SPECIFICATIONS REUSABILITY FROM RAISE METHOD

Laura Felice and Daniel Riesco*

INTIA. Departamento de Computación y Sistemas
Facultad de Ciencias Exactas. Paraje Arroyo Seco. Tandil. Argentina
Universidad Nacional del Centro de la Provincia de Buenos Aires
lfelice@exa.unicen.edu.ar
* Departamento de Informática
Facultad de Ciencias Físico Matemáticas y Naturales
Universidad Nacional de San Luis
Ejercito de los Andes 950. San Luis. Argentina
driesco@unsl.edu.ar

Abstract

During the RAISE specification development process, a variety of components and infrastructures are built. All these components are not independent, but they are related to each other, specially when we specify different systems into the same infrastructure. The RAISE method is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions. So, the reuse process is crucial in all the stages of the development, but there is not explicit reference to the specification reusability in this development process.
This paper presents a rigorous process for reusability for RAISE reusable component. We provide the mechanism to select a reusable component in order to guide RAISE developers in the software construction and specification.

1. Introduction

Software components are typically very rich in information, making the task to characterize them and capture their relevant properties difficult. However, this is not the only reason which makes software reuse difficult.
Information retrieval methods based on analyses of natural-language documentation have been proposed for constructing software libraries [9,11]. Software components represented by natural language can make the retrieval process a task with ambiguity, incompleteness and inconsistency. All these problems can be minimized by using a rigorous method to the retrieval a component.
The RAISE method [2] is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions.
Based on the above observation, we propose to introduce a Reusable Component (RC) model for the definition of the reusable component structure into RAISE.
In this work we propose the RC model for the definition of the structure of a reusable component that integrates specifications in RSL (RAISE Specification Language) [5] 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 relation.
- Realization: hierarchies of RSL complete algebraic specifications related by realization relations.
- Code: hierarchies of imperative RSL schemes related by implementation relations and linked to object-oriented code.

We define a rigorous process for reusability of RC components. Its manipulation, by means of specification building operators (Rename, Extend, Combine, 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.

2. Related Work

Different approaches to specify the reusable components functionality 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 RAISE method we emphasize the work of Beltaifa and Moore [1] where 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 [10] 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 y Cheng [3] is another approach that provides a formalism to register components 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 [13] is related to the specification matching. It is very important to emphasize this proposal has been referenced by a lot of authors.
The survey paper by Krueger [8], discussed different approach for software reuse. Eight categories, such as high-level languages, source code components, application generators, etc., are discussed.

3. RC Model Description

RC (Reusable Component) 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 [4].

3.1 RC components

The *specialization level* describes a hierarchy of incomplete RSL specifications as an acyclic graph. The nodes are related by specialization relations. 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 [10].
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 these 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 [6]. Associated with them, we can also distinguish between abstract and concrete styles. Imperative and concrete styles use variables, assignments, loops, channels (in concurrent specifications), etc; that are related to design decisions about data structures. Every specification at the realization level is linked to sub-components at the code level.
The *code* level groups a set of schemes in RSL associated with code. RAISE method provides translation processes, which start with a final RSL specification and produce a program in some executable language, for example C++ using the translation tool component of the RAISE toolset [7].

3.2 RC relationships

It is worth considering that the three relations (Specialization, Realization and Code) form the "RAISE implementation relation" [6]. Any formal system that aims to provide a means of specification and development must provide a notion of implementation. That is, if specification E1 is related to specification E2 in the model, we need to know if E1 and E2 are in the "RAISE implementation relation". The following properties must be satisfied:

- "Properties preservation: All properties that can be proved about E1 can also be proved for E2 (but not vice versa in general).
- Substitutivity: An instance of E1 in a specification can be replaced by an instance E2, and the resulting new specification should implement the earlier specification"

4. RAISE DEVELOPMENT PLAN APPLYING A REUSE MODEL

Usually, engineers proceed from applicative to imperative specifications. We propose to introduce the RC model for the definition of the reusable component structure into RAISE method. Where to introduce this model?. RAISE developers starts with the Module scheme specification, defines an abstract applicative module, develops a sequence of concrete applicative modules and the corresponding set of imperative modules from the final applicative modules. Summarizing, we can picture them as in figure 1.

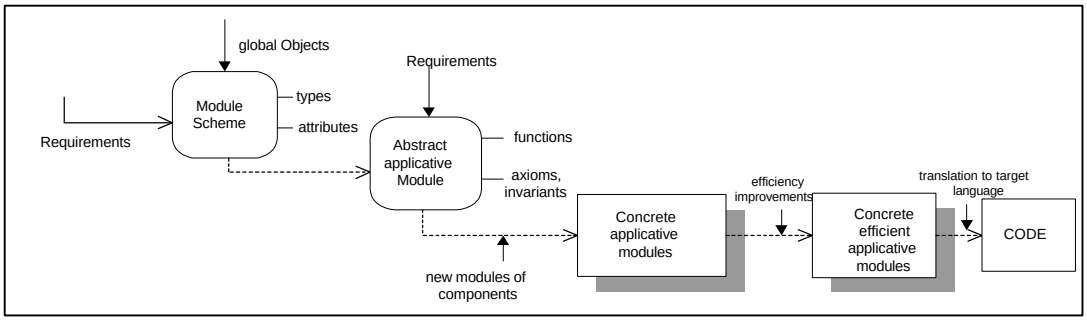

Figure 1. Overview of the RAISE Method

During the first stages of development, when the interaction with the stakeholders is crucial, the use of client-oriented requirements engineering techniques seems to be necessary in order to enhance the communication between the stakeholders and the software engineers. It has been proposed a systematic reuse approach that integrates natural language requirement specifications with formal specifications in RSL. Some heuristics are described to develop a formal specification in RSL starting from models belonging to the Requirements Baseline [12].

The objective is that engineers can make reuse in all development stages. We propose to introduce a RC model in all the development steps; in order to include abstraction, selection, specialization, and integration of software artifacts in the RAISE method.
Suppose that as part of a system implementation we need a component that we have specified with an Abstract applicative module specification $\Sigma_{Q(query)}$. Further, suppose that there is an abstract module in our library with specification $\Sigma_{L(library)}$ and its implementation has been verified to be correct with respect to the specification of Σ_Q. If we can show that Σ_Q is matched by Σ_L under generalized module match, then it is known that we can use the library module and the behavior will be consistent with that specified by Σ_Q. We are using specification match to check that using a library component will not "break" our system.
Thus, when we apply the mechanism to Concrete applicative module we are selecting an abstract applicative specification wholly adapted to the system requirements having it a translation to the concrete specification in the library.

5. THE REUSE PROCESS

Formal specifications are used to model the problem requirements and the function of the library components. The specifications are written in RSL language. The classification scheme consists of a collection of formal definitions representing possible component features in the domain. The formalization of the scheme permits automated classification of the specifications. The retrieval mechanism is based on syntactic comparison of features sets. The components returned by the retrieval mechanism are passed on to a more detailed evaluation that uses specification matching to determine reusability.
The results of specification matching determine the relationship that exists between the retrieved components and the requirements specification. The adaptation phase allows [4] to determine whether a mechanism exists to adapt or combine the retrieved components to solve the problem.
There is evidence that specification matching to determine component reusability can be carried out using automated theorem proving [13]. Attempting specification matching over a large library of components is not a practical retrieval mechanism. The idea is to work by classifying components in a way that components likely to match for reusability will be assigned similar features. By formally defining the classification features and the feature assignment process, classification could be automated.
The method has the following steps: *decomposition, identification, adaptation* and *composition* depicted in figure 2.

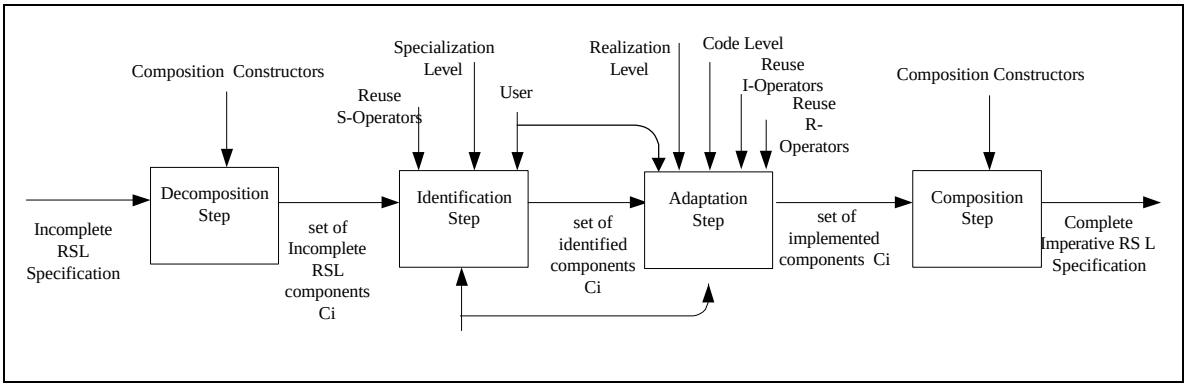

Figure 2. The method

In Decomposition step the decomposition of a goal specification Eg into sub-specifications E1, E2, ... En is formalized.
In Identification step for each specification Ei a component Ci (in the specialization level) and a sequence $s_1,s_2,...,s_n$ of RSL specifications must be identified, verifying the implementation relation. A leaf in Ci must be selected as a candidate to be transformed. The identification of a component is correct if it can be modified by rename, hide and extend operators to match the query Ei.
In Adaptation step, not only a leaf in the sub-component associated in the realization level but also a sequence of operators used in the previous steps are applied. Then, a scheme in the code level is selected and the same operators in the selected leaf are applied. Finally, in Composition step, the sub-specifications Ei and their implementations are composed. Here, we are concentrated in the Identification step.

5.1 RC Identification

Our classification components are described by a set of features. Each feature represents a property or an attribute of the component. Features can be refined to give them a more precise meaning. Our objective is to support effective retrieval of component. So, in order to be useful in practice the component specifications should be independent of the kind of component, allowing the storage of different kinds of them, e.g. requirements definitions, designs, documentation, etc..
Being a RSL specification a collection of modules, and a module basically a named collection of declarations either a scheme or an object -objects and schemes defined using class expressions-, we propose a classification based on a feature-oriented.
Each feature describes a property of the component, and is characterized by:

(a) *kind of component*: describing the function of different kinds of components like: requirements specifications, designs specifications, systems specifications, documentation, etc.;
(b) *operations* performed
(c) *component structure*: modules involved (schemes and objects).
    (c.1) granularity of each module: list of objects related with the class.
(d) *relationships* to another component ('implements' relations and composition of components)
(e) *component specification style* (applicative sequential, imperative sequential, imperative sequential, applicative concurrent or imperative concurrent and abstract and concrete styles).

Localized the possible components, the objective is to compare a component with the query. This process has two essential steps: signature matching and semantic matching. The signature matching enables a syntactic comparison of a query specification with specifications existing in RC reusable components. The semantic matching compares the specifications dynamic behavior. The bases of the signature matching come from [13], even though they were adapted to the identification of RC components.

6. Conclusions

In this paper a strategy to classify and select a reusable component is presented. We define the RC model, in the context of RAISE method, for the description of reusable components and a transformational process with reuse from RSL specifications to code.
Our goal is to solve a problem which is a weakness of RAISE formal method. RAISE method applies the RSL language in all the stages of the development. This method uses RSL to specify domains and requirements, and to design software. In this sense, the proposal is not only to apply the software components reuse but also a domain specifications reuse, i.e. in the confines of the domain engineering.
Currently, we are working on the application of the process we proposed to a concrete infrastructure. In particular, we are specifying the Agricultural Production component having the

Milk Production System [12] specification as a reusable component. Both of these components belong to the Agricultural System Infrastructure, also defined in [12].

References

[1] Beltaifa, R, Moore, R: A Software Reuse Infrastructure for an Efficient Reuse Practice. Technical report 230, UNU/ IIST, 2001. Macau www.iist.unu.edu.

[2] Bjorner, D. Software Engineering: A New Approach; Lecture Notes, Technical University of Denmark. 2000

[3] Chen, Y; Betty H. C. Cheng (1997) "Formally Specifying and Analyzing Architectural and Functional Properties of Components for Reuse". Proc. of 8th Annual Worshop on Software Reuse (WISR8).

[4] Felice L., Leonardi C., Favre L., Mauco V(2001). "Enhancing a rigorous reuse process with natural language requirement specifications". Proceedings of '2001 Information Resources Management Association International Conference'. Toronto. Canadá.

[5] George, C., Haff, P., Havelund, K., Haxthausen, A., Milne, R., Nielsen, C., Prehn, S., Ritter, K. (1992) "The RAISE Specification Language", Prentice Hall.

[6] George, C., Haxthausen, A., Hughes, S., Milne, R., Prehn, S., Pedersen, J. (1995) "The RAISE Development Method", Prentice Hall.

[7] George, C. RAISE Tools User Guide. Technical report 227, UNU/ IIST, 2001. , Macau www.iist.unu.edu.

[8] Krueger, C. (1992) "Software Reuse", ACM Computing Surveys, Vol 24, N° 2, June.

[9] R. Helm and Y.S, Maarek. Integrating Information Retrieval and Domain Specific Approach for Browsing and Retrieval in Object-Oriented Class Libraries. In Proceedings of OOSPLA91, pages 47-61, 1991.

[10] Hennicker,R., Wirsing, M.(1992)"A Formal Method for the Systematic Reuse of Specifications Components", Lecture Notes in Computer Science 544, Springer-Verlag.

[11] Y.S. Maarek, D.M. Berry, and G.E. Kaiser. An Information Retrieval Approach for Automatic Constructing Software Libraries. IEEE Trans. Software Engineering, 17(8):800-813, August 1991.

[12] Mauco, Virginia, George, C.(2000) "Using Requirements Engineering to Derive a Formal Specification". Technical Report 223, UNU/IIST, Macau www.iist.unu.edu.

[13] Zaremski, A., J. Wing, J. (1997) "Specification Matching of Software Components". ACM Transactions on Software Engineering and Methodology (TOSEM), Vol 6, No 4, 333-369.