategy that hinders program legibility leading to maintainability' issues [1].

In this context, the sooner these aspects can be identified the better, because they can be incorporated as part of early models, therefore influencing the SPL and products architecture upfront, instead of demanding changes only during design or implementation tasks.

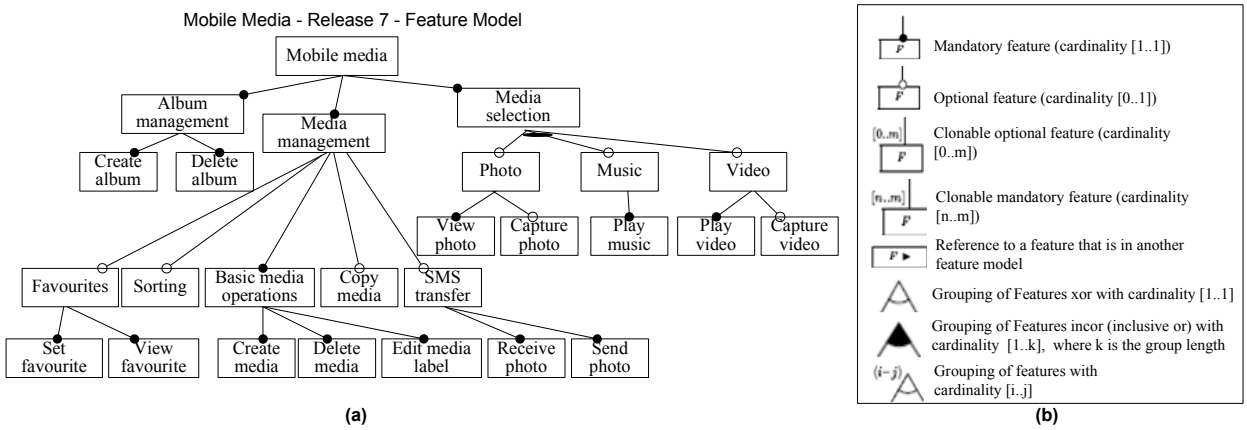

Figure 1: (a) Mobile Media Feature Model (b) Feature model notation.

One proposal to model a system' requirements including early aspects specification is using AOVgraph [8] an aspect- oriented intentional model. AOVgraph provides mechanisms to analyze positive and negative relationships among functional and non-functional requirements as well as to crosscutting concerns separation and composition.

In this work we intend to propose PL-AOVgraph, a Product Line extension to AOVgraph that includes variability information. PL-AOVgraph models will fill the gap that the feature models have regarding to the SPL requirements. The model will provide both variability information and requirements details. PL-AOVgraph is proposed as a seamless extension of AOVgraph. It promotes a natural blending of software product line and aspect-oriented architectural abstractions. Instead of burdening the requirements model with new abstractions to express product line specification, PL-AOVgraph adapts existing AOVgraph abstractions.

We also discuss possible SPL development scenarios, such as starting with a feature model and then generating a PL-AOVgraph, and vice-versa. In fact, the feature model can be automatically generated from the PL-AOVgraph, providing a simpler view and guaranteeing models consistency. The paper is structured as follows. Section 2 contains the background and the running example. Section 3 presents the PL-AOVgraph extension. A discussion is presented in Section 4 and related work at Section 5. Finally, Section 6 summarizes the paper with concluding remarks and future work.

2. Background

In this section we present the background of this work. Section 2.1 contains a brief description about feature model. Section 2.2 presents our running example, the Mobile Media system. Section 2.3 contains a brief description of AOVgraph.

2.1 Feature Model

A feature is a system property that is relevant to some stakeholders. Features are organized in feature diagrams. A feature diagram is a tree with the root representing a concept and its descendent nodes are features. Feature models are feature diagrams plus additional information such as feature descriptions, binding times, priorities, or stakeholders, among others.

Feature modeling is a key approach to capturing and managing common and variable features in a SPL [5]. They are used during early stages of SPL development for scoping the system family, later as a basis for building the product line architecture, and finally during the application engineering for guiding the requirement elicitation and analysis. Feature models were proposed as part of the Feature-Oriented Analysis method (FODA) [5].

There is a series of distinctive types of features identified:

- Concrete features such as data storage or functions that may be realized as individual components;
- Aspectual features that may affect several components and can be modularized using aspect technology;
- Abstract features such as performance requirements that are usually mapped to a configuration of components and/or aspects;
- Grouping features may represent variation points and they are mapped to a common interface of plug-compatible components.

2.2 Running Example - Mobile Media

MobileMedia (MM) [9] is a software mobile product line that provides support to manage (create, delete, visualize, play, send) different kinds of media (photo, music) on mobile devices. It extends an existing SPL, MobilePhoto, by including mandatory, optional, and alternative features. There are different MobileMedia implementations in Java, AspectJ and CaesarJ. Each implementation has 10 releases, where each release contains additional functionalities with respect to its predecessor release. Releases 1 to 5 deal only with photos and release 6 includes new features: store, play, and organize music. While release 7 includes features about videos.

Figure 1(a) illustrates the MM Feature Model of release 7. This model contains mandatory features representing commonalities such as *AlbumManagment, MediaManagement* and *MediaSelection*. The optional features are: *Favourites, Sorting, Copy Media, SMS Transfer, CapturePhoto* and *CaptureVideo*. Finally, the alternative (inclusive or) features are the media types: *Photo, Music* and *Video*. Figure 1(b) shows the feature model notation [4] used in this work.

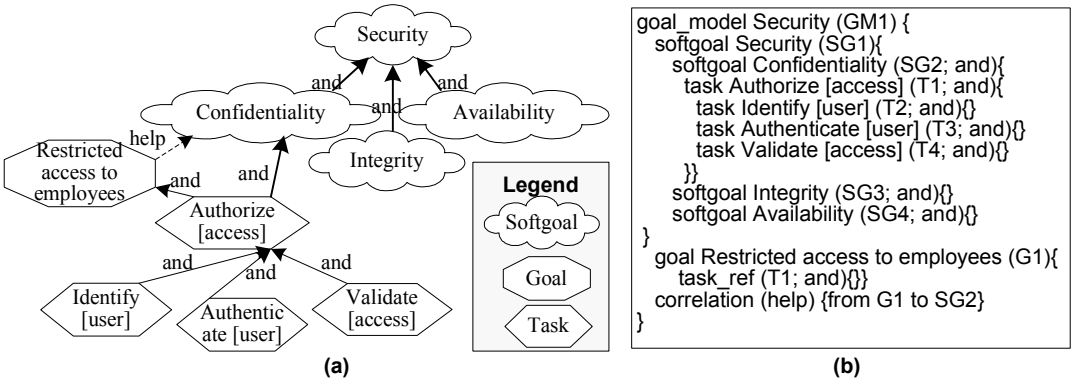

Figure 2: Example of AOVgraph model: (a) graphical notation and (b) textual notation.

Table 1: Summary of the mapping rules.

Feature model	⇔	PL-AOVgraph
Root of feature model	⇔	Name of goal model
Hierarchy of features	⇔	Hierarchy of goals, softgoals, and tasks
Feature	⇔	Task, goal or softgoal
Mandatory Feature	⇔	Contribution = and Property [typeFeature=mandatory; cardinality=(1,n)]
Optional Feature	⇔	Contribution = or Property [typeFeature=optional; cardinality=(0,n)]
Alternative xor/inc-or Feature Alternative xor/inc-or Feature	⇔ ⇔	Contribution = xor/inc-or Property [typeFeature=alternative; groupFeature={}; cardinality=(1,1)/(1,n)]
Constraint = "implies the non-selection"	⇔	Correlation (hurt)
Constraint = "implies"	⇔	Correlation (make)
Feature reference	⇔	Task_ref, goal_ref, or softgoal_ref

2.3 AOVgraph

AOVgraph [8] is an aspect-oriented intentional model, represented by AND/OR decomposition graphs. Its relationships map not just positive and negative conflicts between requirements (goals, softgoals, and tasks), they also map how these requirements crosscut each other. Furthermore, they represent choices of different options of how a given requirement may be achieved.

Requirements are represented by softgoals, goals, and tasks. Usually, softgoals are related to non-functional requirements, tasks are functional requirements and goals represent organizational objectives or the reason to select some group of tasks.

Decompositions, dependences and conflicts are represented by contributions, correlations or crosscutting relationships. Contributions can be AND, OR, or XOR labeled. Correlations can be MAKE, HELP, UNKNOWN, BREAK, and HURT labeled. While crosscutting relationships register how the requirements are scattered and tangling, through pointcuts, advice, and intertype declarations [8].

Figure 2 illustrates an example of an AOVgraph model, in (a) its graphical representation and in (b) its textual representation. In this example, we can see that the Security softgoal can be decomposed into *Integrity, Availability* and *Confidentiality* softgoals; *Confidentiality* is reached by *Authorize [access]* task; which is decomposed into *Identify [user], Authenticate [user]* and *Validate [access]* tasks; *Authorize [user]* contributes also to *Restricted access to employees* goal, which is correlated with *Confidentiality*.

3. Proposal

In order to promote software product-line engineering in the early stages of the software development process, we propose a bi-directional mapping between two modeling languages: Feature Model and PL-AOVgraph. Section 3.1 presents details of PL-AOVgraph, an extension of AOVgraph to deal with variability on SPLs, and Section 3.2 presents details of our approach to map PL-AOVgraph and feature model.

3.1 PL-AOVgraph

In this work we extend AOVgraph to consider issues of the SPLs domain, such as variability, in the SPL requirement model. The idea is to use the existing abstractions and to extend AOVgraph model without burdening the original model, in the following way: creating four new properties named isFeature, typeFeature, groupFeature and cardinality. The first one, determinates if a goal, softgoal, or task is to be represented as a feature in the feature model; the second one represents which type of feature a goal, softgoal, or task will represent in the feature model, this property was added to allow users choice if a element in PL-AOV-graph is to be a feature, it will be better explained in future work; the third property defines which features are grouped; and the forth property represents cardinality of features and groups of features.

This extension does not violate AOVgraph foundations, nor makes the model harder, because it only adds properties, a secondary element in AOVgraph. Furthermore, an inc-or label was added into PL-AOV-graph, in order to represent inclusive or contributions.

Table 1 summarizes how each element in a feature model is represented in PL-AOVgraph: Features are abstracted by tasks, goals, and softgoals; relations between features are abstracted by contributions; and variability is abstracted by properties.

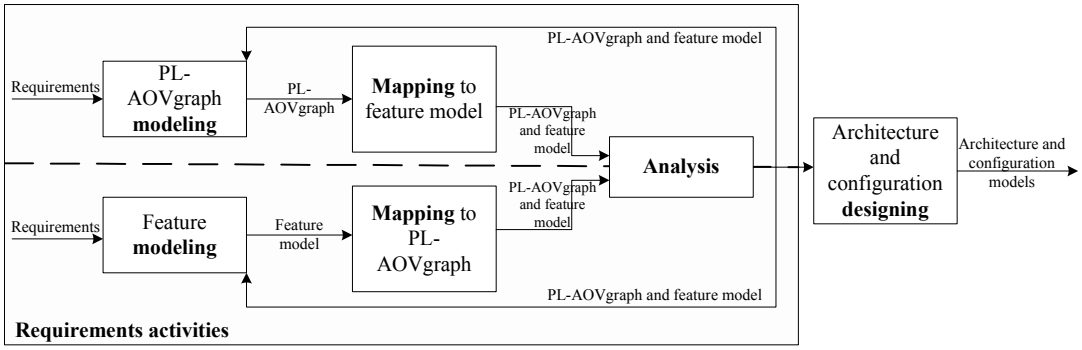

Figure 3: Product-line development process using PL-AOVgraph and feature model.

3.2 Process

Figure 3 shows how to use Feature Model and PL-AOVgraph to promote aspect-oriented product-line development. In the beginning of the process, there are two independent alternative flows: (i) the first (illustrated above with dotted line) begins with PL-AOVgraph modeling after which we transform this model into a feature model; and (ii) the second flow (illustrated below with dotted line) begins with Feature modeling after which we transform this model into a PL-AOVgraph model. In both cases, the third step is to analyze the created models in order to identify mistakes and omissions. If there are changes to be done, then the analyst goes back to model PL-AOVgraph or feature model, otherwise configuration and aspectual architecture models can be generated in the design activity, using appropriated approaches [8].

With this process, we can observe three possible situations: (i) the process only begins with feature modeling; (ii) the process only begins with PL-AOVgraph modeling; and (iii) the process begins with both feature and PL-AOVgraph modeling. In the first and second situations, we generate the other model as another view of the system, which will be used to help understand the system domain and to bridge the gap between requirements and architecture or configurations. However, in the third situation where both models are available, it is also necessary guarantee the consistency between them and to keep track of traceability relationships between them. Besides the consistency, it is also important to notice that starting from a Feature Model, generating the PL-AOVgraph and then generating the Feature Model back, will probably generate a Feature Model different from the first one.

In this paper, we explore the first two situations, while the third one will be explored in future work. Therefore, in this paper we deal with bi-directional mapping in order to promote the product-line development: i) map PL-AOVgraph into Feature model and ii) map Feature model into PL-AOVgraph.

Bi-directional mapping advantages are: (i) there is a feedback among these models, making the generated models richer; (ii) we can guarantee consistency between them; (iii) we can promote completeness in both models; (iv) we can generate aspectual architecture models from features models; and (v) we can generate configuration models from PL-AOVgraph models. On the other hand, the main drawback is the overhead to keep both models consistent

These transformations are based on the semantic and syntactic elements from both models. Therefore, although these models have divergent goals, they have some similarities, such as: both of them are structured as trees; they represent functional and non-functional requirements and they have some similar relationships. However, they also have some semantic differences, such as: PL-AOVgraph focuses on representing all requirements of a system, while Feature models focus on representing the variability; PL-AOVgraph represents the system in terms of goals, softgoals, and tasks, while Feature models represent the system in terms of features.

Table 1 summarizes the mapping rules to transform these models each other, and in the following sections we detail these mapping rules.

3.2.1 Feature model to PL-AOVgraph

The process to map feature model into PL-AOVgraph consists of the following five steps:

1. The root of feature model is transformed into a goal model in PL-AOVgraph;
2. Each feature is transformed into a task. If the feature is optional, then it is transformed into a task with contribution relationship or; if it is mandatory, it is transformed into a contribution relationship labeled and; if it is alternative, it is transformed into a contribution relationship labeled xor or inc-or depending on the cardinality of the feature group;
3. Furthermore, each task is annotated with the following properties: isFeature = “yes”; typeFeature = “mandatory | alternative | optional”;
4. Constraints (“implies | implies the non-selection”) are transformed into correlation relationships labeled make or hurt;
5. If there are many constraints to a same feature (for instance, A implies B, C, and D; and B, C and D’s father is X), this feature generates a crosscutting relationship, on which: it will be the pointcut (A will be pointcut); the features implicated will be transformed into advice (B, C, and D will be advice); and them father will be the source of relationship (X will be source).

Figure 4 shows an example of a PL-AOVgraph generated from a feature model. We can see that all features are

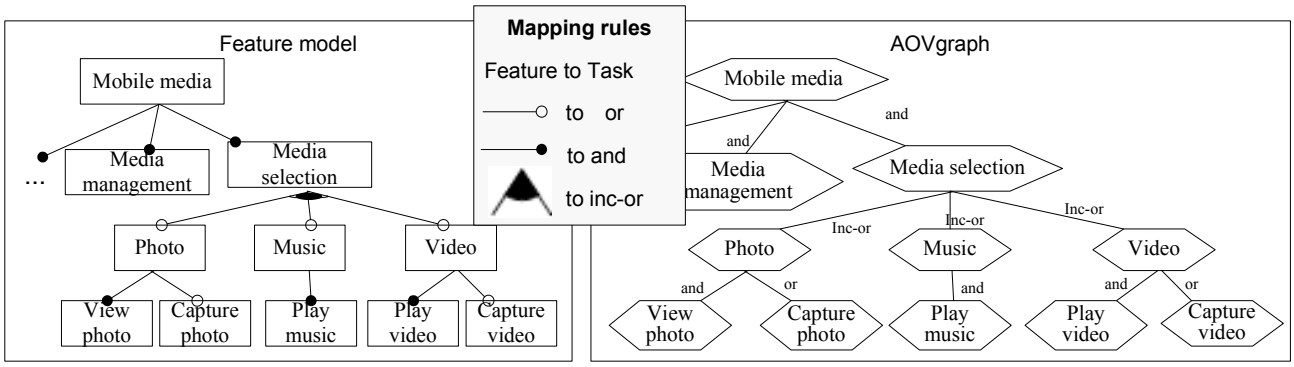

Figure 4: Example of transformation between Feature Model and PL-AOVgraph.

```
goal_model Mobile media (GM1) {
...
  task Video (T3; and) {
    property {isFeature=yes;
              typeFeature=alternative inc-or;
              groupFeature={Photo; Music}
  }
  task Play video (T4; and) {
    property {isFeature=yes;
              typeFeature=obligation}
  }
  task capture video (T5; and) {
    property {isFeature=yes;
              typeFeature=optional}
  }
...
} }
```

Figure 5: Example of "properties" on AOVgraph specifications.

transformed into tasks, and the hierarchy between them follows the same hierarchy of features. Figure 5 shows how properties are modeled in the textual representation of PL-AOVgraph.

3.2.2 PL-AOVgraph to Feature Model

The process to map PL-AOVgraph into feature model consists of the four following steps:

1. Each goal model is transformed into the root of a feature model;
2. Each task, goal and softgoal is transformed into a feature unless there is a property called "isFeature=no". If there are not these properties, then the kind of contribution defines the type of feature: or to optional, and to mandatory, xor to alternative xor, and inc-or to alternative inc-or features;
3. References to goals, softgoals, and tasks, represented in PL-AOVgraph by task-ref, goal-ref, and softgoal-ref, are transformed into: (i) Ref features — if this element is defined in the same goal model and (ii) into features with a graphic symbol — if this element is defined in another goal model (in accordance with the notation defined in Figure 1(b);
4. Advice of crosscutting relationships are transformed into features in accordance with the following rules: each pointcut will be a feature that group the sub-features generated by tasks, goals, and softgoals from advice.

Figure 6 shows an example of a Feature Model generated from a PL-AOVgraph by following the mapping steps.

4. Discussion

As presented in Section 3, this work proposes a bi-directional mapping between the Feature Model and PL-AOVgraph in order to promote software product-line engineering. This section relates our experience with this mapping, showing our difficulties and drawbacks.

- In the feature model notation used in this work it is not possible to identify if a feature is a functional or non- functional. Therefore, we decided to map each feature (functional or not) into a task. This is not always the better option, this problem can be solved if we use a notation which makes abstract and concrete features different, as PL-AOVgraph does. This situation shows that a feature model does not distinguish the type of requirements.
- Names generated from feature models cannot be appropriated to tasks in PL-AOVgraph, since task names should contain a verb. This information could also be used to identify if a feature is functional or non-functional and thus map it into a softgoal or task. For now, we did not address this issue.
- PL-AOVgraph is more detailed than the feature model, since its goal is to represent all requirements of a system, while the feature model aims to model commonalities and variabilities. Therefore, with this mapping we can generate a feature model excessively detailed making the variability visualization less evident. On the other hand, we can generate PL-AOV-graph excessively summarized, making the representation of requirements insufficient.
- While the Mobile Media case study has allowed us to define elements to define the proposed mapping rules, we consider these rules as initial rules which can be further refined. On the other hand, we could see with this work that although these models have different purposes and semantic differences, there are enough similarities to work with them at the same time and that one can give feedback to the other.

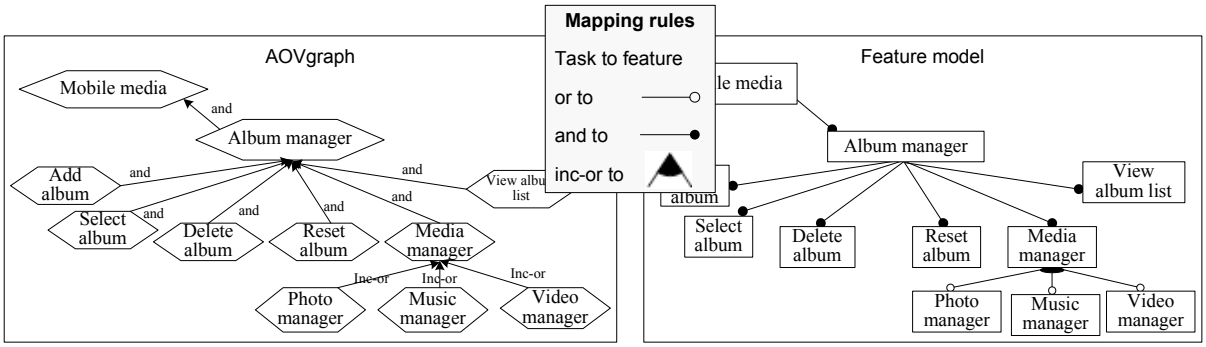

Figure 6: Example of transformation between PL-AOVgraph and Feature Model.

5. Related Work

We can cite some approaches related to our work, as following: Yu et al [10] define the mapping between goal model and feature model, but in their approach the goal model does not represent grouped features and cardinality. Silva et al [7] map Aspectual i* to feature model. In this mapping, Aspectual i* is not also sufficient to represent all variability of a feature model. Therefore, Borba and Silva [2] extend i*, creating new relationships in order to represent this variability.

Our work is similar to these approaches, since it establishes a mapping between PL-AOVgraph and feature model and we also have created some elements in PL-AOVgraph in order to represent completely variability. However, there are two main differences: (i) the elements created in PL-AOVgraph are defined as properties, which are less intrusive elements than the elements created by Borba and Silva in i* [2]; (ii) our approach does not aim to substitute PL-AOVgraph for the feature model, or vice-versa. Our approach states that both models are essential to SPL engineering, and thus they should be developed at the same time. In this way, the engineer has two views, in early stages of the software development, that make it possible to analyze, model and make decisions about the SPL development.

6. Conclusions

In this work we propose a bidirectional mapping between PL-AOVgraph and feature models. The PL-AOVgraph model provides both variability information and requirements details while feature models do not include requirements details. We presented a mapping between SPL and AOVgraph abstractions and discussed two possible development scenarios: starting with a feature model or a PL-AOVgraph model and generation the other. Steps were proposed to achieve one model from the other one.

As future work we are going to refine some mapping rules, such as those related to constraints and crosscutting relationships; define a traceability mechanism which manages and propagates changes made in PL-AOVgraph or feature models; develop a tool to automate these rules. We also want to evaluate how the use of both models in real environments makes the SPL development easier. Another future work is considering alternative variability models and also map them from/to PL-AOVgraph.

Acknowledgements. This work is part of the Latin-American Aspect-Oriented Software Development network (LA-AOSD), supported by CNPq. Sérgio Soares is partially supported by CNPq, grant 309234/2007-7. Thais Batista is partially supported by CNPq, grant 301880/2007-7.

References

[1] V. Alves, P. Matos, Jr., L. Cole, A. Vasconcelos, P. Borba, and G. Ramalho. Extracting and evolving code in product lines with aspect-oriented programming. *Transactions on aspect-oriented software development IV*, pages 117–142, 2007.

[2] C. Borba and C. Silva. A comparison of goal-oriented approaches to model software product lines variability. In *ER '09: Proceedings of the ER 2009 Workshops (CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS) on Advances in Conceptual Modeling - Challenging Perspectives*, pages 244–253, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[4] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *OOPSLA'05 - Workshop on Software Factories*, October 2005.

[5] K. C. Kang et al. Feature-oriented domain analysis feasibility study. Technical report, Software Engineering Institute, 1990.

[6] G. Kiczales et al. Aspect-oriented programming. In *European Conference on Object-Oriented Programming, ECOOP'97*, pages 220–242. Springer, June 1997.

[7] C. Silva, F. Alencar, J. Araújo, A. Moreira, and J. Castro. Tailoring an aspectual goaloriented approach to model features. In *20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, pages 472–477. Knowledge Systems Institute Graduate School, July 2008.

[8] L. F. Silva, T. V. Batista, A. Garcia, A. L. Medeiros, and L. Minora. On the symbiosis of aspect-oriented requirements and architectural descriptions. In *Proceedings of the 10th international conference on Early aspects*, pages 75–93. Springer-Verlag, 2007.

[9] T. Young. Using aspectj to build a software product line for mobile devices. Master's thesis, University of British Columbia, 2005.

[10] Y. Yu, J. C. S. do Prado Leite, A. Lapouchnian, and J. Mylopoulos. Configuring features with stakeholder goals. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 645–649, New York, NY, USA, 2008. ACM.