On the Integration of the Feature Model and PL-AOVGraph

Lidiane Santos
Computer Science Department
Federal University of Rio Grande do Norte (UFRN)
diane_lid@hotmail.com

Lyrene Silva
Computer Science Department
Federal University of Rio Grande do Norte (UFRN)
lyrene@gmail.com

Thais Batista
Computer Science Department
Federal University of Rio Grande do Norte (UFRN)
thaisbatista@gmail.com

ABSTRACT

In this paper we propose PL-AOVGraph, an extension to the aspect-oriented requirements modeling language, AOV-Graph, to support the definition of software product line requirements. With PL-AOVGraph it is possible to specify requirements and variabilities. In general SPL variabilities are represented using the Feature Model, however, this model does not represent the requirements of the system. PL-AOVGraph and the Feature Model are complementary approaches as they represent different perspectives of a system. With the goal of inserting PL-AOVGraph in the SPL development process, this work proposes a bi-directional mapping between PL-AOVGraph and the Feature Model.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications.

General Terms
Documentation, Design, Languages.

Keywords
Software Product Line, Feature Model, PL-AOVgraph, requirements, variabilities

1. INTRODUCTION

Software Product Line Development (SPL) [2] supports the creation of a portfolio of similar products using a common software infrastructure to assembly and configure parts designed to be reused across products. SPL approaches identify *commonalities* of all family members, as well as features that vary among members of the family, the *variabilities*. Thus, members of a family have a basic set of common functions with many variants. A fundamental challenge in this context is to manage the variabilities by defining the variation points and the dependencies between them. A same feature can be spread and tangled in a same product. In order to handle this crosscutting nature of common and variable features, Aspect-oriented software development (AOSD) [3] has been recently explored in the development of SPLs. In general such crosscutting elements cannot be suitably modularized with conventional variability mechanisms, such as conditional compilation or inheritance [2]. Therefore, AOSD can be used to support improved modularity of crosscutting concerns, expressing them as aspects.

EA'11, 21-MAR-2011, Porto de Galinhas, Brazil
Copyright 2011 ACM 978-1-4503-0645-4/11/03…$10.00.

Following this tendency of integrating SPL and AOSD, in this paper we propose PL-AOVGraph [7], an aspect-oriented requirement language that extends AOV-Graph [9] by adjusting its aspect-oriented abstractions to support the SPL concepts. PL-AOVGraph includes a new type of relationship and properties.

Feature models represent commonalities and variabilities in terms of *features*. A *feature* is a concept that is prominently visible to any stakeholder involved in the development of applications. This model provides a clear representation of the features that are relevant to the product line family domain. However, the high level of abstraction of the feature model lets several requirements details aside. Thus, the feature model must be integrated with other requirements model in order to provide more detailed and meaningful information to the development of a SPL. In this context, we propose PL-AOVGraph, an aspect-oriented requirement modeling language that represents both the variability and the requirement information. The aim of PL-AOVGraph is to complement the feature model by (i) detailing the requirements with SPL information, and (ii) identifying and modularizing crosscutting concerns.

As the feature model is already part of the SPL development process and with PL-AOVGraph it is possible to identify and modularize the crosscutting concerns, in this work we propose a bi-directional mapping between the Feature Model and PL-AOVgraph. Via this mapping it is possible to associate the elements of the feature model and the PL-AOVgraph elements and to include PL-AOVGraph in the development process of SPL using existing feature model. The mapping defined in this work was implemented in the ReqSys tool [5] – an Eclipse plug-in that allows the automatic generation of a PL-AOVGraph specification from a feature model and vice-versa. This paper also presents a case study that illustrates the result of the mapping.

This paper is structured as follows. Section 2 contains a brief presentation about the feature model. Section 3 presents PL-AOVGraph. Section 4 contains the details about the bi-directional mapping. Section 5 presents the case study. Section 6 present some related work and Section 7 contains the final remarks.

2. FEATURE MODEL

As previously mentioned, features are organized in feature models that are hierarchical graphs where the root is the context of the model and the descendent nodes are features. Features can be classified into: (i) **Mandatory**: all products of the family must contain this feature; (ii) **Optional**: the products can contain this feature or not; (iii) **Alternative**: the products must contain exactly one feature from a group of features; (iv) **Inclusive-or**: the products must contain at least one from a group of features.

Feature models can contain additional information such as cardinality, groups, attributes, references, and annotations defined by the users.

Figure 1 shows a part of the Mobile Media feature model, a SPL to mobile devices. "Media Selection", "View Photo", "Play Music", and "Play Video" are mandatory. "Capture Photo" and "Capture Video" are optional and "Photo", "Music", and "Video" are inclusive-or, representing the variabilities.

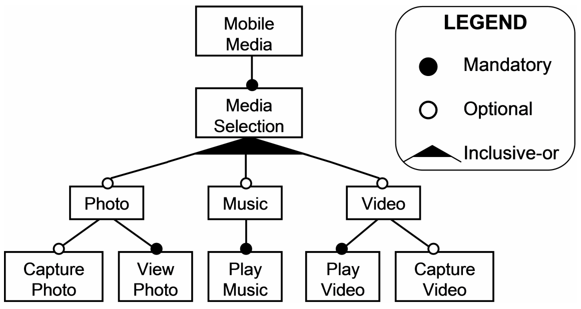


Figure 1. Mobile Media Feature Model (partial).

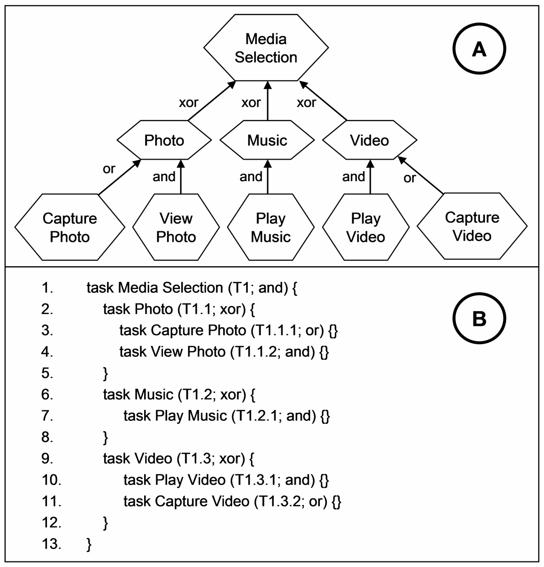
3. PL-AOVGraph

PL-AOVGraph is an extension of the AOV-Graph goals model that inherits all its properties. It can represent positive and negative conflicts among the requirements (goals, softgoals, and tasks). It also modularizes the crosscutting concerns. AOV-Graph is open to include new properties to the goals model by using the *property* element. Thus, PL-AOVGraph does not include new elements it semantically enriches existing AOV-Graph elements by including the following properties to support variabilities: cardinalityMin, cardinalityMax, groupFeature, cardinalityGroupMin, cardinalityGroupMax, and isFeature.

The *cardinalityMin* and *cardinalityMax* properties are used to associate the minimum and maximum cardinality to a component, respectively. The PL-AOVGraph *groupFeature*, property specify the members of a group and the cardinalityGroupMin and cardinalityGroupMax properties are used to determine the cardinality of the group. The *isFeature* property indicates if a PL-AOVGraph component is equivalent or not to a feature. This is a decision of the requirements engineer when elaborating the PL-AOVgraph specification because depending on the abstraction level, a requirement is not always a feature. For instance, the feature model that focuses on users in general does not present implementation requirements.

PL-AOVGraph also includes a new type of contribution relationship, named *inc-or*, to indicate that at least one and at most all elements with this relationship must be included in the product line.

Figure 2 presents the PL-AOVGraph representation of Mobile Media, (a) graphical notation (b) textual notation. The "Media Selection" task has three contributions of the *xor* type (Photo, Music, and Video), that indicates that one of those must be included in the product line. The "Photo" task has two contributions: (i) *or* type (Capture Photo), indicating that this element can be included or not in the product (ii) *and* type (View Photo), indicating that this element is always included. The "Music" task has just one contribution *and* (Play Music) and the "Video" task has two contributions: (i) *and* (Play Video), (ii) *or* (Capture Video).



```
task Media Selection (T1; and) {
    task Photo (T1.1; xor) {
        task Capture Photo (T1.1.1; or) {}
        task View Photo (T1.1.2; and) {}
    }
    task Music (T1.2; xor) {
        task Play Music (T1.2.1; and) {}
    }
    task Video (T1.3; xor) {
        task Play Video (T1.3.1; and) {}
        task Capture Video (T1.3.2; or) {}
    }
}
```

Figure 2. PL-AOVgraph example: (a) Graphic Notation (b) Textual Notation.

4. BIDIRECTIONAL MAPPING

This section describes a bidirectional mapping among the feature model and PL-AOVgraph. Section 4.1 explains how this mapping can be inserted in SPL development process. Section 4.2 presets a running example – the Smart Home system. Section 4.3 defines the mapping rules, associating the elements of these artifacts, features models and PL-AOVGraph specifications. Section 4.4 reports some constraints of this bi-direction mapping.

4.1 The process

Silva et al [10] explain two situations to use this bidirectional mapping in the SPL development: (i) when there is only a PL-AOVGraph specification, and (ii) when there is only a feature model. In the first case, a PL-AOVGraph model is created from requirements and it will be input to the bidirectional mapping, generating a feature model. In the second case, there is a feature model generated from the requirements and it will be the input to the bidirectional mapping, generating a PL-AOVGraph specification. After that, in both of cases, the outputs must be analyzed in order to identify and correct mistakes and omissions and then go back to the bidirectional mapping again. When corrections are not necessary, the PL-AOVGraph specification and the feature model can be used to help the development of the architecture and other design models.

4.2 Running Example – Smart Home

Smart Home [7, 9] is a SPL to residential systems. A smart home can contain several floors, with many rooms, each room can contain controllers, such as, weather, doors, windows, lights controllers, fire detector, and presence simulator. [6].

Sánchez et al [6] defines the Smart Home features model (Figure 3) and describes its functional requirements. Based on this requirements and non-functional requirements defined by Tomás et al [11], we create a PL-AOVGraph specification [7] presented in Figure 4.

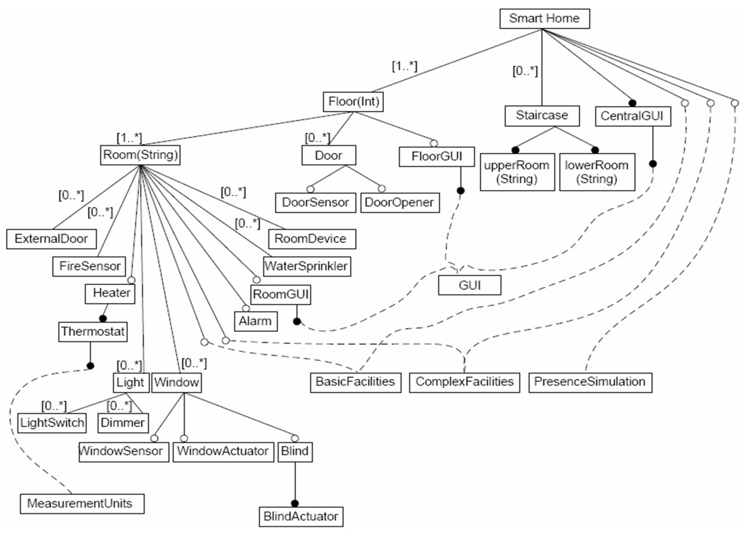


Figure 3. Smart home feature model.

```
goal_model (Smart Home; GM1){
  task Fire Detection (or; T2;){
    task Enable Alarm (or; T2.1;){
      task Activate Siren (inc-or; T2.1.1;){ property{isFeature=no} }
      task Activate Lights (inc-or; T2.1.3;){ property{isFeature=no} } }
    task Sprinkle Water (and; T2.2;){} }
  task Light Management (or; T3;){
    task Regulate Intensity Light Automatically (inc-or; T3.1;){}
    task Select Predefined Values [Light] (inc-or; T3.2;){
      task Select mode [TV watching] (inc-or; T3.2.1;){}
      task Select mode [Reading] (inc-or; T3.2.2;){}
      task Select mode [Normal] (inc-or; T3.2.3;){}
      task Select mode [Ambient] (inc-or; T3.2.4;){} } }
  task Presence Simulation (or; T5;){
    task_ref = (Regulate Blinds Automatically; T1.2; inc-or;) }
  task Minimize Waste of Energy (or; T6;){
    task Measure Luminosity (and; T6.1;){}
    task Detect Movement (and; T6.2;){}
    task_ref = (Regulate Heater Automatically; T4.2.1; inc-or;) }
  softgoal Security (S1;){
    softgoal Maintaining Privacy (S1.1;){
      softgoal Access Control (S1.1.1;){} }
    softgoal Protect Communications (S1.2;){} }
  softgoal Availability (S2;){
    softgoal Availability [Controllers] (S2.1;){}
    softgoal Availability [Sensors] (S2.2;){}
    softgoal Availability [Actuators] (S2.3;){} }
  correlation (hurt){
     source = softgoal_ref = (Availability; S2;)
     target = softgoal_ref = (Security; S1;) }
   crosscutting (source = Light Management (T3)){
     pointcut (PC1): include(Presence Simulation; T5;) and
                     include(Minimize Waste of Energy; T6;)
     advice (around): PC1{
         task_ref = (Regulate Intensity Light Automatically; T3.1; inc-or;)} } }
```

Figure 4. Smart Home PL-AOVgraph specification.

4.3 Mapping Rules

Section 4.3.1 presents the mapping rules to generate the PL-AOVGraph specification from the feature model. Section 4.3.2 shows the mapping rules to transform the feature model into a PL-AOVgraph specification.

4.3.1 Mapping Features model to PL-AOVgraph

Table 1 summarizes the rules to transform Features models into PL-AOVgraph.

By using these rules (described in table 1), the feature model of figure 3 is mapped into the PL-AOVGraph specification illustrated in Figure 5, as following:

- Rule 1: "Smart Home" root in the feature model is mapped into the Smart Home goal model in PL-AOVgraph.
- Rule 2: All hierarchy of this feature model is mapped into a similar hierarchy in PL-AOVgraph. In this case, features are mapped into tasks.
- Rule 3: Mandatory (for instance, "BlindActuador") and optional features (for instance, "FloorGUI"), are mapped into AND and OR contributions, respectively.
- Rule 4: Features with cardinality (for instance "Floor") are transformed into tasks with cardinalityMin and cardinalityMax properties.
- Rule 6: Reference features (for instance, "GUI") are mapped into task_ref in PL-AOVgraph.

Rule	Description
1	Each feature model generates a goal model.
2	Each goal model generated consists of a hierarchy identical to the feature model hierarchy, i.e., a feature father will be transformed into a task father in PL-AOV-graph, and so on.
3	Mandatory, optional, alternative and o r-inclusive features are represented by and, or, xor, and inc-or contribution relationships, respectively.
4	Features with cardinality are transformed into tasks with cardinalityMin and cardinalityMax properties.
5	Grouped features with cardinality are transformed into grouped tasks with groupFeature, cardinalityGroupMin and cardinalityGroupMax properties.
6	Features defined as reference are mapped to task_ref.
7	If a feature has an annotation about a correlation relationship (hurt, break, make, help, unknown) then it is mapped into a source of a correlation whose type and target will be described by an annotation.
8	If there are more than one reference to the same feature then this feature is mapped to an advice of a crosscutting relationship, features linked to that feature are mapped into pointcuts and the feature father of feature referred is mapped into the source of this crosscutting relationship.
9	If a feature has an annotation defining one type of PL-AOVGraph components (task, goal, softgoal), then this feature generates a component of the type described in this annotation.

Table 1: Rules to transform Feature Models into PL-AOVgraph.

```
goal_model (Smart Home; GM1){
 task Floor(int) (T1;){property{cardinalityMin=1;cardinalityMax=n;}
   task FloorGUI (or; T2;){
     task_ref = (GUI; T26; and;) }
   task Door (T3;){property{cardinalityMin=0;cardinalityMax=n;}
     task DoorSensor (or; T4;){}
     task DoorOpener (or; T5;){} }
   task Room(String) (T6;){property{cardinalityMin=1;cardinalityMax=n;}
     task RoomDevice (T7;){property{cardinalityMin=0;cardinalityMax=n;}}
     task WaterSprinkler (T8;){property{cardinalityMin=0;cardinalityMax=n;}}
     task RoomGUI (or; T9;){
       task_ref = (GUI; T26; and;) }
     task Alarm (or; T10;){}
     task Window (T11;){property{cardinalityMin=0;cardinalityMax=n;}
       task Blind (or; T12;){
         task BlindActuador (and; T13;){} }
       task WindowActuator (or; T14;){}
       task WindowSensor (or; T15;){} }
     task Light (T16;){property{cardinalityMin=0;cardinalityMax=n;}
...
```

Figure 5. Smart Home PL-AOVGraph generated from the Feature model (partial view).

In this case study, there are no situations to use rule 5, 7 and 9, because there are no grouped features either annotations. “GUI”, “BasicFacilities”, and “ComplexFacilities” features can be considered advices of a crosscutting relationship, because there are references repeated more than once (in accordance with the rule 8 in Table 1), however in the feature model proposed by Sánchez et al [6] each one of these features is a distinct feature model (distinct tree). Therefore, these features should be analyzed by the requirement engineer. The Rule 8 is not applied in this case. But if we consider the example shown in Figure 6 we see that the "Regulate Light Intensity Automatically" reference feature appears twice in the feature model, therefore it represents the advice of the crosscutting relationship, while "Presence Simulation" and "Minimize Waste of Energy" are the pointcuts because they are the features that call the feature corresponding to advice. The source of the crosscutting relationship is "Light Management" because it is the father of the referenced feature, as shown in Figure 7 (lines 11-15).

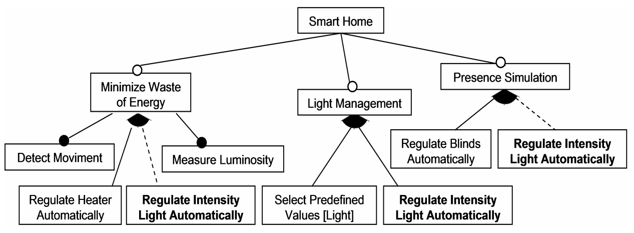

Smart Home
Minimize Waste of Energy
Light Management
Presence Simulation
Detect Moviment
Measure Luminosity
Regulate Blinds Automatically
Regulate Intensity Light Automatically
Regulate Heater Automatically
Regulate Intensity Light Automatically
Select Predefined Values [Light]
Regulate Intensity Light Automatically

Figure 6. Identifying crosscutting relationship in the feature model.

```
1.  goal_model (Smart Home; GM1){
2.    task Minimize Waste of Energy (or; T1;){
3.        task Measure Luminosity (and; T2;){}
4.        task Detect Movement (and; T3;){}
5.        task Regulate Heater Automatically (inc-or; T4;) }
6.    task Presence Simulation (or; T5;){
7.        task Regulate Blinds Automatically (inc-or; T6;) }
8.    task Light Management (or; T7;){
9.        task Regulate Intensity Light Automatically (inc-or; T8;){}
10.       task Select Predefined Values [Light] (inc-or; T9;){} }
11.   crosscutting (source = Light Management (T7)){
12.       pointcut (PC1): include(Presence Simulation; T5;) and
13.                       include(Minimize Waste of Energy; T1;)
14.       advice (around): PC1{
15.           task_ref = (Regulate Intensity Light Automatically; T8; inc-or;) } } }
```

Figure 7. Crosscutting relationship generated from the feature model.

4.3.2 Mapping PL-AOVGraph to the Feature Model

Table 2 presents the rules to transform a PL-AOVGraph specification into a Feature Model.

Rule	Description
1	Each goal model is mapped into a feature model root.
2	Goals, softgoals and tasks hierarchy is mapped into a similar feature hierarchy, i. e., a task root is transformed into a feature root, a goal leaf is mapped into a feature leaf, and so on.
3	And, or, xor and inc-or contributions are mapped into mandatory, optional, alternative and or-inclusive features, respectively.
4	Goals, softgoals and tasks with cardinalityMin and cardinalityMax properties generate features with these properties, as follows: if cardinalityMin=0 then it is generated an optional feature with cardinality [0..m], if cardinalityMin != 0 then it is generated a mandatory feature with cardinality [n..m], where n is given by cardinalityMin and m is given by cardinalityMax.
5	Goals, softgoals, and tasks grouped with the *groupFeature* property are mapped into grouped features with cardinality [i..j], where i is given by cardinalityGroupMin and j is given by cardinalityGroupMax.
6	Goals, softgoals, and tasks that are references generate references features.
7	Correlation relationships are mapped into annotations with the type of correlation and a feature related to target, this annotation is added to the feature generated by source of this correlation.
8	In crosscutting relationship, advices are mapped into reference features related to features defined in pointcuts.
9	Components with *isFeature* set to “no” are not mapped into a feature
10	The type of components in PL-AOVGraph (task, goal, or softgoal) generates an annotation in the feature, specifying this type.

Table 2. Rules to transform PL-AOVGraph into the Feature Model.

By using these rules (table 2), the PL-AOVGraph specification (Figure 4) is mapped into the feature model (Figure 8), as follows:

- Rule 1: “Smart Home” goal model (line 1) is mapped into a feature model root.
- Rule 2: Features are generated from tasks, goals and softgoals, following the same hierarchy described in PL-AOVgraph.
- Rule 3: Tasks with AND (for instance, “Sprinkle Water”, line 6), OR (for instance, “Fire Detection”, line 2) and inc-or (for instance, “Regulate Intensity Light Automatically”, line 8) contributions are mapped into mandatory, optional and inclusive-or features, respectively.
- Rule 6: Task_refs (for instance, “Regulate Blinds Automatically”, line 15) are transformed into reference features.
- Rule 7: Correlation relationships are mapped into annotations, for instance, the hurt correlation from “Availability” to “Security” (lines 28-30) is mapped into an annotation in Availability describing the type (hurt) and the target (Security).
- Rule 8: As the crosscutting relationship has not a representation in feature model, it is mapped into features, for instance: the “Regulate Intensity Light Automatically” advice (line 35) generates two reference features related to the features correspondent to its pointcuts: “Presence Simulation” (line 32) and “Minimize Waste of Energy” (line 33).

- Rule 9: PL-AOVgraph elements with the *isFeature* property equal to “no” are not transformed into a feature. For instance “ActivateSiren” (line 4) and “Activate Lights” (line 5) tasks. It is necessary to stress this decision and the setting is not done by bidirectional mapping, it is a manual configuration.
- Rule 10: each feature is generated with an annotation describing the type of the PL-AOVGraph component from which it was originated. Figure 8 presents two annotations, in “Security” and “Presence Simulation” features.

In this case study rules 4 and 5 are not necessary.

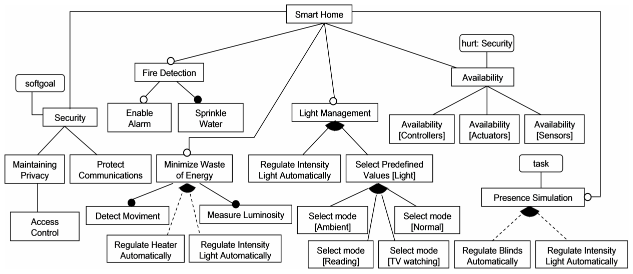


Figure 8. Smart Home feature model generated from PL-AOVgraph.

4.4 Constraints

There are some constraints to be considered when executing the bidirectional mapping between the feature model and PL-AOVgraph:

Feature model to PL-AOVGraph – (i) tasks naming cannot be appropriated because tasks names should contain a verb, while features name cannot contain it; (ii) only the around *advices* are generated. Intertype declarations are not generated, so there are limitations in the use of the resources offered by PL-AOVgraph.

PL-AOVGraph to the Feature model – feature models contain so many features, because each requirement (task, goal and softgoal) generates a feature (except when the *isFeature* property is set to “no”). It can be seen as an advantage, because the generated feature model is more complete, and can be seen as a drawback, because this feature model can be too big.

5. CASE STUDY: SMART HOME

As illustrated in Figure 9, this case study consists of two stages where in each one two transformations are performed. All transformations have been automated by the ReqSys tool [5].

In the first stage, using the smart home PL-AOVGraph specification it was generated a Feature Model. Then, this Feature Model was the source for the reverse transformation, producing a new specification PL-AOV-graph.

In the second stage, using the Feature Model defined by Sanchez et al [6], partly presented in figure 3, it was generated a PL-AOVGraph specification which after was input to the inverse transformation, producing a new Feature Model.

After the transformations, we compared the results. These results are presented in section 5.1. The full description is available at https://sites.google.com/site/plaovgraph.

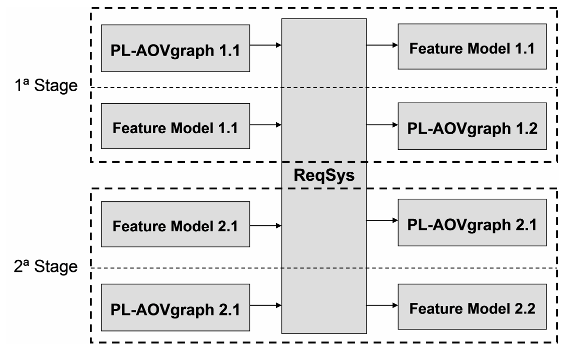


Figure 9: Steps used to transform the case study.

5.1 Analysis of the Case Study

Sections 5.1.1 and 5.1.2 present the analysis of 1st and 2nd stage of this case study, respectively. Section 5.1.3 describes the analysis involving the artifacts used in both steps.

5.1.1 Analysis of 1st Stage

Comparing the PL-AOVGraph specifications 1.1 and 1.2, some tasks of the 1.1 specification do not appear in the 1.2 specification. This occurs because on the specification 1.1 these tasks have the *isFeature* property set to “no”. Another difference is that the identifier of the tasks always is incremented but this does not affect the consistency of the specification.

In the PL-AOVGraph specification 1.2 there is no problem regarding the constraint of naming of tasks since this specification was obtained from the Feature Model 1.1 which was also generated from a PL-AOVGraph specification.

Regarding the Feature Model 1.1 it has a large number of features since each requirement has been transformed into a feature, except only two, due to the *isFeature* property. We conclude that this strategy make difficult the visualization of the feature model and the analysis of variabilities. This problem is worse in case of large specifications.

5.1.2 Analysis of 2st Stage

The Feature Models 2.1 and 2.2 are equal. Regarding the PL-AOVGraph specification 2.1, we observe the problem of naming tasks because the name of the features in the Feature Model 2.1 is not always composed by a verb. An example is the “Door” task. The noun “door” alone does not indicate any function that the system needs to realize. Therefore it is not a requirement. In this case it would be relevant to describe which functionalities of the system would interact with the door, for example “Open/Close Door automatically”.

5.1.3 General Analysis

Comparing the Feature Models 1.1 and 2.1 we conclude that they present different views of the system. At first some features represent physical components of the house, it is the case of "Heater", "Window", and "Light", while in the second all features represent requirements of the system.

Feature Models generated from PL-AOVGraph specifications represent the system under a more detailed view. On one hand this is an advantage because it makes the model more complete, it provides more information to the development team. On the other hand, there is the drawback of the complexity and the size of the model, which can negatively interfere in its use.

Similarly, the PL-AOVGraph specifications 1.1 and 2.1 are quite different since the first was developed based on the requirements of the Smart Home aiming at guiding the development team regarding the functionalities that the system needs to execute, while the second was generated based on the features which in some instances makes the specification confusing because of the vagueness of the requirements.

Anyway, the ReqSys tool generates a Feature Model that has limitations but it serves as a basis for adjustments in order to have a more appropriate model. Similarly, specifications generated from the Feature Models, serve as an initial release to be corrected when necessary. Thus, in situations where there is only one these artifacts ReqSys generates the other automatically saving the requirements engineer of the burden of developing the artifact from the scratch.

6. RELATED WORK

Alférez et al [1] presents two complementary approaches to variabilities and requirements management in SPL: (i) Semantics-based Variability Modelling, through extended Requirements Description Language (RDL) which maintains semantic links between variabilities and requirements; and (ii) Variability Modeling Language for Requirements (VML4RE), a domain specific language that allows to specify and to relate variabilities to abstractions of requirements expressed in different models of requirements.

Silva et al [8] present the language i*- c, an extension of i* with support to cardinality for representation of the variability in SPL, in addition the approach G2SPL (Goal to Software Product Line) that allows to identify features from i* models and configuration of products within the SPL.

Both works have defined languages that allow representing variabilities and some mechanism to relate these variabilities to the requirements. Our work is similar to them once it defines the extension of AOV-Graph for SPL aiming to fully represent the variability. But our work also developed a bi-directional mapping between PL-AOVGraph and the Feature Model, automated by the ReqSys tool. We consider that the Feature Model and PL-AOVGraph are essential for the development of a SPL since they present different complementary views. Therefore, these models must be used simultaneously.

7. Final Remarks

In this work we presented PL-AOVGraph and a bidirectional model between the Feature model and PL-AOVgraph. We presented the transformation rules that allows the association between PL-AOVGraph and the Feature model. Such rules were implemented by the ReqSys plug-in that automates such activity. We used a well-known case study to analyze and validate the mapping mechanism.

REFERENCES

[1] Alférez, M. et al. A Metamodel for Aspectual Requirements Modelling and Composition. AMPLE Project Deliverable D1.3, September 2008.

[2] Brown, L. et al. A widget framework for augmented interaction in SCAPE. In Proceedings of the 16th Annual ACM Symp. on User interface Software and Technology Canada, 2003. UIST '03. ACM Press, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697

[3] Filman, R. E. et al. Aspect-Oriented Software Development. Addison-Wesley, 2005.

[4] Neiva, D. F. S. Engenharia de Requisitos para Linha de Produto de Software. Recife, 2008. Tech. Report – UFPE.

[5] Rocha, D. K. F. Visualização de Requisitos a partir de modelos AOV-Graph. Natal, 2009. 84p. Tech. Report. UERN

[6] Sánchez, P. et al. A Metamodel for Designing Software Architectures of Aspect-Oriented Software Product Lines. AMPLE Project deliverable D2.2, September 2007.

[7] Santos, L. O. PL-AOVgraph: Uma extensão de AOV-Graph para Linha de Produto de Software. Natal, 2010. 84p. Technical Report UERN. https://sites.google.com/site/plaovgraph.

[8] Silva, C. et al. G2SPL: Um Processo de Engenharia de Requisitos Orientada a Objetivos para Linhas de Produtos de Software. In: 13th Workshop on Requirements Engineering (WER 2010), 2010, Cuenca. Proceeding of the 13th Workshop on Requirements Engineering, 2010. p. 1-11.

[9] Silva, L. F. Uma Estratégia Orientada a Aspectos para Modelagem de Requisitos. Rio de Janeiro, 2006. 222p. PhD Thesis - PUC-Rio.

[10] Silva, L. et al. On the Role of Features and Goals Models in the Development of a Software Product Line. In: International Workshop on Early Aspects at AOSD'10, 2010.

[11] Tomás, M. R. S. et al. SMART HOME. Technical Report. Universidade Nova de Lisboa, 2009. http://subversion.assembla.com/svn/erdssmarthome/Relatorios/SmartHome.pdf