# Enhancing Formal Methods with Feature Models in MDD

**Felice Laura**
*Universidad Nacional del Centro de la Provinicia de Buenos Aires, Argentina*

**Ridao Marcela**
*Universidad Nacional del Centro de la Provinicia de Buenos Aires, Argentina*

**Mauco María Virginia**
*Universidad Nacional del Centro de la Provinicia de Buenos Aires, Argentina*

**Leonardi María Carmen**
*Universidad Nacional del Centro de la Provinicia de Buenos Aires, Argentina*

## INTRODUCTION

As formal methods offer a wide spectrum of possible paths towards designing high-quality software, they are receiving increasing attention in the academia and the industry, especially where safety or security is important (Streitferdt; Riebisch & Philippow, 2003). By using formal methods early in the software development process, ambiguities, incompleteness, inconsistencies, errors, or misunderstandings can be detected, avoiding their discovery during costly testing and debugging phases.

A well-known formal method is the RAISE Method (George, Haxthausen, Hughes, Milne, Prehn, & Pedersen, 1995), which has been used on real developments (Dang Van, George, Janowski, & Moore, 2002). This method includes a large number of techniques and strategies for doing formal development and proofs, as well as a formal specification language, the RAISE Specification Language (RSL) (George, Haff, Havelund, Haxthausen, Milne, Nielsen, Prehn, & Wagner, 1992), and a set of tools to help writing, checking, printing, storing, transforming, and reasoning about specifications (George, 2001). However, formal specifications are unfamiliar to stakeholders, whose active participation is crucial in the first stages of software development process to understand and communicate the problem. This also holds in Domain Analysis (DA) (Kang, Kim, Lee, & Kim, 1998), because its first stage is to capture the knowledge of a particular domain, making necessary to have a model that is comprehensible by software engineers and domain experts.

A way to contribute to bridge this gap is to work in the integration of a DA phase into the RAISE Method, in order to specify a family of systems to produce qualitative and reliable applications in a domain, promoting early reuse and reducing development costs.

The use of Feature Models (FM) (Eisenecker & Czarnecki, 2000) to represent the DA facilitates the customization of software requirements. In DA, features and relationships between features (called Domain Feature Model) are used to organize the requirements of a set of similar applications in a software domain.

In this article, an informal strategy which starts by defining this feature model following one of the several proposals that facilitate the construction of FM: Feature-Oriented Reuse Method (FORM) (Kang, Kim, Lee, & Kim, 1998), is presented. Then, this model is transformed, by means of a set of manual heuristics, into a RSL specification that can be later developed into a more concrete one to automatically obtain a prototype to validate the specification by using the RAISE Tools. The use of FM is motivated by the fact that stakeholders often speak about product characteristics in terms of "features the product has and/or delivers," using them to communicate their ideas, needs, and problems.

In order to fit the main proposal of enhancement of formal developments with the RAISE Method into Model Driven Development (MDD) paradigm (Mellor, Clark, & Futagami, 2003), an ATL (Atlas Transformation Language) (ATL, 2012) transforma-

DOI: 10.4018/978-1-4666-5888-2.ch700

tion which allows the automatic derivation of a first abstract RSL specification of a domain starting from a Feature Model has been developed. The ATL rules define how features and relationships between them (the source model) are matched and navigated to produce the RSL specification (the target model) (Felice, Ridao, Mauco, & Leonardi, 2011). The rules follow closely the principles proposed in the RAISE Method, so this first and still incomplete specification may be later developed into a more concrete one following the RAISE Method steps. With a concrete specification, the RAISE tools can be used to automatically obtain a quick prototype and get a feeling of what the specification really does.

An improved version of the ATL transformation is presented in this article. With this transformation, a MDD software development process which obtains an initial RSL specification that will be the basis for a PIM (Platform Independent Model) is enhanced.

## BACKGROUND

### The Feature Model

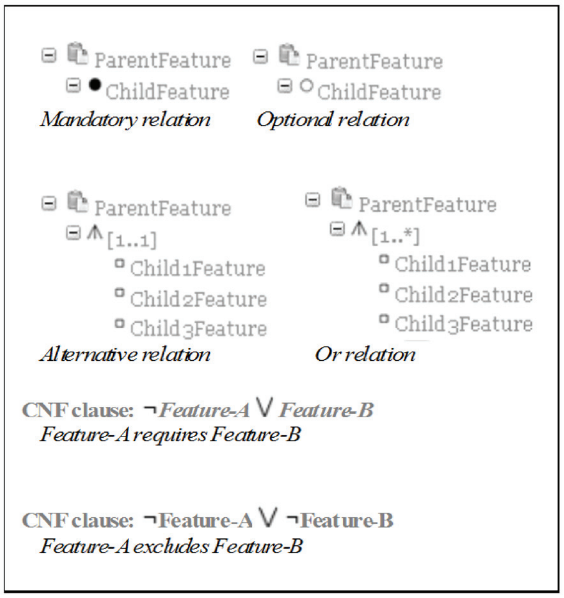
Domain modeling is the result of the analysis of commonalities and variabilities of systems within a domain. It is a high level description of the application family, providing a framework for describing the essential characteristics. Examples of more relevant methods include Feature-Oriented Domain Analysis (FODA) (Kang, Cohen, Hess, Novak, & Peterson, 1990), Organization Domain Modeling (ODM) (Simos, 1995), and FORM. They support the notion of feature-oriented. This is a concept based on the emphasis this method places on finding the features or functionalities usually expected in applications for a given domain. FM is a modeling notation used to represent the variability in a system family and describes all valid configurations.

The model captures commonality as an AND/OR graph. Then, this model is used to define parameterized reference architectures and appropriate reusable components which are instantiated during actual application development. FM are able to describe the aspects, commonalities or characteristics of a system and include variability modeling for system families. Commonalities can be modeled by common features (mandatory features whose ancestors are also mandatory), and variabilities can be modeled by variant features, such as optional, alternative, and or-features.

A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems (Eisenecker & Czarnecki, 2000). A feature is detailed in any number of other features (subfeatures). Mandatory features describe detailed aspects that the parent feature must support, while optional features may be selected when creating a concrete system from a feature model. Alternative features have a multiplicity similar to the UML multiplicity, defining how many of the features must or may be selected. In addition to the hierarchical relationship, a constraint relation defines if a feature requires or excludes with any other features. Figure 1 summarizes the notation used in (Montero, Pena, & Ruiz-Cortes, 2008) for Feature Diagram (FD).

The feature requested by a stakeholder is called a concept feature, it is the root node of the FD and all features are represented as child nodes. The hierarchical relationships mandatory, optional, and alternative are represented by different edges between the nodes. A simple line with a filled circle represents a mandatory relationship, while the optional is represented with a line ending with an empty circle. Arcs spanning two or more edges of the feature nodes depict a set of alternative features. The arc is annotated with the multiplicity

*Figure 1. FD notation*


ParentFeature
ChildFeature
Mandatory relation
ParentFeature
ChildFeature
Optional relation
ParentFeature
[1..1]
Child1Feature
Child2Feature
Child3Feature
Alternative relation
ParentFeature
[1..*]
Child1Feature
Child2Feature
Child3Feature
Or relation
CNF clause: ¬Feature-A ∨ Feature-B
Feature-A requires Feature-B
CNF clause: ¬Feature-A ∨ ¬Feature-B
Feature-A excludes Feature-B