Reuse of Formal Specifications

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

Daniel Riesco
Universidad Nacional de San Luis, Argentina

INTRODUCTION

During the Rigorous Approach to Industrial Software Engineering (RAISE) specification development process, a variety of components and infrastructures are built. All of these components are not independent, but related to one another, especially when we specify different systems into the same infrastructure. The RAISE method (Bjorner, 2000) is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions. Thus, the reuse process is crucial in all of the stages of the development, but there is no explicit reference to the specification reusability in this development process.

Software components are typically very rich in information, making the task of characterizing them and capturing their relevant properties difficult. However, this is not the only reason that makes software reuse difficult. Krueger (1992) provides a brief general survey and very clear view of different approaches for software reuse.

Information retrieval methods based on analyses of natural-language documentation have been proposed to construct software libraries (Helm & Maarek, 1991; Maarek, Berry & Kaiser, 1991). Software components represented by natural language can make the retrieval process a task with ambiguity, incompleteness and inconsistency. Using a rigorous method to the retrieval of a component can minimize all of these problems.

Based on these observations, we introduce a Reusable Component (RC) model for the definition of the reusable component structure into RAISE. Related to this method, it is important to emphasize the work of Beltaifa and Moore (2001). They propose an infrastructure to support reuse improving the efficiency of reusing software components.

RC model integrates RAISE Specification Language (RSL) specifications (George et al., 1992) and object-oriented code. RC model describes object-oriented classes at different levels of abstraction:

- Specialization: hierarchies of RSL implicit specifications related by formal specialization relationship.
- Realization: hierarchies of RSL complete algebraic specifications related by realization relationship.
- Code: hierarchies of imperative RSL schemes related by implementation relationship and linked to object-oriented code.

Also, a rigorous process for reusability of RC components is defined. Its manipulation, by means of specification building operators (Rename, Extend, Combine and Hide), is the basis for the reusability.

Our approach allows that the properties of components formally specified can be characterized by giving a functional (RSL specification) description. Therefore, they may be useful to someone searching for a particular component.

Different possible classes of existing RC components may be retrieved using a formal reasoning technique: an exact match to the query specification, a component more general than the query, or a component more specific than the query.

BACKGROUND

Different approaches to specify reusable components functionalities have been proposed. The way in which the components can be used with others can play a critical role in the reuse implementation.

Related with the RAISE method, we emphasize the work of Beltaifa. They propose an infrastructure to support reuse which improve both the ease and efficiency of reusing software components. The main difference with our work is the integrated process defined for all stages of the development method.

As a typical related work, we can mention Hennicker and Wirsing (1992) who present a model for reusable component definition. A reusable component is defined as an unordered tree of specifications where any two consecutive nodes are related by the implementation relation and the leaves are different implementations of the root. The work of Chen and Cheng (1997) is another approach that provides a formalism to register compo-

nents properties to reuse them based on the architecture and integration of the system. They are related to LOTOS tools to facilitate the retrieval of the reusable component.

On the other hand, the work of Zaremski and Wing (1997) is related to the specification matching. It is very important to emphasize this proposal has been referenced by a lot of authors.

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, 2002). Writing the initial specification is the most critical task in software development. If it is wrong, that is, 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.

What kinds of errors are made at the beginning? The main problem is that we may not understand the requirements. The requirements are written in a natural language, and, as a result, likely to be ambiguous. The aim of the initial specification is to capture the requirements in a formal and precise way.

MAIN THRUST OF RAISE AND REUSE

The aim of the project RAISE was to develop a language, techniques and tools that would enable industrial usage of "formal methods" in the construction of software systems. The results of this project include the RSL language, which allows us to write formal specifications; a method to carry out developments based on such specifications, and a set of tools to assist in edition, checking, transforming and reasoning about specifications.

RSL is a "wide spectrum" language that can be applied at different levels of abstraction as well as stages of development. It includes several definition styles such as model-based or property-based, applicative or imperative, sequential or concurrent.

A development in RAISE begins with an abstract specification and gradually evolves to concrete implementations. The first specification is usually an abstract applicative one, for example, functional or algebraic. A first algebraic specification should have:

- A hierarchy of modules whose root is the system module;
- A module containing types and attributes for the non-dynamic identified entities; and
- The signatures of the necessary functions associated with types. These functions should be categorized as generators (if the associated type or a type dependent on it appears in their result types) and as observers. Besides, preconditions should be formulated for partial functions. These preconditions are expressed by means of functions called "guards".

The specification may contain invariants expressed as functions.

RC Model Description

RC describes object classes at three different conceptual levels: specialization, realization and code. These names refer to the relations used to integrate specifications in the three levels. A more detailed description can be found in Felice, Leonardi, Favre, and Mauco (2001).

RC Components

The specialization level describes a hierarchy of incomplete RSL specifications as an acyclic graph. The nodes are related by the specialization relationship. In this context, it must be verified that if P(x) is a property provable about objects x of type T, then P(y) must be verified for every object y of type S, where S is a specialization of T.

Specialization level reconciles the need for precision and completeness in abstract specifications with the desire to avoid over-specification.

Every leaf in the specialization level is associated with a sub-component at the realization level. A realization sub-component is a tree of complete specifications in RSL:

- The root is the most abstract definition.
- The internal nodes correspond to different realizations of the root.
- Leaves correspond to sub-components at the implementation level.

If E1 and E2 are specifications E1 can be realized by E2 if E1 and E2 have the same signature and every model of E2 is a model of E1 (Hennicker & Wirsing, 1992).

Adaptation of reusable components, which consumes a large portion of software cost, is penalized by over-dependency of components on the physical structure of data.

The realization level allows us to distinguish decisions linked with the choice of data structure. In RAISE, there are four main specification style options. They are applicative sequential, imperative sequential, applicative concurrent and imperative concurrent (George, Haxthausen, Hughes, Milne, Prehn, & Pedersen, 1995). Associated with them, there are two styles: abstract and concrete. Imperative and concrete styles use variables,