Category: Software & Systems Design

Integrating Domain Analysis into Formal Specifications

Laura Felice
Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina

Daniel Riesco
Universidad Nacional de San Luis, Argentina

INTRODUCTION

Reusability is widely suggested as a key to improve software development productivity. It has been further argued that reuse at domain level can significantly increase reuse at later stages of development. The development of a particular system that exploits previously accumulated domain knowledge can be the source for new insights about the domain that adds to or refines. The classification of similar problems grouped by domains is the key to find reusable solutions that belong to similar problems. This work deals with the integration of a reusability model into a formal method in order to enhance the benefits of reusability at the domain and design levels. Working with formal methods, software reuse problems such as the detection of inconsistencies in component integration can be revealed in early stages of development.

The rigorous approach to industrial software engineering (RAISE) (George et al., 1995) formal method was originally designed to be applied at different levels of abstraction as well as stages of software development. It includes several definition styles of specifications such as model-based or property-based, applicative or imperative, sequential or concurrent. However, it is not easy to identify reusable specifications, due to the fact that objects identified from this method tend not to be reusable in other applications as they are defined without a proper domain perspective. Even though RAISE supports object-orientation, the major approaches to object identification (keyword analysis, structured analysis, scenario-based analysis) can not provide support for reusability and adaptability of applications without analyzing the commonality and variability among a family of applications in a domain (Lee, Kang, Chae, & Choi, 2000).

To address this problem, there have been methods for reuse whose development has been based on the notion of "domain orientation" (Barstow, 1985; Prieto-Diaz, 1987), which emphasizes a group of closely related applications in a domain rather than a single application. The exploitation of commonality across related software systems is a fundamental technical requirement to achieve successful software reuse. Software product lines (PL) (Kang, Kim, Lee, & Kim, 2003) present a solid approach in large scale reuse. Due to the PL's inherit complexity, many PL methods use the notion of "features" to support domain modeling (e.g., FODA) (Kang et al., 1990), FORM (Kang, Kim, Lee, & Kim, 1998), FeatuRSEB (Griss, Allen, & d´Alessandro, 1998). These methods identify common abstractions across the applications of a domain in order to engineer reusable domain components. Feature modeling mainly focuses on identifying commonalities and variabilities among products of a PL, and organizing them in terms of structural relationships (e.g., aggregation and generalization) and configuration dependencies (e.g., required and excluded).

This work is related to the introduction of a feature-oriented reuse method to the RAISE components specifications and to give a solution to "bridge" the gap between the domain analysis and the specifications in RAISE (Riesco, Felice, Debnath, & Montejano, 2005). In particular, FORM method (feature-oriented reuse method) is introduced in order to gradually improve the reuse of RSL (George et al., 2002) components specifications, incorporating the effectiveness of software reusability as an integral part of the software specification process, considering the following:

- To create a plan of reuse as part of the project plan
- To add steps to look for and evaluate reusable component candidates to use in the project
- To add guidelines to create a future component for reuse
- To evaluate the benefits and costs associated with the practice of reuse in the project
- To finally identify created components of the project which have the potential for reuse in the future

BACKGROUND

PL engineering is an emerging software engineering paradigm, which guides organizations toward the development of products from core assets rather than the development of products one by one from scratch. Two major activities of PL are core asset development (i.e., product line engineering) and product development (i.e., product engineering) using the core assets. The paradigm of developing core assets for application development has been called domain engineer-

ing (DE), in which emphasis is given to the identification and development of reusable assets from an application "domain" perspective (Lee, Kwanwoo, Kang, & Lee, 2002). The purpose of domain engineering is to develop domain artifacts that may be used and reused in development applications for a given domain. DE consists of activities for gathering and representing information on systems that share a common set of capabilities and data. In usual approaches to software reuse, the product of such domain engineering might include only the reusable components and their parametric representations applicable to that domain.

DE, the key to systematic software reuse, has two phases: domain analysis (DA) and domain implementation.

DA is the process of discovering and recording the commonalities and variabilities in a set of software systems, while domain implementation is the use of the information from DA to create reusable assets and new systems within a domain (Frakes, 1994).

The view of DA follows the line of thought pioneered in Neighbors' DRACO system (Neighbors, 1980), where the importance of DA in reusability was pointed out. This is summarized as "to identify objects and operations for a class of similar systems."

DA is a process that affects the maintainability, usability, and reusability of a system and includes:

- Domain definition
- Domain analysis
- The development of a domain architecture
- The construction of components (specifications, design, documentation)

In the field of software engineering, DA is seen as a prerequisite for successful reuse not only by the researchers in the reuse community but also by the methodologists who have introduced component-based development methods (D'Souza & Wills, 1999; Gamma, Helm, Johnsson, & Wlissides, 1995; Jacobson, Booch, & Rumbaugh, 1999). A review of some domain analysis methods is presented in Kang (1990) and an extensive domain analysis bibliography can be found in Hess et. al. (1990).

The result of the analysis, usually known as the domain model, is retrieved for reuse in future developments or similar systems and also, for maintainability of legacy systems.

In a reuse strategy, DA must be maintained over many systems, and the repository should contain domain models that form the basis of subsequent development activities.

DA is essential to formalize reuse. However, it is missing from most software development methods. Reuse engineering extends information engineering by adding the new stage "domain analysis" 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 the requirements may not be clear. 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.

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 (George et al., 2002). 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. So, the introduction of a DA method is a crucial task considering the possibility of reusing the specifications in the future.

Examples of more relevant DA methods include FODA, FORM, and 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. The FORM engineering processes are illustrated in Figure 1.

THE FORM METHOD

FORM product line engineering consists of two major processes: asset development and product development, as it can be seen in Figure 1. 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 an architecture, and adapting components and generating code for the product.

This method has been applied to several industry application domains including elevator control systems, electronic bulletin board systems yard automation systems and PBX, to create product line software engineering environments and software packages (Kang et al., 2003).