An Approach to Support a Flexible Feature Modeling

Eldânae Nogueira Teixeira[1], Cláudia Werner[1], Aline Vasconcelos[2]

[1]Federal University of Rio de Janeiro
COPPE - System Engineering and Computer Science
P.O. Box 68511 - Rio de Janeiro, RJ 21945-970 Brazil

[2]Federal Fluminense Institute (IF-Fluminense)
Dr. Siqueira Street, 273 – Dom Bosco
Campos dos Goytacazes, RJ 28030-130 Brazil

{danny, werner}@cos.ufrj.br, apires@cefetcampos.br

Abstract. *The variability of a domain in software reuse approaches can be specified through feature modeling. This modeling can be represented with different notations, that encompass some concepts with the same semantics, regardless of the provided graphics and nomenclatures .In this work it was performed a domain analysis of three meaningful notations achieving a mapping of the concepts and properties from one notation to another. The goal was to achieve flexibility in variability modeling applying the results of the study in the Odyssey environment, a software reuse infrastructure based on domain models. The Odyssey adaptation allowed it to represent different feature notations and the possibility of transitioning between them.*

1 Introduction

Domain Engineering (DE) [1][2], and one of its variants, i.e., Software Product Line (SPL) [3], are key approaches to support software reuse, aiming to accomplish it in a systematic way at all stages of software development. Both techniques incorporate the Domain Analysis (DA) phase. This activity consists of collecting information and knowledge about a class of systems (the domain), to exploit its commonality and variability.

The results of Domain Analysis can be captured and represented in a domain model, a high-level description of the system family. Several modeling approaches have been developed and can be applied to represent the variability of a domain. Feature modeling, one of the representation techniques most used in these approaches, was originally proposed as part of the Feature-Oriented Domain Analysis (FODA) method [4], and since then, it has been applied in a variety of approaches. This modeling seeks to express domain requirements as features, which can be specified as prominent or distinctive, and user-visible aspects, qualities, or characteristics of a software system or systems [4].

First, this modeling helps in defining the scope of the class of systems or domain, identifying relevant characteristics, which should be retained or discarded. Later, the points and ranges of variation captured in feature models need to be mapped to a common architecture that is representative for the family of systems.

Feature modeling can be expressed with different notations, which might be chosen considering several factors such as higher adequacy to the modeling requirements, greater knowledge of the development team, popularity etc. Also, an appropriate Software Development Environment (SDE) should provide feature modeling, with support for reuse, aiming at achieving efficiency and adequacy to the development process. However, the tools and environments available have several deficiencies and, frequently, do not completely fulfill the users' need. In general, they do not offer the opportunity to choose a more appropriate notation, being limited to the concepts and properties offered by a single notation used in the SDE. Also, most environments do not support the different steps of the reuse process. Moreover, they frequently present representation deficiencies, with limitations of the graphical and visualization aspects.

Therefore, the goal of this work is to achieve modeling flexibility in a software reuse environment. It was conducted in the Odyssey SDE [5], a reuse environment based on domain models, which modeling structure was fixed, providing support only for the Odyssey-FEX notation [6], its proprietary notation. This environment supports all phases of software reuse, encompassing the development for reuse through a Domain Engineering process, and development with reuse through an Application Engineering process. Moreover, it provides model consistency checking through its plugin Oraculo, allowing application instantiation with a certain degree of reliability.

Initially, two other meaningful notations, referenced in the literature were added to the Odyssey environment, in order to evaluate the proposed approach for modeling flexibility and the possibility of transitioning between the feature notations, i.e. the notation proposed by Czarnecki *et al*. [7][8], and the one defined by Gomaa [9]. Odyssey adaptation required a detailed study of the concepts encompassed by each notation in order to identify their similarities and differences. Therefore, a Domain Analysis for feature modeling was conducted. Also, the feasibility to represent different feature notations and the possibility of transitioning between them were accomplished by studying the environment modeling structure.

The rest of this paper is organized as follows. Section 2 reviews some concepts related to variability and feature modeling; the proposed approach is presented in Section 3; Section 4 details the adaptations that were made in order to apply the notation flexibility mechanism in the Odyssey environment; Section 5 summarizes some meaningful related work; and, finally, conclusions and future work are presented in Section 6.

2. Background

Domain Engineering is the process of identifying and organizing knowledge about some class of problems – the problem domain – to support the description and solution of those problems [10]. During Domain Engineering, the commonality and the variability of the product family is defined. Shared assets are implemented so that the commonality can be exploited during Application Engineering. During Application Engineering, individual, customer specific software products are ideally developed by selecting and configuring shared assets resulting from Domain Engineering [11].

The concept of software variability seeks to explore the benefits that exist in the similarities found in a family of systems and to manage its diversity. It is the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context [12]. The variability concept is defined by the introduction of the so called variation points. A variation point defines a decision point together with its possible choices (functions or qualities). The available functions or qualities for a variation point are called variants [13].

The notation for variability modeling can be graphical, textual or a combination of both forms of representation. However, there appears to be some consensus that there is a relation between features and variability, in that variability can be more easily identified if the system is modeled using the concept of *features* [12]. A major advantage of discussing a system in terms of features is that they bridge the gap between requirements and technical design decisions [12].

The feature modeling technique aims at capturing and managing the similarities and differences in order to facilitate the understanding of users, domain specialists and developers with regard to the general capacities of a domain, which are expressed by features. Therefore, this model provides the basis for the development, configuration and parameterization of reusable artifacts [14].

3. Feature Modeling Flexibility Approach

As mentioned before, feature modeling may be performed by applying distinct notations. However, among the several available representations, some concepts have the same semantics, regardless of the provided graphics and nomenclatures. The comprehension of these different alternatives of notations is the basis for the development of an approach that involves some kind of relationship between them.

In this work, it was decided to perform a detailed study of the concepts encompassed by three meaningful notations, referenced in the literature. A domain analysis of the notations was performed in order to identify their similarities and differences. The goal was to identify which concepts have the same semantics and which are particularities of a notation and influence the representation of the domain.

As a result of this study it was possible to establish a mapping of the concepts and properties from one notation to another. The impacts of the transformations were also evaluated and resulted in the identification of some loss of information that occurs due to limitations imposed by the set of elements covered by a particular notation.

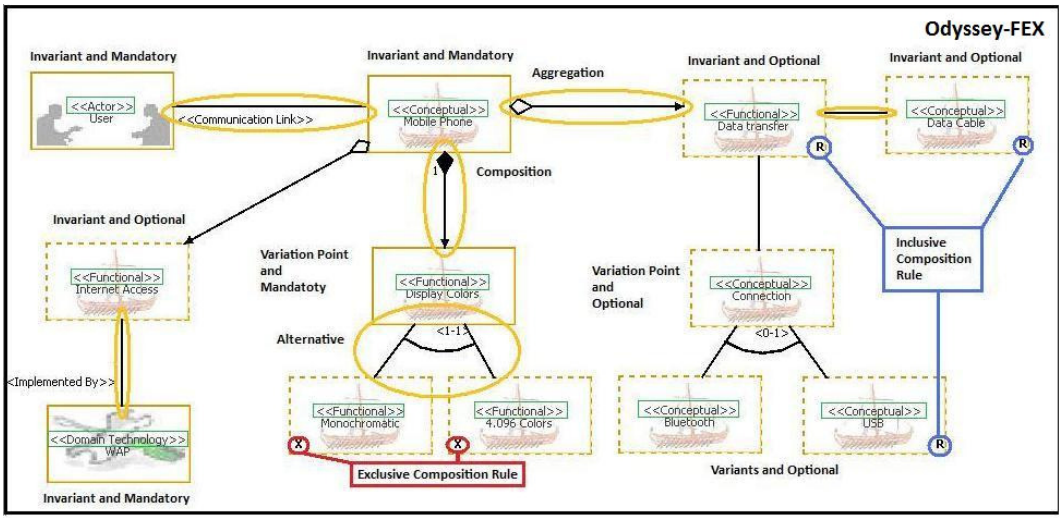
3.1. Comparative Study

This work involves the Odyssey-FEX notation [6], a proprietary notation used in the Odyssey environment, the notation proposed by Czarnecki *et al*. [7][8], and the one defined by Gomaa [9].

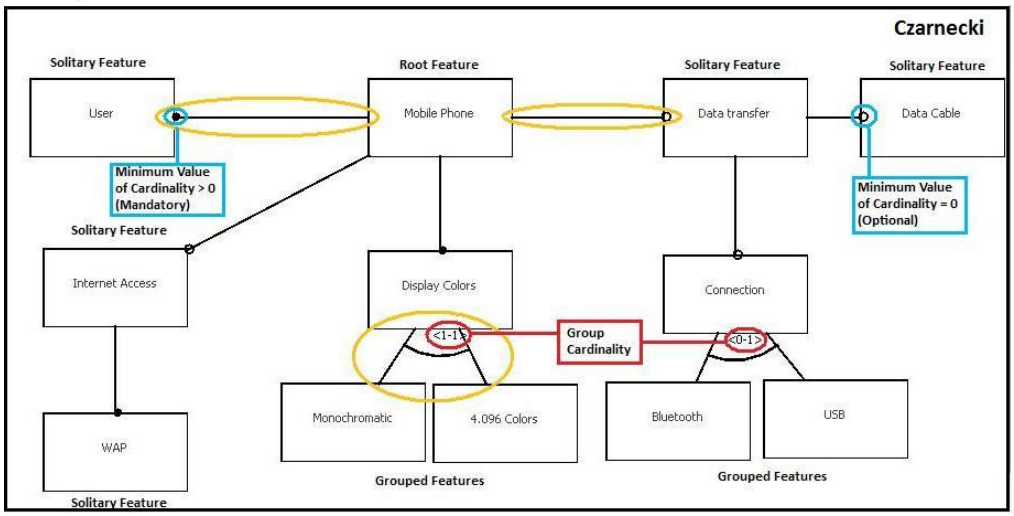
The study was divided into three classes of concepts: (1) feature taxonomy, (2) dependency and mutually exclusive feature relations, and (3) other feature relationships. The core properties of each notation were identified and a mapping between concepts semantically equivalent was established.

To illustrate the important concepts discussed in this section, we will use an example of the mobile phone domain represented in the three different studied notations, using the Odyssey environment (Figure 1).

(A)



(B)



(C)

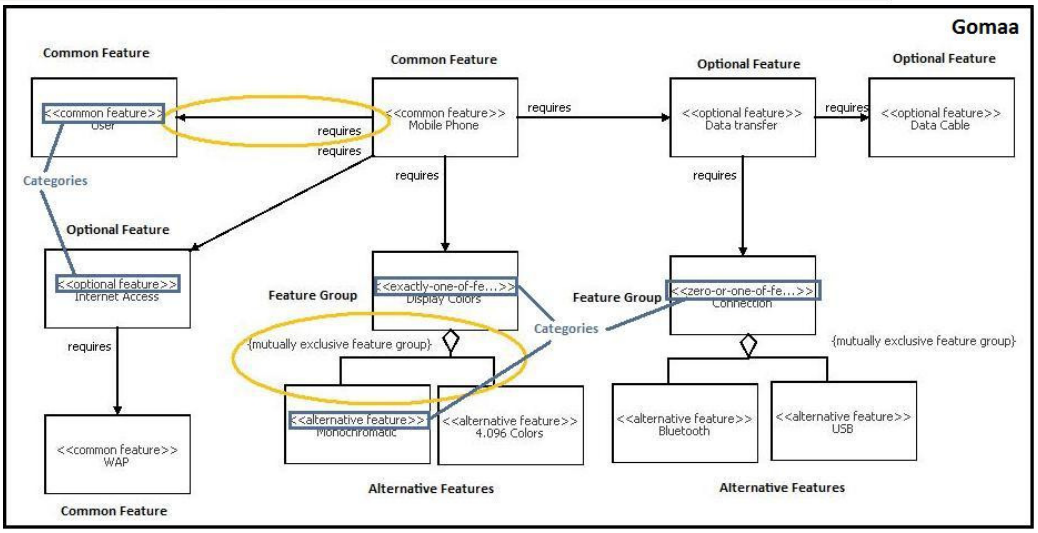


Figure 1. The Mobile Phone domain in the 3 notations (Figure 1 – A: Odyssey-FEX notation; Figure 1 – B: Czarnecki's notation; Figure 1 – C: Gomaa's notation).

The first class of concepts deals with the feature taxonomy and the correspondence between the multiple classifications and feature properties among the notations. The Odyssey-FEX notation has multiple feature categories, related to the different phases in the software life cycle. In this notation, a feature can be classified as a functional, conceptual or entity feature, which represents the analysis phase. Also, a feature can be classified as operational environment, domain technology or implementation techniques, which represents the design phase and technological aspects. These categories are explicitly represented by stereotypes related to each feature, as shown in Figure 1 - A. In this example, *Mobile Phone* and *Data Cable* features are examples of conceptual features, and the *WAP* feature is classified as a domain technology feature related to the *Internet Access* feature, a functional one. There is not any equivalence for these classifications in the other notations, where the classification is mostly based on the variability and optionality concepts. The Odyssey-FEX notation also represents the variability and optionality concepts, but this is an orthogonal classification associated to the other feature categories. The optionality concept is represented by a dashed shape in Odyssey-FEX notation and by empty circles in the links between features in Czarnecki's notation. In Gomma's notation, this concept is treated using stereotypes that classify the features as common or optional features. In Figure 1, the feature *Display Colors* is a so-called mandatory feature, as well as the *Mobile Phone* feature and the WAP feature. The concept of variation point can be exemplified by the feature *Connection*, and its variants, *Bluetooth* and *USB*.

The variability concept associated to the optionality concept resulted in the mappings listed in the Table 1.

Odyssey-FEX	**Czarnecki**	**Gomaa**
Taxonomy - Variability and Optionality concept		
Optional *Variation point* with cardinality value as: <0,1>/ <0,k>, k>0	Feature Group	<<zero-or-one-of-feature group>>/<<zero-or-more - of-feature group>>
Mandatory *Variation point* with cardinality value as: <1,1>/<n,k>, n>0, k>1	Feature Group with minimum cardinality value as: <1,1>/<n,k>, n>0, k>1	<<exactly –one-of-feature group>>/ <<one- or-more - of-feature group>>
Variant in a variation point with only optional variants and maximum cardinality value as one or exclusive composition rules between all its variants	Grouped feature in a variation point with maximum cardinality value as one (<0,1>/<1,1>)	<<alternative feature>>
Variant in a variation point with only optional variants and cardinality value as: <0,k>, k>1	Grouped feature in a variation point with cardinality value as: <0,k>, k>1	<<optional feature>>
Variant as a default feature	This concept does not have any mapping	<<default feature>>

Table 1. Variability and optionality concepts

In the Czarnecki's notation it is possible to address a feature as a root feature, since the model is represented by a tree, in a hierarchical structure. Also, Gomma's notation has a tree model and the concept of root can be found. In the example of Figure 1, the *Mobile Phone* feature is classified as a root feature in Czarnecki's notation, but there is not any graphical representation which emphasizes this. This semantic has not the same importance in Odyssey-FEX, which uses an acyclic graph to represent the model.

The definition of parameter values, during the configuration phase in a SPL, is a possibility available in Czarnecki and Gomaa's notations. It is possible to define an attribute, its type and its default value for a feature. In Gomaa's notation, it is also possible to determine a range of values related to an attribute of a feature which is classified as parameterized feature.

The Odyssey-FEX notation also provides additional properties related to a feature, such as the representation of an external feature, a not yet defined feature, or an organizational feature. These properties are not represented by the other notations. An external feature is one related to other domains, expressing the domain interfaces. A not yet defined feature is an identified feature in the domain which is not yet refined at other model abstraction levels. And the latter, i.e., the organizational feature, is just to ease the domain understanding or its organization, not being concretely related to a real domain use.

Table 2 presents a summary of concepts related to the categories and properties presented in each notation and the correlations between them.

Odyssey-FEX	Czarnecki	Gomaa
Taxonomy / Categories		
Classifications by analysis and design phases	This concept does not have any mapping	This concept does not have any mapping
This concept does not have any mapping	Feature with a definition of an attribute (type, default value)	<<parameterized feature>>
This concept does not have any mapping (model as an acyclic graph)	Root Feature (model as a tree)	Common feature which the other model features are extended
Taxonomy /Properties		
Name	Name	Name
Layer (domain or technology)	this concept does not have any mapping	This concept does not have any mapping
Other classifications: Feature not-defined/ External feature/ Organizational feature	This concept does not have any mapping	This concept does not have any mapping

Table 2. Concepts related to the Taxonomy class of concepts

The second class deals with the concepts related to feature dependency and mutually exclusive feature relationships, which are summarized in Table 3.

Odyssey-FEX notation best represents these restrictions by two types of composition rules: inclusive and exclusive. Inclusive rules represent feature dependencies, which means that the features involved in the rule should be selected together. Exclusive rules represent mutually exclusive feature relationships. In this case, the features must not be selected together for the same product. These rules can be combined with boolean expressions, which can be composed by two or more features, forming an expression combination. The feature dependencies can also be represented by the dependency relationship and it is graphically represented by an arrow linking the features as in the UML notation. The limitation of this representation is the relation involving just two features in contrast with the multiple possible combinations possible with the inclusive rules. In the other notations, this concept is represented by other means. In Gomaa's notation, relations, called require and mutually inclusive relationships, are used to represent the concept of dependency, and types of features to represent the concept of mutually exclusive relations. In Czarnecki's notation, values of cardinality are used to represent these concepts. The major weaknesses of these two last notations related to the composition rules is the possibility to establish a relationship involving more than two features, which is not possible in Czarnecki and Gomaa's notations. For example, in Figure 1 - A, the require composition rule, *Data Transfer and USB requires Data Cable* can be visualized by the mark, represented with "R" in the features involved by the rule.

Odyssey-FEX	**Czarnecki**	**Gomaa**
Dependency and mutually exclusive relation		
Mutually Exclusive relation between variants		
Variation point with maximum cardinality value as one or using exclusive composition rule between all its variant	Or-exclusive feature group <1,1>/ Or-exclusive feature group <0,1>	<<exactly-one-of-feature group>>/ <<zero-or-one-of-feature group>>
Mutually Exclusive relation between invariants		
Exclusive composition rule between the invariants	This concept does not have any mapping	This concept does not have any mapping
Dependency relation		
Inclusive composition rule/ Dependency relationship	This concept does not have any mapping	Require relationship/ Mutually inclusive relationship

Table 3. Main results of the dependency and mutually exclusive relations class of concepts

The third and last class deals with the concept of other feature relationships. Table 4 presents the correlation between these relationships of the notations.

It is important to note that Odyssey-FEX notation provides both UML relationships, e.g. dependency, association, composition and generalization, and features

relations, e.g. the alternative relationship, which represents the relationship among a variation point and its variants. In Figure 1 – A, there are some examples of these relationships, such as an association relationship between the *Data Transfer* and *Data Cable* features, an aggregation relationship between the *Mobile Phone* and *Data Transfer* features, a composition relationship between *Mobile Phone* and *Display Colors* features. Also, there is an alternative relationship between the *Display Colors* variation point feature, and its variants, *Monochromatic* and *4.096 Colors*. This expressiveness obtained by all these relations is not achieved by the other notations, which do not provide all of the mentioned relationships. The authors of Odyssey-FEX notation believe that by applying rich semantics to feature relationships, it is possible to achieve greater capacity of representation and expression for domain semantics. In Czarnecki's notation, the relations between features do not have any semantic associated, they are just links between features, which can be noticed in Figure 1 - B. But the notation has a relation that links different models, called feature reference relationship. Gomaa's notation has only the two dependency relations, i.e. require and mutually inclusive relationships, which can be visualized in Figure 1 - C. But all the notations have the concept of group provided by the link between the variation point and its variants.

Odyssey-FEX	**Czarnecki**	**Gomaa**
Relationships		
Transitions: Odyssey-FEX to Czarnecki / Odyssey-FEX to Gomaa		
UML relationships (e.g. Composition, Aggregation, Generalization, Association)	Common link between the features (not the same semantic – standard mapping defined by the study[1])	Require relationship (not the same semantic – standard mapping defined by the study[1])
Odyssey-FEX proper relations: Implemented by and Communication Link	Common link between the features (not the same semantic – standard mapping defined by the study[1])	Require relationship (not the same semantic – standard mapping defined by the study[1])
Alternative relationship (relation between variation point and its variants)	Relationship between the feature group and its grouped features	Relationship between the feature group and its variants
Transitions: Czarnecki to Odyssey-FEX / Czarnecki to Gomaa		
Association (not the same semantic – standard mapping defined by the study[1])	Feature Reference	Require relationship (not the same semantic – standard mapping defined by the study[1])

Table 4. Main results of the other relationships

The result of this study reflects the multiple alternatives to model a domain using the technique of feature modeling and the diversity of notations. It also emphasizes the differences between the notations and their properties, and highlights some aspects to be

[1] In this case a loss of information occurred and the study determined a standard correlation between the notations to permit the minimal representation of the information in the target notation.

considered in order to choose a specific notation. This choice is often associated to the possibility of support by a SDE, which provides the notation and support for the reuse process. Thus, it is important that a certain modeling flexibility can be offered by an SDE, so development teams can choose the notation by their preference or adequacy to the project, not by imposition of availability in a SDE. Motivated by the lack of tools and environments which could provide notation flexibility, this work extended the Odyssey SDE in this regard, by adapting it to provide the three feature notations mentioned before.

4. Modeling Flexibility in the Odyssey SDE

Odyssey development environment [5] is a software reuse infrastructure based on domain models, which provides automated tools to support the distinct phases of a reuse process. It has a hierarchical internal structure represented through a semantic tree of objects. This is organized by categories of models composed by different modeling items. The work was focused on the feature model, known in Odyssey as "Feature View", which used to be fixed, providing support only for its proprietary Odyssey-FEX notation. The goal of this work was to perform the adaptations in the Odyssey structure, allowing it to represent different feature notations and the possibility of transitioning between them.

Its structure has been transformed, creating a basis with the common elements among the notations, i.e. the Feature Base. This solution attempts to better structure the concepts inside the environment, defining a transparent boundary among the particularities of each notation and a conceptual base for sharing similarities. The implementation was based on the State Pattern, which allows an object to change its behavior when its internal state changes [15], and can be visualized in Figure 2.

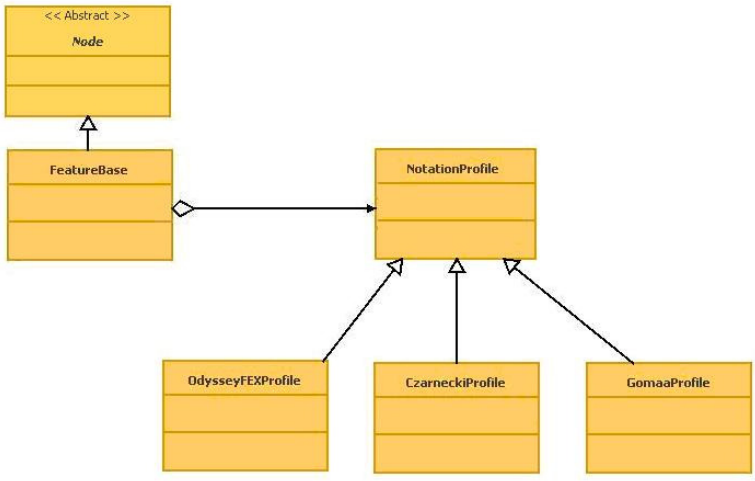


Figure 2. Structure implemented in the Odyssey SDE

FeatureBase class saves an instance of the current state, named in this approach as Notation Profile, which represents the notation that is being used at the moment within the Odyssey SDE. The *NotationProfile* corresponds to a superclass of common behavior encapsulation associated with profiles. The number of profiles is the number of notations represented in the environment. Each one combines the particularities of a

notation, i.e. its specific behaviors. As the developer changes the notation in the environment, the profile also changes to adapt its behavior and provide the information related to the notation in use.

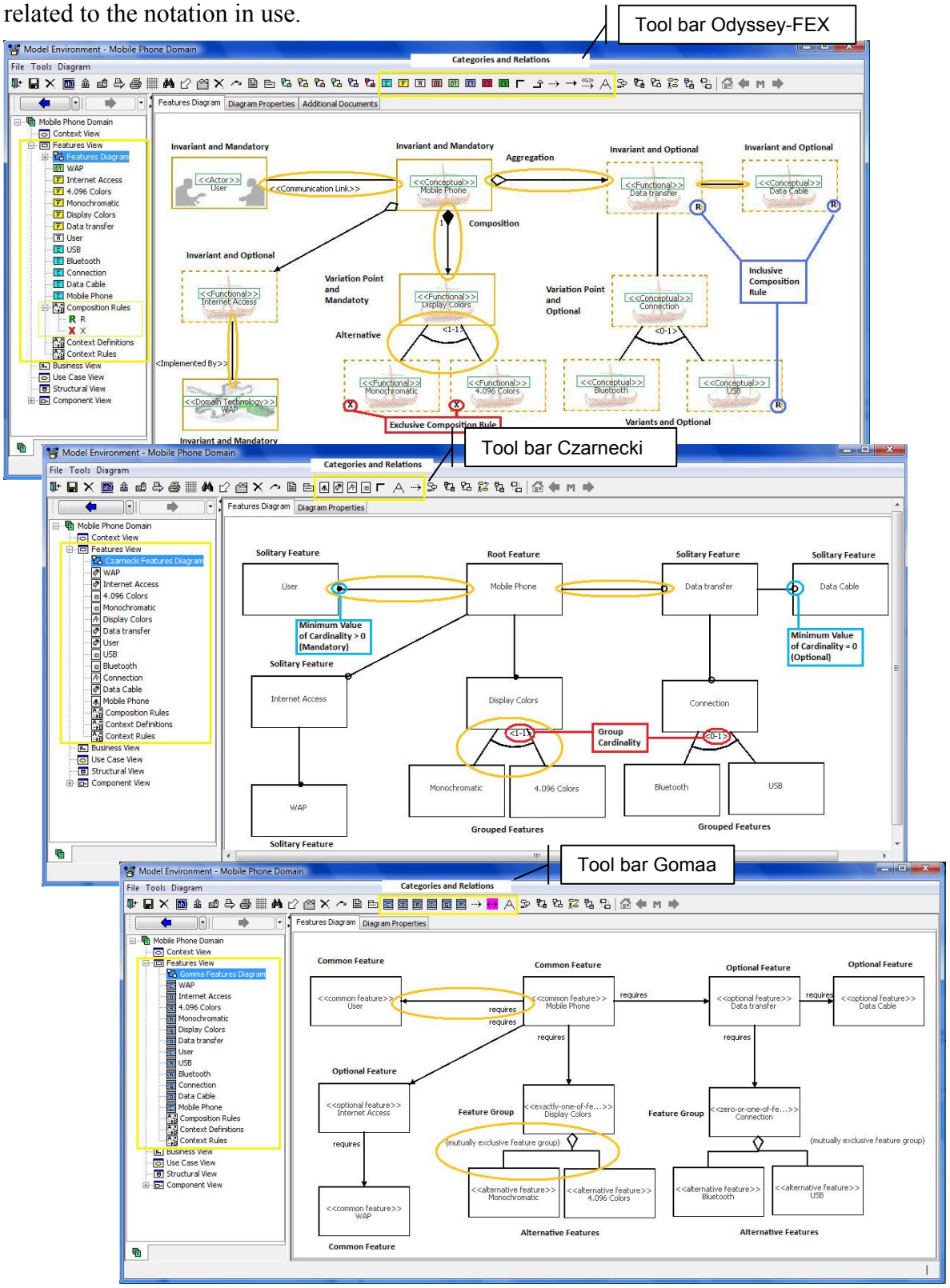


Figure 3. Different views of the Odyssey SDE adapted to the three notations

Then, the environment structure is divided into packets responsible for conceptual representation and graphical representation. The adaptation of the semantic structure was described in the above paragraphs. The structures related to the visualization of the elements were also adapted. The Lexical structure is related to the graphical representation of each element, while the Presentation structure is intended to support graphical interfaces for configuration of the semantic elements. These structures were also adapted to provide a specific panel with a custom tool bar according to the notation in use, as shown in Figure 3. For each notation, different adjustments are dynamically performed to adapt the environment and its diagramming tool to provide the actions and the elements needed to build the model in accordance with the notation selected by the user. The semantic and graphical representations applied in the environment can be visually noticed in Figure 3.

The first part of Figure 3 represents the initial model. The other two represent the results of the transitioning process from one notation to another provided by the environment. This process intends to map the concepts related to the two notations involved in the process, i.e. the current notation and the target one. The process is divided in three steps, namely: (1) presentation of information about the standard mappings and the possible loss of information identified by the approach; (2) an interaction between process and user, altering options in the standard mappings; (3) conclusion of the process. In the example of Figure 3, the domain was firstly modeled in the Odyssey-FEX notation. Then a transitioning process transformed it into the Czarnecki's notation and to finish it was transformed into the Gomaa's notation.

5. Related work

Some environments were evaluated considering their support to feature modeling and reuse. In Antkiewicz & Czarnecki [16] a summary of meaningful related work is presented. One of them is the AmiEddi the first editor to support the notation for feature modeling described in Czarnecki & Eisenecker [17]. It does not include cardinality. His successor, CaptainFeature includes cardinality and uses a feature diagram based on properties defined by the Czarnecki's notation [8]. Another tool, the ConfigEditor, was an initial prototype, allowing application configuration based on domain features. This functionality was later integrated into the CaptainFeature tool. However, these tools do not support feature notation configuration, as proposed in this work.

Another tool described in Antkiewicz & Czarnecki [16] is the ReqLine tool, a research tool which aims to integrate feature modeling with requirements engineering, not supporting cardinality, but allowing different types of relationships between features through a hierarchy. It also proposes model consistency checking and product configuration through feature selection. Although this tool provides consistency checking and feature selection for application instantiation, as Odyssey, it does not support feature notation selection and mapping.

Pure::Variants, also presented in the Antkiewicz & Czarnecki [16], is a commercial tool for feature modeling and configuration based on a tree structure. The tool does not support cardinality, but offers comprehensive modeling constraints between features based on constraints using Prolog.

Another tool, the Generic Modeling Environment (GME) [18], is a configurable modeling environment, developed at the Institute for Software Integrated Systems at Vanderbilt University. Its configuration can be achieved by using metamodels which specifies the modeling paradigm of an application domain, that includes syntactic and semantic information, i.e., concepts for construction of models, possible relationships between the concepts, how to organize and visualize concepts and how to determine rules to models construction. This tool, although providing feature modeling configuration by adapting it to the domain concepts, requires an additional effort to assemble its structure and add the functionalities that express the user needs. On the other hand, this work already provides a structure with support for three different notations, not requiring the user the need to configure the environment.

The analysis identified a fixed structure provided by these tools, which offer just one notation for feature modeling, being restricted to its concepts and properties. This drawback can limit the software development process, since the environment cannot attend to particular requirements of the modeling needs of a team or company. The GME tool, although configurable by a metamodel, requires that the user understands this structure to be able to assemble its structure.

In most of the environments, restrictions on the form of modeling representations are observed, some too rigid and difficult to be understood by the user. Also, most of them are only modeling environments and do not provide support to the various steps involved in a reuse process.

Therefore, a more flexible environment, which allows representing different feature notations and the possibility of transitioning between them, supporting all stages of software reuse, can be an important contribution for an adequate development process.

6. Conclusions

The goal of this work was to achieve modeling flexibility in order to minimize the constraints found in current reuse environments, helping developers in the modeling activity. It was developed within the Odyssey SDE, including the study of its structure and the concepts related to different notations of feature modeling.

The establishment of concept relations between the three notations addressed in this study can be highlighted as an important contribution. The work demonstrates the feasibility of multiple notations for feature modeling in a reuse environment, offering the user a choice according to his criteria of adequacy and allowing the transition between notations. Also, the approach enables extensions for future incorporation of new notations, where the particularities of the notation will be included as a new profile associated in the structure implemented in the Odyssey SDE.

Some limitations and needs of extensions have been identified, being considered opportunities for future work. An example can be the extension of the Application Engineering, an available functionality in the environment, in order to deal with the new incorporated notations. Other possibility is to extend the criticism mechanism reachable in the Odyssey environment. The goal is to allow the model consistency verification in an automatic manner, verifying the need to adapt the set of rules of model formation to

meet the modeling of features represented by the other notations on the environment, beyond the notation Odyssey-FEX. Another extension is to offer to users the possibility of describing their own notation, by the decomposition of features of each notation elements.

Also, the plan and implementation of evaluation studies more complete has to be included in the future works, involving the proposed approach as a whole and the use of the environment. As well as get the opportunity to apply the approach and tool in the development of LPS real and large. Thus, it is possible to identify opportunities for development and improvement of the approach and the extension of the environment with possible desired notations.

The study of some existing notations for feature modeling, associated with the identification of fundamental concepts in the variability modeling, may be a first step towards understanding the needs of representation and mappings for the existing representations, which can culminate in the unification of feature notations and a consensus in this regard. As it has occurred with other areas, we hope to see an effort in this direction for feature modeling.

Acknowledgments. The authors would like to thank CAPES, CNPq and FAPERJ for the financial support.

References

1. Arango, G. F. "*Domain Engineering for Software Reuse*." PhD thesis, University of California at Irvine, 1988.
2. Griss, M. L., Favaro, J., D'Alessandro, M., 1998, "Integrating feature modelling with the RSEB". In: Proceedings of Fifth International Conference on Softwre Reuse - ICSR5, pp. 76-85 Victoria, British Columbia, Canada.
3. Northrop, L., "SEI's Software Product Line Tenets", IEEE Software, v.19, n. 4 (July/August, 2002), pp. 32-40.
4. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S., 1990, "Feature-oriented domain analysis (FODA) feasibility study". Technical Report CMU/SEI-90TR -21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1990.
5. Odyssey, "Odyssey SDE Homepage", http://reuse.cos.ufrj.br/odyssey
6. Fernandes, P., Werner, C., 2008, "Ubifex: Modeling context aware software product lines". In 2nd International Workshop on Dynamic Software Product Line Conference, Limerick, Ireland, 2008, pp. 3-8.
7. Czarnecki, K., Helsen, S., Eisenecker, U., "Staged Configuration using feature models"., Software Product Lines: Third Internacional Conference, SPLC , Proceedings, v. 3154, Boston, MA, USA, August 30-September 2, 2004, pp. 266-283.

8. Czarnecki, K., Helsen, S., Eisenecker, U., "Formalizing cardinality based feature models and their specialization", Software Process: Improvement and Practice, v.10, n.1 (March, 2005), pp. 7-29.

9. Gomaa, H., "Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures", Addison-Welsey, 2004.

10. G. Arango, and R. Prieto-Diaz, "Introduction and Overview: Domain Analysis Concepts and Research Direction", G. Arango R. P.–D.A. (eds), Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, 1991, pp. 9-25.

11. G. Halmans, K. Pohl, "Communicating the variability of a software-product family to customers". Software and Systems Modeling, 2(1) 2003 15-36.

12. Svahnberg, M., Van Gurp, Bosch, J., "A Taxonomy of Variability Realization Techniques". ISSN: 1103-1581, Blekinge Institute of Technology, Sweden, 2002.

13. Halmans, G., Pohl, K., "Communicating the variability of a software-product family to customers". Software and Systems Modeling, 2(1) 2003 15-36.

14. Lee, K., Kang, K., Lee, J., "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering", Software Reuse: Methods, Techniques, and Tools : 7th International Conference, ICSR-7, Proceedings, Austin, TX, USA, April, 2002, pp. 62-77

15. Gamma, E., Helm, R., Johnson, R., Vlissides, J., " Design Patterns: Elements of Reusable Object-Oriented Design". Addison-Wesley, Reading, MA, 1995.

16. Antkiewicz, M., Czarnecki, K., "FeaturePlugin: feature modeling Plug-in for Eclipse", OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, Vancouver, British Columbia, Canada, Oct. 24-28, 2004. pp. 67-72.

17. Czarnecki, K., Eisenecker, U.W., 2000, "Generative programming: Methods, Tools, and Applications", Addison-Wesley, Boston, MA.

18. Lédeczi, A., Maróti, M., Völgyesi, P., "The Generic Modeling Environment", Institute for Software Integrated Systems, Vanderbilt University, Nashville, USA, 2001.