Integrating Formal Methods with Reuse Techniques

Laura Felice, INTIA, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina; E-mail: lfelice@exa.unicen.edu.ar
Carmen Leonardi, INTIA, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina; E-mail: cleonard@exa.unicen.edu.ar
Ma. Virginia Mauco, INTIA, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina; E-mail: vmauco@exa.unicen.edu.ar

ABSTRACT

It is widely accepted by software community that formal methods increase software quality and reliability, and even though their industrial use is still limited it has steadily been growing. A well-known formal method is the RAISE Method [4,5] based on the idea that software development is a stepwise. Originally designed to be applied at different levels of abstraction as well as stages of development, RAISE is successfully applied to different domains obtaining high precise specifications of components. However, there is no explicit reference to the specification reusability in the process. Feature models have received much attention in the software engineering community who see the Domain Analysis [7] as a prerequisite in a successful reuse, for example FODA [8], FORM [9] and Featured Reuse-driven Software Engineering Business (FeatuRSEB) [6]. This paper presents a brief overview of feature modeling and the integration into RAISE. Some key relations between features are formalized using RAISE specification language [3]. Such integration allows to take advantage of formal methods in the context of software reuse.

INTRODUCTION

Formal methods have come into use for the construction of real systems, as they help to increase software quality and reliability, and even though their industrial use is still limited it has steadily been growing. A well-known formal method is the RAISE Method, which has been used on several real developments. 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.

In particular, there are two main activities in the RAISE method: writing an initial specification, and developing it towards something that can be implemented in a programming language. Writing the initial specification is the most critical task in software development. If it is wrong, i.e. if it fails to meet the requirements, the following work will be largely wasted. It is well known that mistakes made in the life-cycle are considerably more expensive to fix than those made later.

Our goal is to work with reuse in the confines of the domain engineering, where there is no reference in RAISE process. Therefore, the introduction of a Domain Analysis method into RAISE is a crucial task considering the possibility of reusing the specifications in the future.

Examples of more relevant Domain Analysis methods include FODA, FORM and Featured Reuse-driven Software Engineering Business (FeatuRSEB). 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.

In a reuse strategy, domain analysis must be maintained over many systems, and the repository should contain domain models that form the basis of subsequent development activities. Domain analysis is essential to formalize reuse. However, it is missing from most software development methods. Reuse engineering extends information engineering by adding this new stage, to provide a place in the life cycle where the most valuable reusable components for the domains of the enterprise can be identified and a library containing these components can be created. At this stage of the software development, working with formal methods (or formal specification languages, specifically) implies to provide a means of unambiguously stating the requirements of a system, or of a system component. In this way, formally specified system components that meet the requirements of components of the new system can be easily identified. Thus, components that have been formally specified and sufficiently well documented can be identified, reused and combined to form components of the new system.

Nevertheless, the main problem is that we may not understand the requirements. Specially, when the requirements are written in a natural language the result is likely to be ambiguous. The aim of the initial specification is to capture the requirements in a precise way applying a reusability model.

Based on this paradigm, our work consists in the incorporation of the feature model into a RAISE formal method, filling the gap between requirements and the RSL (RAISE specification language) specifications. In this work we suggest introduce the phase -reusable domain analysis- using a feature model and expressing the relationships among them in RSL language. Thus, we can combine domain analysis notions with a formal language in early phases of development process. Particularly, we use the feature model proposed by FORM method, briefly described in section 2.

The remainder of the paper is organized as follows. In section 3, we briefly introduce the Raise method. Section 4 presents the integration of Domain analysis into RAISE. In section 5 we give a formalization of the relation in a feature model in RSL language. Section 6 concludes the paper and describes future work.

2. THE FORM METHOD

FORM product line engineering consists of two major processes: asset development and product development. Asset development consists of analyzing a product line (domain analysis, feature modeling) and developing architectures and reusable components based on analysis results. Product development includes analyzing requirements, selecting features, selecting and adopting architecture, and adapting components and generating code for the product.

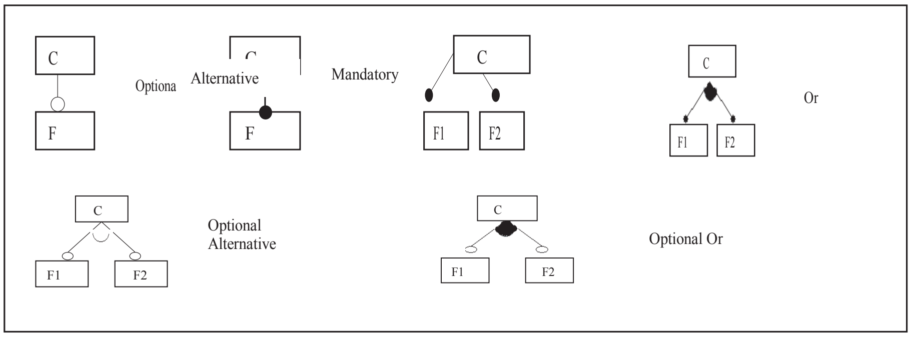
The FORM (Feature-Oriented Reuse Method) method comes as a concretization of the FODA (Feature-Oriented Domain Analysis) processes and has been recently extended by Feature Oriented Product Line Software Engineering (FOPLE) [10,11]. It provides guidelines for the creation of the feature model, design and implementation phases. FORM performs an analysis of domain´s features and attempts to provide a mapping between features and architectural components. This method follows all principles of modern software, being flexible, extendible and maintainable.

The use of features is motivated by the fact that users and developers often speak of product characteristics in terms of "features the product has and/or delivers". That is, services provided, and techniques used in applications are abstracted as features, and they are used by domain experts to communicate their ideas, needs, and problems.

To create coherent models, feature analysis involves tasks for identifying, classifying, and organizing product features as models. Feature models are a well accepted means for expressing requirements in a domain on an abstract level, and it resides between the requirements model and the system design model.

As potential features are identified, they are classified according to the types of information they represent. For example, users are concerned with functions provided by the systems (i.e. service features); analysts and designers are concerned about domain technologies, and developers are concerned about implementation techniques.

Figure 1. Types of features



FORM separates features into four different feature categories:

- *Capability features* are distinct services, operations, or non-functional aspects. Features of this category are end-user visible and are selected by the customer to specify the desired system.
- *Operating environment features* address the hardware and software components used by the family. All the components of a system with their interfaces and protocols are part of this category.
- *Domain technology features* are domain specific technologies and problem solutions, used by domain experts.
- *Implementation technique features* are general problem solutions, which may be used in different domains.

A feature model should cover all four categories of features for a domain. To make it possible, FORM uses the following constructs:

A *feature diagram*: a graphical AND/OR hierarchy of features, capturing the logical relationships (composition / generalization) among features. Three types of relationships are represented in this diagram: "composed-of", "generalization/specialization", and "implemented-by". Features themselves may be "mandatory" (unless specified otherwise), "optional" (denoted with a circle), or "alternative" (denoted with an arc) [Figure 1]. A mandatory feature is neccesary for general users, and an optional feature is neccesary for partial users. Czarnecki [1] introduced the notion of sets of features.

Using the optional, mandatory and alternative criteria for features, it is possible to define subsets with constraints for minimum and maximum number of features to be taken out of this set.

Composition rules that supplement the feature diagram with mutual dependency and mutual exclusion relationships.

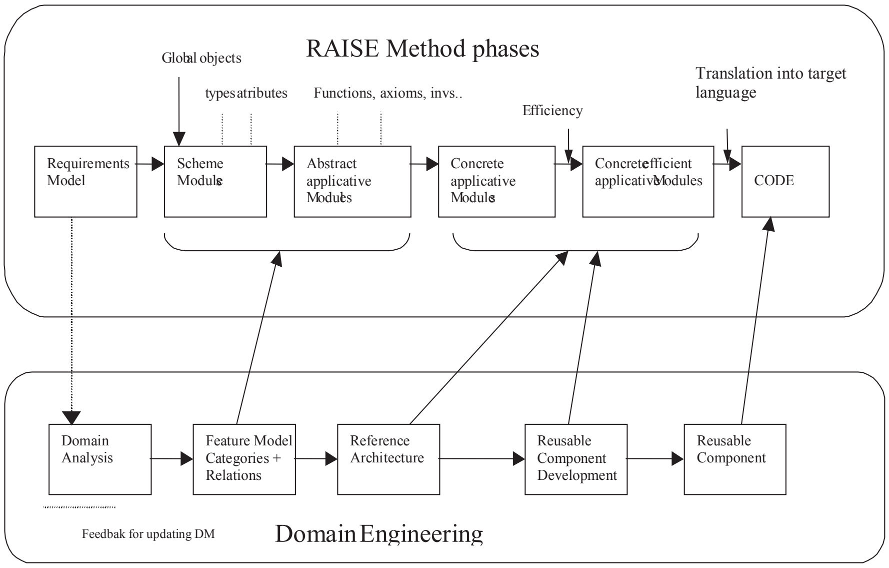
Depending on the domain, it is possible that AND/OR diagram tends to become complex. The AND/OR diagram was used to show the relationships among selected features.

Features are considered following the four level feature hierarchy, since the hierarchy reflects step-wise refinement in the reference architecture. These concepts are strongly connected with the style of development in RAISE, where the separate and step-wise are the basis to build a solid specification of an infrastructure.

3. RAISE OVERVIEW

RAISE method provides guidelines to hierarchically structure a specification, aiming at encouraging separate development and step-wise development. A development in RAISE begins with an abstract specification and gradually evolves into concrete

Figure 2. RAISE phases and DE


RAISE Method phases
Glob al objects
types atributes
Functions, axioms, invs..
Efficiency
Translation into target language
Requirements Model
Scheme Modules
Abstract applicative Modules
Concrete applicative Modules
Concrete efficient applicative Modules
CODE
Domain Analysis
Feature Model Categories + Relations
Reference Architecture
Reusable Component Development
Reusable Component
Feedbak for updating DM
Domain Engineering

implementations. RAISE proposes to structure modules hierarchically in order to get a particular component over by reference only to it and its suppliers, to limit the effects of changes of a module to it and its clients, and to limit the properties of a module to it and its suppliers. It is an object-oriented method and covers a large portion of systems development phases.

Moreover, the RAISE method permits the abstract specification of sequential as well as concurrent systems, modular operations for decomposing large systems into subsystems and composing subsystems into a more complex system.

4. INTEGRATING DOMAIN ANALYSIS INTO RAISE

In this section we present the integration of domain analysis into RAISE method. The underlying idea is to specify and design a family of systems to produce qualitative application in a domain, as we can see in the lower part of figure 2, promoting early reuse and reducing development costs.

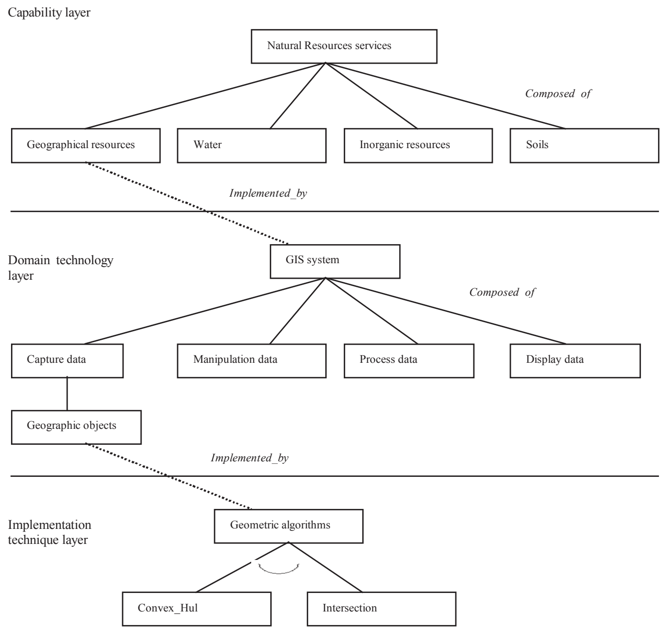
Domain engineering gives a set of guidelines that can be used to derive domain products from the feature model. The objective of domain engineering is to establish a mapping between the decision space (Feature model) and the artifact space (Reference Architecture). Each feature with the corresponding relations in the decision space somehow constrains the selection of the final reference model considering differences in types of features following the four level hierarchy. Feature model gives a set of guidelines that can be used to derive domain products from it and it is considered as an intermediate step between analysis and design models.

4.1 The Feature Model and RSL Modules

The objective of a feature model is to capture commonalities and variabilities of a family. There is a trace from the requirements to the feature model, and there are relations within the feature model. Categories help to elaborate features, but feature relations do not have a rigorous definition and we need the precise usage of relations for modeling.

Given the example of [12] in Figure 3 (Feature Model of Agriculture system domain), next section deals with some relations among features, and we show them described in a way we can automatically check the feature model for consistency. "Agriculture system" is an information system whose objective is a model which will help deciders to identify problems with the management and the access to resources for several purposes. Following, we discuss how the feature model serves as a guideline to identify RSL reusable modules.

Figure 3. Agriculture System partial feature model



In the partial example illustrated in Figure 3, Natural Resources service can be mapped respectively, into Natural Resources RSL module, performing a set of operations with its specific role. Once a feature is mapped into a module, the sub-features of the feature such as Geographical Resources, Water, Inorganic resources, Soils can be modules that are part of Natural Resources following the same type of relationships in the feature model (e.g., generalization, aggregation).

Moreover, operations can be mapped as internal functions to provide services, and they are a collection of types and values without type of interest. On the other hand, non-functional features include end-user visible application characteristics that cannot be identified in terms of services or operations, like quality attribute, cost, etc. So, they can not be mapped into any RSL constructions.

With respect to the model operating environment features, the RSL specifications are independent from the operating environment. These features can be mapped into the subsidiary RSL modules, which are less important from the point of view of development.

Model domain technology features such as GIS systems will be considered by the RSL system modules and they will be expected to be finally implemented as software modules. In object-oriented terms they will form the objects of the software system.

Modules derived from implementation technique features are generally used to implement or to derive concrete applicative specifications derived from capability or domain technology features. Geometric Algorithms would be part of a module called "Algorithms". This module will be defined as a generic module, i.e. a module we expect to instantiate more than once with different parameter, being the sub-features (Convex_Hull or Intersection) the possible instantiations. Each RSL module derived would be later refined and completed with the definition of functions. Once the RSL modules are derived from the feature model, they need to be organized into a model in order to represent how they are related to each other and what the contexts for their use are.

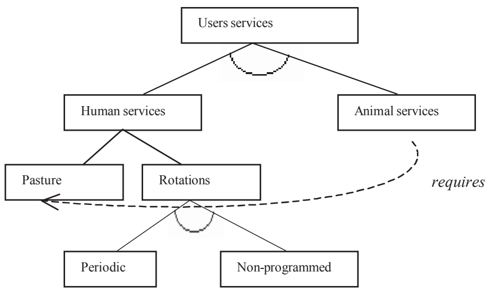
This simple heuristic is very useful in providing a good first reference architecture that will be the basis to specify the Concrete applicative RSL modules.

4.2 Dependencies in a Feature Model

To complete the feature model, we give an overview of dependencies used not only in FORM but also in FOPLE and FeatureRSEB. "Composed of" relation is used when a parent feature consists of a set of child features. "Generalization/specialization" relation connects very general features with concrete ones. "Implemented by" relation represents a connection between user-visible features and their implementation strategy, used in the specific domain. In Figure 3 we have taken a subset out of all the features of the Agriculture system. We show features of Natural Resources services: the features "Geographical resources", "Water", "Inorganic resources" and "Soils" are so-called mandatory features, and are part of the instance by definition.

The Capability layer is used to represent the details of the family required to further develop the system design, while Domain technology layer are specific technologies and problem solutions linked with the sub-feature Geographical resources. The Implementation technique layer are the 'solutions' used by domain experts (Example: Geographic objects are "implemented by" Geometric algorithms). "Requires" dependency (uni-directional) is used to describe if one feature needs another (e.g. Animal services "requires" Pasture, figure 4). "Exclude" depen-

Figure 4. "Requires" relation



dency (bi-directional) is used when one feature conflicts with another. "Hints" dependency is used to express that the choice of another feature increases the system usage. "Mathematical" dependency describes the relative impact from one feature to another.

5. FORMALIZING THE RELATIONS OF THE FEATURE MODEL IN RSL

RSL is one of the most versatile and comprehensive languages for formal specification, design and development of software. A significant advantage of using RSL is that it combines both algebraic and coalgebraic specification techniques in one specification language.

Different features in a feature model are related by different kinds of relations. Generalization, requires, excludes and implemented_by are considered binary relationships. Each relation in a feature model must be well formed. Next, the structure for each type of relation is defined as follows:

```
type

  Rel={|r:Rel1 • well_formed (r ) |},
  Rel1=
     Generalization | Implemented_by | Composed_of | Requires |
     Excludes ….
type
  Generalization::
      subfeature: Feature
      superfeature: Feature,
 Composed_of::
         has-part: Feature
         Part-of: Feature-set,
 Implemented_by::
      source: Feature
      target: Feature,
 Requires::
      source: Feature
      target: Feature,
 Excludes::
      source: Feature
      target: Feature,
   ……………..
```

Below, we give the boolean function "well_formed" used to define well-formed relationships. Each relation has different properties. For example, the "generalization" relation must satisfy the following: a subfeature can not be root; the superfeature can not be leaf and the subfeature can not redefine the attributes of their superclasses. The "requires" relation describes that the binding of one variant implies the need of another variant (required variant). "Excludes" defines a feature relation that the selection of one feature excludes the selection of the other (see below).

Also, as relations are described in RSL language, all the concepts involved in the feature model can be specified in RSL. Multiplicity for features and parameters for features [1] in a graphical notation way may result ambiguous. It is neccesary to give clear semantics. The need to consider the complete set of relations has been identified by the feature modelling community, and approaches to formalizing them are defined in several formal languages [2, 14].

6. CONCLUSIONS

The use of formal methods in system development can help to overcome inconsistencies, and should aid the promotion of software reuse in early stages of software development. In this paper we present a first approach to integrate a feature model into RAISE methods to be used in the context of software reuse. The feature model enables to work with the identification of commonalties and variabilities among related applications creating a feature model of a specific domain. More precisely, a feature model has been developed for the "Agriculture system" being the basis to the specifications of the RAISE reusable component. In this work we identified basic features relations, i.e. Generalization, Requires and Exclude, among the features in a domain context. Our contribution is not only to define features, categories and relations in RSL; but also to give an approach to a rigorous reasoning of feature models. We are currently extending the work by taking feature interaction [13]. Feature interaction occurs when one feature modifies the operations of another. Also, we are working towards a reference architecture in the RAISE method framework.

REFERENCES

[1] Czarnecki, K. " Generative Programming". Dissertation TU-Ilmenau, 1998.

[2] Detlef Streifert, Matthias Riebisch, I. Philippow; Formal Details of Relations in Feature Models; In:Proceedings 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems. USA April 2003.

[3] George, C., Haff, P., Havelund, K., Haxthausen, A., Milne, R., Nielsen, C., Prehn, S. and Ritter, K. The RAISE Specification Language, Prentice Hall, UK, 2002.

[4] George, C. RAISE Tools User Guide. (Technical report No.227) Retrieved from http:// www.iist.unu.edu, 2001.

[5] George, C., Haxthausen, A., Hughes, S., Milne, R., Prehn, S. and Pedersen, J., The RAISE Development Method, Prentice Hall, UK, 1995.

[6] Griss, D.; Allen, R. And d´Alessandro, M.: "Integrating Feature Modelling with the RSEB". In Proceedings of the 5th International Conference of Software Reuse (ICSR-5), 1998.

[7] Hess, J; Novack, W; Carrol, P; Cohen, S; Holibaugh, R; Kang, K; and Peterson, A. "A domain analysis bibliography". SEI-90-SR-3 Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[8] Kang, K., et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

```
value
  well_formedGen: Feature x Feature-> Bool
  well_formed Gen (subfeature, superfeature) ≡
   ~is _root(subfeature) ^
   ~is_leaf(superfeature)^
   (∀at1,at2: Attribute •
       (at1∈ attributes(subfeature)^ at2 ∈
         attributes(superfeature))⇒
         name(at1)≠ name(at2)
```

"*Generalization*" RSL construct

```
value
  well_formedReq: Feature x Feature-> Bool
  well_formedReq (requester, supplier) ≡
   is_selected(requester) ⇒ is_selected (supplier)
```

"*Requires*" RSL construct

```
value
  well_formedExc: Feature x Feature-> Bool
  well_formedExc (source, target) ≡
    is_selected(target) ⇒ ~is_selected (source) ^
    is_selected(source) ⇒ ~is_selected (target)
```

"*Excludes*" RSL construct

[9] Kang, K; Kim, S, Lee, J and Kim, K "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures". Annals of Software Engineering, 5, pp: 143-268.

[10] Kang, K; Kim, S, Lee, J and Kim, K "Feature-Oriented Product Line Software Engineering: Principles and Guidelines. In: Domain Oriented Systems Development: Practices and Perspectives". Taylor & Francis, 2003, pp: 19-36.

[11] Lee, K; Kang, K; Chae, W and Choi B. "Feature-based approach to object-oriented engineering of applications for reuse". In Software -Practice and Experience 2000:3. J.Wiley &Sons Ltd, pp: 1025-1046.

[12] Riesco, D; Felice, L; Debnath, N and Montejano, G (2005). Using a feature model for RAISE specification reusability. In Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration, IRI – 2005. Las Vegas, NV USA. (pp: 306-311).

[13] Zave, P. Feature interactions and formal specifications in telecommunications. IEEE Computer. 1993.

[14] Zhang, Hongyu; Jing Sun and Hai Wang, Formalizing and Analyzing Feature Models in Alloy, RMIT CS Technical Report TR-04-2, March 2004.