Feature Modeling for Context-Aware Software Product Lines

Paula Fernandes, Cláudia Werner, Leonardo Murta
Federal University of Rio de Janeiro
COPPE - System Engineering and Computer Science
P.O. Box 68511 - Rio de Janeiro, RJ 21945-970 Brazil
{paulacibele, werner, murta}@cos.ufrj.br

Abstract

One of the first activities to develop a software product line is the feature analysis. This activity produces a feature model to represent commonalities and variabilities among products of a product line. Context-aware applications use context information to provide services and relevant information for their users. One of the challenges to build a context-aware product line is how to represent context information in a feature model. This paper proposes a modeling notation, called UbiFEX, for representing context information and defining context adaptive rules in a feature model.

1. Introduction

Context-aware systems are part of a wide range of systems within ubiquitous computing [18]. These systems use context information to provide relevant services and information to the users. In different context situations, users may access different data and exploit different aspects of an application. For instance, when a tourist arrives to a new place and accesses a mobile tourist guide application in his smartphone, he expects that recommended tours are based on his preferences and location.

Software product line paradigm [13] explores commonalities and variabilities in a set of applications for a specific domain aiming at increasing productivity and quality. It this way, it proposes a systematic software development approach based on a product family. This approach guides building new applications from reusable assets and building the assets themselves. Furthermore, this paradigm has proved itself as an efficient way to deal with varying user needs [9].

Although, most approaches have focused on the development of statically configured products using core assets with variation points [7]. All variations are instantiated before a product is delivered to customers. Hence, these approaches provide development time adaptation in which different versions of application have been generated according to customers and runtime environment specific characteristics.

For example, a mobile application such as a tourist guide is intended to execute in a variety of mobile devices and customized to different user preferences. Thus, if development time adaptation is used, the number of versions will increase exponentially. Mobile devices have limited memory, storage and processing, and it is not always possible to load at the same time all necessary components to all runtime contexts.

This static focus is not adequate to deal with the dynamism of context-aware applications because they need a runtime adaptation in which applications adapt their behavior according to context changes. Context-aware application development should benefit from the software product line concept in terms of reusability and configurability. However, it introduces new challenges to software product line engineering [16].

One of the first activities to develop a software product line is the feature analysis. This activity identifies externally visible characteristics of products in a product line and organizes them into a model called feature model [11]. A feature is a system property that is relevant to some stakeholder and is used to capture commonalities and variabilities among products in a product line.

Feature modeling allows us to model the common and variable properties of product-line members throughout the stages of product-line engineering. Feature model is used since early stages for deciding which features should be supported by a product line and which should not until product derivation. Therefore, for a context-aware product line it is important to represent context information in this model. Without this representation we can not explicitly know how this information impacts in the feature selection at runtime. We noticed that the most part of well known feature model notations [4][8] do not deal with this aspect in an effective way and they are not concerned with separating this concept for a better understanding of this kind of systems.

This paper proposes an extension to a feature notation to represent context information explicitly in a feature model. Moreover, we analyze the influence of this kind of information in product variability and adaptive decisions. For this purpose, we define new types of features for representing context information in the model. We also define new rules to represent the relationship between features and contexts.

The remainder of this paper is divided into three sections in addition to this introduction. Section 2 reviews some basic concepts related to software product line development and context-aware systems, and summarizes some related works. Section 3 presents UbiFEX, a feature notation extension for modeling context-aware product lines. Finally, section 4 concludes this paper and discusses future directions.

2. Background

2.1. Software Product Lines

According to Software Engineering Institute (SEI), a software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way [13]. Core assets are the essence of a product line and represent configurable elements used to build derived applications.

During the product line development process, particular aspects of products can be highlighted. The variability concept refers to points in the core assets where it is necessary to differentiate individual characteristics of products, being represented with a feature model. In order to model variability, it is necessary to represent all domain concepts and their relationships explicitly.

A feature model represents a domain and aims to make homogeneous the concepts among the participants involved in the process, such as users, domain specialists, and developers. It represents the features of a system family, their commonalities and variabilities, and the relationships among them. In addition, it has a high level of abstraction and is used as a starting point for the feature selection to new products instantiation.

An important relationship in a feature model is the dependence among features that defines when some features should be included in de product due to the presence of other features. On the other hand, it may also define when some features should be removed from the product due to the presence of other features.

This kind of model has some important concepts: (1) variation points establish the necessity of decision-making related to one feature regarding which variants will be used; (2) variants are available choices for a variation point; (3) invariants mean fixed elements that are not configurable in the domain.

For example, considering the mobile tourist guide domain, we can have a functional feature "show map". However, maps can be represented in more than one form, such as an image or a 3-D map. In this case, we have a variation point with two variants (see Figure 1). However, we can also have a feature "list hotels" that has the same behavior for all products derived from the product line. In this case, the "list hotels" feature is considered an invariant.

2.2. Context-aware Systems

Context-aware systems are able to adapt their operations to a specific context without explicit user intervention. They use context information to provide relevant services and information.

Many definitions of context are given in the literature. Dey and Abowd [5] define context as any information that can be used to characterize the situation of an entity that is considered relevant to the interaction between a user and an application, including the user and the applications themselves.

Also, according to these authors, when dealing with context, three entities can be distinguished: places (e.g., rooms, buildings, etc.), people (e.g., individuals and groups), and things (e.g., physical objects, computer components, etc.).

In the literature [1], some attributes for describing a single context can be found: *context type* refers to the category of context; *context value* means the raw data gathered by a sensor; *time stamp* contains a date/time-value describing when the context was sensed; *source* describes how the information was gathered; and *confidence* describes the uncertainty of this context type.

2.3. Related Work

There are many approaches to support context-aware software development [12][6]. However, most of them are not concerned with a systematic software reuse. Their main focus is to solve specific problems for specific domains.

The approaches based on software product line usually propose a systematic software development based in a product family, which can beneficiate context-aware software development in terms of reusability and configurability.

Lee and Kang [10] proposed a feature-oriented approach to develop dynamically reconfigurable core assets for product lines. Features can be selected and configured at runtime. After feature analysis, feature model is refined through feature binding analysis that consists of two phases: feature binding unit identification, and feature binding time determination. Also, a dynamic binding relation is annotated with preconditions. The feature binding graph, generated after this analysis, provides an intuitive and visual description of dynamically changing product configuration.

However, mechanisms to support the definition of relevant context information used to describe preconditions are not identified and context information is not represented in the feature model.

Van der Hoek [17] presents the concept of any-time variability, which involves the ability of a software artifact to vary its behavior at any point in the life cycle. According to the author, a solution to support this kind of approach has to provide four main functionalities: a representation to capture system variabilities; a tool to specify these variabilities; a tool to resolve variabilities; and tools to apply the result of this resolution, in different life cycle points. This work does not exploit the way how the context information is identified. Furthermore, on the variabilities graphic representation, there are not elements which identify how this information influences the system dynamic configuration.

The approach proposed in this paper aims to provide solutions to the weak points found in these works, allowing an explicit representation of the relevant context information to the domain in the feature model and identifying how this information influences on the system dynamic configuration.

3. UbiFEX

UbiFEX is proposed to be a feature notation that provides context information representation and context rules specification. UbiFEX extends Odyssey environment [15] feature notation, called Odyssey-FEX.

This section is divided in four subsections. Section 3.1 presents the main characteristics of Odyssey-FEX notation, used as base for the proposed extension. Section 3.2 describes the features categories defined to allow context representation. Section 3.3 and 3.4, respectively, present expressions and rules used to represent the context influence in the dynamic product configuration. Finally, Section 3.5 introduces the initial ideas to build a context simulation infrastructure.

3.1. Odyssey-FEX

The Odyssey environment provides support to software reuse through domain engineering, product lines and component based development techniques.

Odyssey-FEX [14] was developed to fill some gaps in variability representation detected in other feature notations that could lead to an incorrect modeling of a system family. Some of these gaps are the lack of an explicit representation of variation points and an insufficient representation of dependency and mutual exclusiveness relationship among features.

In this notation, features must be represented according to three dimensions: category, variability, and optionality. There are five categories representing feature types: domain (functional and conceptual), entity, operational environment, domain technology, and implementation techniques.

Domain features are related to the core domain functionalities and concepts. Entity features are the model actors. Operational environment features represent attributes of an environment that a domain application can use and operate. Domain technology features represent technologies used to model or implement a specific domain requirement. Finally, implementation techniques features represent technologies used to implement other features.

According to variability classification, features can be variation points, variants or invariants. These concepts were previously presented in Section 2.1.

In respect to optionality, a feature can be mandatory or optional. This classification indicates whether a feature should be present in all products or not. The optionality refers to the whole domain. In Odyssey-FEX notation, optional features are represented in the model with a dashed shape. It is important to notice that an optional feature may become mandatory when other features are selected. This situation may occur due to the existence of composition rules among those features.

Composition rules define restrictions between features. Odyssey-FEX defines two types of composition rules: inclusive and exclusive. Inclusive rules represent feature dependency. For example, when the antecedent is selected the consequent has also to be selected. Exclusive rules represent mutually exclusive feature relationship. In this case, when the antecedent is selected the consequent must not be selected for the same product. These rules can be combined with boolean expressions. These expressions can be composed by more than two features, forming an expression combination both for a rule antecedent and consequent.

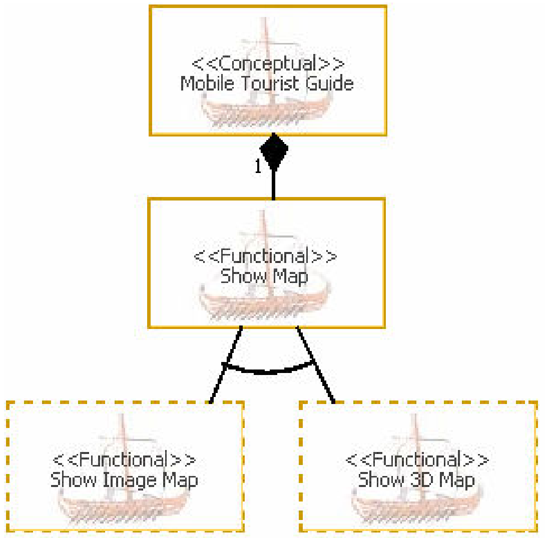

Figure 1. An example of Odyssey-FEX notation.

Odyssey-FEX applies relationship semantic in a feature model, offering a stronger capacity of representation and expression. Features are related to each other using UML relationships, such as association, generalization, and composition. In addition, the notation links a variation point to their variants through alternative relationships. This relationship expresses variability in the feature model.

Figure 1 shows a part of a feature model to a mobile tourist guide domain [3] built using Odyssey-FEX.

Odyssey environment also supports other feature model notations, including the ones proposed by Gomaa [8] and Czarnecki [4]. We decide to extend Odyssey-FEX due to its high level of expressiveness in respect to the other notations.

3.2. New Feature Categories

UbiFEX is proposed to represent context information in an explicit form. For that purpose, we defined two feature categories in addition of those described in the previous section: context entity and context information.

Context entity feature was created to represent relevant context entities for the domain. This relevance is based on the influence of the entity on the system behavior. For example, the mobile tourist guide system has entities such as user, mobile device, or a specific environment. The properties defined to a context entity feature are: name and description.

Context entities can be characterized by context information. Context information feature represents the data that should be collected to describe a context entity, which are relevant to domain applications adaptation.

Based on the attributes described in Section 2.2, we defined a set of properties to this type of feature: name, description, type (i.e., static or dynamic), base type (e.g., string, integer, etc.), initial value (when applicable), and source. Some of these properties are application specific, thus, they can be filled in during the domain analysis or after the initial feature selection to derive an application, according to the modeler decision.

If context features are modeled together with other feature types, the final variability model may be polluted with different concerns, compromising understandability. For this reason, we recommend the use of a separated model for context feature modeling. In this case, relationships between the models are represented by context rules. Odyssey environment allows filtering features in different types of diagrams according to their categories. In this way, we may have different views of a single feature model.

Figure 2 shows a simple context model to mobile tourist guide domain, representing the entities and context information.

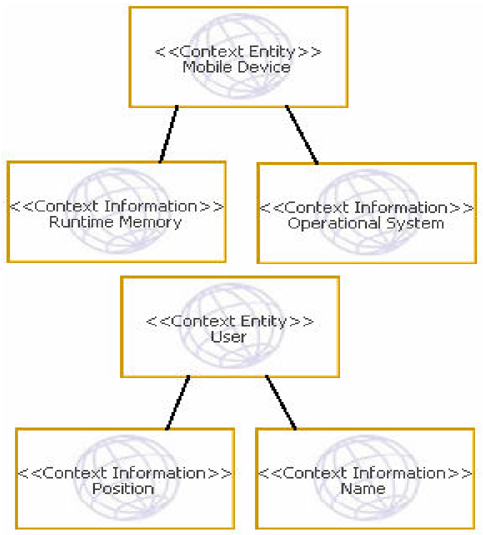

Figure 2. A context feature model.

3.3. Context Definition Expressions

After modeling context entities and information, the next step is the definition of the contexts that are necessary to create the context rules. Context definition is composed by a name and an expression.

```
<context-definition> ::= <expression>

<expression> ::= <CIF><relational-operator> <value>
        | <expression> <logic-operator> <expression>
        | NOT <expression>

<relational-operator> ::=  > | < | >= | <= | = | <>

<logic-operator> ::=  AND | OR | XOR

<value> ::= <string type> | <int type> | <float type>
        | <boolean type>
```

Figure 3. BNF for context definition.

A context can be defined according to the BNF notation (see Figure 3). An expression can be formed by a context information feature (<CIF>), previously described in the feature model, a relational operator, and a value. Also, an expression can be a composition of expressions using logic operators.

Figure 4 illustrates an example of a context definition expression for the mobile tourist guide domain.

```
(Runtime Memory > 64) AND
(Operational System = "Symbian")
```

Figure 4. Example of context definition expression.

In this way, we can define a context named Enough Memory associated with the first expression. A context is active when the evaluation of its expression is true.

3.4. Context Rules

Context rules specify how a specific context affects an application configuration in the domain, determining, for example, the decision about variant selection in a variation point.

The construction of a context rule is similar to a composition rule. The rule is formed by an antecedent, the operator *implies*, and a consequent. The antecedent is an expression that can contain contexts, features, and logic operators. The operator *implies* means that if the antecedent is true, then the consequent is selected. The consequent is an expression that can contain features and logic operators. For the domain used in this paper, Figure 5 shows an example of a context rule.

This rule can be used to optimize functionalities. For example, when the tourist wants to visualize a city map, the application analyses the active contexts and rules and choose the better variant to the runtime context.

Enough Memory **implies** Show 3D Map

Figure 5. Example of a context rule.

The context feature model with context definitions and context rules can be exported to an XML file. This file can be used as input to generate, for example, a middleware configuration file. These rules are also represented in the feature model. Features that are part of the consequent in some rule are marked with the rule identifier. In this way, it is easier to identify which features have been influenced by context.

3.5. Context Simulation

Starting from these proposed extensions, we are working on a tool to simulate variations in the defined contexts to analyze the behavior of the product line architecture. The Odyssey environment domain engineering process works with component-based architectures and it is possible to map features to application components.

In this way, we can identify inconsistencies between composition rules and context rules. The advantage is that it can be done at development time, reducing the cases where there is no possible product configuration and the application will probably fail.

For example, consider the previous mentioned mobile tourist guide domain. The simulation follows the following steps. First, we have to choose an initial configuration for the product, selecting the initial features. After that, we can simulate variations in the context information features values, such as Runtime Memory, and determine which contexts are active based on context definition expressions. The next step is the context rules analysis, to check if there are modifications on the product configuration. If modifications are found, a new set of features is selected and the consistency of composition rules is checked. For a complete simulation, these steps have to be repeated for each new generated configuration.

This simulation can run on-line, immediately notifying inconsistencies, or in batch mode, enacting distinct scenarios and generating reports with all existing inconsistencies.

4. Conclusions and Future Work

This paper presented a feature model notation to context-aware product lines called UbiFEX. Some extensions are proposed to an existing feature modeling notation, named Odyssey-FEX, to support representation of context information and dynamic product derivation. In this way, UbiFEX proposes a solution for the weak points found in the related work regarding context representation, providing a better understanding and representation of the relevant context entities and information in a context-aware domain, using the same type of model to represents other feature types.

That explicit representation of context is essential to modeling context-aware applications. Contexts influence directly the behavior of these applications. Therefore, it is important to identify them since the first product line development activities.

UbiFEX also promotes a first step to a dynamic configuration of products based on context rules. Moreover, it allows an early verification of the product execution through context simulation.

Context information is a key element to produce self-adaptive applications in ubiquitous computing [2]. For this reason, our first concern was the way of dealing with this type of information in the software product line development. However, there are still many challenges to build context-aware product lines, mainly due to the dynamic adaptation aspect of this class of applications.

As future work, we are planning to evaluate the proposed feature notation in a real scenario, modeling a family of existing context-aware applications for mobile devices. Then, we can analyze if relevant contexts and adaptive rules can be represented in an effective way. Since Odyssey environment supports other feature model notations, we intend to estimate the effort to extend the proposed approach to these notations.

Moreover, we also intend to use feature models to dynamic product derivation, developing an automated product variant selection based on features and rules proposed in this paper.

5. Acknowledgments

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

6. References

[1] Baldauf, M. and Dudstar, S. 2004. A survey on context-aware systems. Tech. Report TUV-1841-2004-24, Tech. Univ. of Vienna.

[2] Bardram, J. E. 2005. The Java Context Awareness Framework – A Service Infrastructure and Programming Framework for Context-Aware Applications, Third International Conference, Pervasive2005, Munich, Germany, 98-115.

[3] Baus, J., Cheverst, K., and Kray, C. 2005. A survey of map-based mobile guides. Map-based mobile services - Theories, Methods, and Implementations, 197-216.

[4] Czarnecki, K., Helsen, S., and Eisenecker, U. 2004. Staged Configuration Using Feature Models. In Proc. of Software Product Line Conference (SPLC 2004), Lecture Notes in Computer Science, Springer-Verlag, 266-283.

[5] Dey, A. 2001. Understanding and Using Context. *Personal Ubiquitous Comput.* 5, 1 (Jan. 2001), 4-7.

[6] Garlan, D., Siewiorek, D., Smailagic A., and Steenkiste, P. 2002. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing* 1, 2 (April 2002), 22-31.

[7] Gomaa, H. and Hussein, M. 2003. Dynamic Software Reconfiguration in Software Product Families. In Proc. of the 5th Int. Workshop on Product Family Engineering (PFE), Lecture Notes in Computer Science, Springer-Verlag, 435-444.

[8] Gomaa, H. 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison-Wesley Professional.

[9] Hallsteinsen, S., Stav, E., Solberg, A., and Floch, J. 2006. Using Product Line Techniques to Build Adaptive Systems. In *Proceedings of the 10th international on Software Product Line Conference* (August 21 - 24, 2006). International Conference on Software Product Line. IEEE Computer Society, Washington, DC, 141-150.

[10] Lee, J. and Kang, K. C. 2006. A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In *Proceedings of the 10th international on Software Product Line Conference* (August 21 - 24, 2006). International Conference on Software Product Line. IEEE Computer Society, Washington, DC, 131-140.

[11] Lee, J. and Muthig, D. 2006. Feature-oriented variability management in product line engineering. *Commun. ACM* 49, 12 (Dec. 2006), 55-59.

[12] McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. 2004. Composing Adaptive Software. *Computer* 37, 7 (Jul. 2004), 56-64.

[13] Northrop, L. M. 2002. SEI's Software Product Line Tenets. *IEEE Softw.* 19, 4 (Jul. 2002), 32-40.

[14] Oliveira, R. 2006. Formalization and Consistency Checking in Variabilities Modeling. Master Thesis. Federal University of Rio de Janeiro, Rio de Janeiro, Brazil (in Portuguese).

[15] Software Reuse Team. 2008. Odyssey Project. http://reuse.cos.ufrj.br/odyssey.

[16] Sugumaran, V., Park, S., and Kang, K. C. 2006. Introduction – Software product line engineering. *Commun. ACM* 49, 12 (Dec. 2006), 28-32.

[17] van der Hoek, A. 2004. Design-time product line architectures for any-time variability. *Sci. Comput. Program.* 53, 3 (Dec. 2004), 285-304.

[18] Weiser, M. 1999. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3 (Jul. 1999), 3-11.