IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4253

Applying A Reusable Component Model in Raise Formal Method

Laura Felice
INTIA, Departmento de Computacion y Sistemas, Facultad de Ciencias Exactas, Paraje Arroyo Seco, Argentina
Universidad Nacional del Centro de la Provincia de Buenos Aires, Tel: 54 2293 432530, lfelice@exa.unicen.edu.ar

Daniel Riesco
Departamento de Informatica, Facultad de Ciencias Fisico, Matematicas y Naturales
Universidad Nacional de San Luis, Argentina, driesco@unsl.edu.ar

ABSTRACT

There is a great diversity in the software engineering technologies that involve some form of software reuse. However, there is a commonality among the techniques used. For example, software component libraries, source code compilers, and generic software templates all involve abstracting, selecting, specializing, and integrating software artifacts.

In order to include them in the RAISE (Rigorous Approach to Industrial Software Engineering) method, we propose to introduce a RC (Reusable Component) model in all the development steps. The RAISE method is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions. There is not explicit reference to the specification reusability in this development process, so our model will allows to be an integrated mechanism to the method.

In this paper, we provide the mechanism to select a RSL (RAISE Specification Language) reusable component in order to guide RAISE developers in the software construction and specification. Thus, developers can efficiently locate, understand, compare and select the appropriate artifacts from a collection.

INTRODUCTION

Software reusability is the ability to construct new artifacts of software, using others existing in a different context from which they were originally developed. "If instead of being developed for just one project, a software element has the potential of serving again and again for many projects, it becomes economically attractive to submit it to the best possible quality techniques, such as formal specifications of components." [9].

It is impossible to manage a library with a large number of components without a systematic classification scheme, which splits object domains into sub-domains that can be identified. It is also convenient to reflect this division in the components. Software reusability takes many different requirements into account, some of which are abstract and conceptual while others are concrete and bound to implementation properties. Reusable components must be specified in an appropriate way. For example at more abstract levels we need descriptions satisfying three conditions:

- "They should be precise and an unambiguous.
- They should be complete or at least as complete as we want, in each case.
- They should not over-specify" [9]

At more concrete levels we need to include the specification together with the implementation, in the software itself.

There are many works that prove that software reusability can be addressed from formal descriptions [7]. In this work we propose the RC model for the definition of the structure of a reusable component that integrates specifications in RSL [4] and object-oriented code.

The results of RAISE [1] project are a "wide spectrum" specification and design language, an associated method [5] and a commercially available set of tools. The language allows us to specify in different styles: applicative or imperative, sequential or concurrent. The RAISE method is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions.

The 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. The manipulation of them by means of specification building operators (Rename, Extend, Combine, Hide) is the basis for the reusability.

The paper has the following structure: in Section 2 we give the most important related works. Section 3 gives a brief descriptions of RAISE and the RSL language. In Section 4 we present our strategy. Section 5 describes the RC model. Next, the Reuse Process is described finishing with conclusions.

RELATED WORKS

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. As a typical related work, we can mention Hennicker and Wirsing [6] 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 [2] 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 [11] is related to the specification matching. It is very important to emphasize this proposal has been referenced by a lot of authors.

Related with the mechanism to select a component, we can mention REBOUND [10], a framework whose objective is to guide in the searching for a solution based on existing libraries of components.

It is very interesting the work of Krueger [7], who makes an important comparative analysis for different approaches. He uses a taxonomy for their descriptions and comparisons in terms of reusable artifacts and the way in which these artifacts are abstracted, selected, specialized and integrated.

THE RSL LANGUAGE

The aim of the project RAISE, was to develop a language, techniques and tools that would enable industrial use of formal methods. The results of this project include the RSL Language which allows us to write formal specifications. In addition to this, a method to carry out developments based on such specifications, and a set of tools to assist

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

in edition, checking, transforming and reasoning about specifications [3] are provided.

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 us 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.
- 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.

RAISE DEVELOPMENT PLAN APPLYING A REUSE MODEL

Usually, engineers proceed from applicative to imperative specifications. We propose to introduce the RC (**R**eusable **C**omponent) model for the definition of the structure of a reusable component into RAISE method. Where to introduce this model?. RAISE developers begins 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 model

The objective is that engineers can make reuse in all of development stages. We propose to introduce a RC model in all the development steps; in order to include abstraction, selection, specialization, and integration software artifacts in the RAISE method.

THE RC MODEL

It 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. Figure 2 illustrates these levels

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 subcomponent 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 [6].

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 [5]. 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++.

It is worth considering that the three relations (Specialization, Realization and Code) form the "RAISE implementation relation" [5]. 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 in general vice versa).
- 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"

Where:

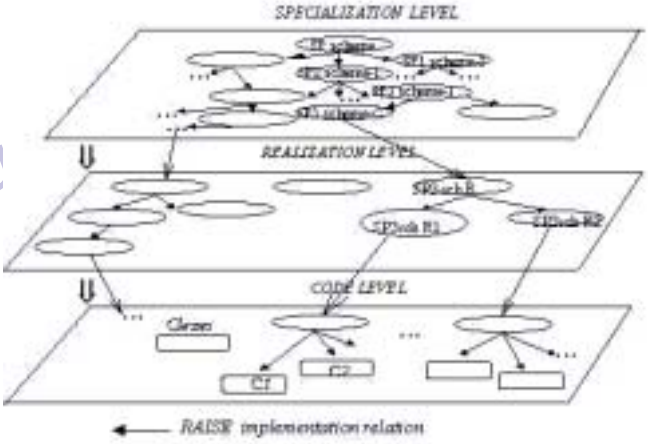
- *SPi scheme* is the most abstract RSL specification of the component module,
- *SPi scheme-I* are the incomplete RSL specifications of modules, in an abstract applicative style,
- *SPi scheme-C* are the complete RSL specifications of modules both in abstract applicative style and concrete applicative style,
- *SPi schR* are the complete RSL specifications of modules according to decisions like the use of databases, these are the concrete applicative specifications with efficiency improvements.

The primary idea is to provide a mechanism to identify RSL reusable components satisfying the user's query.

Formal Definition of RC Model

In this section the sub-components of the three conceptual levels are defined as follows:

Figure 2: Different conceptual models



Specification Level: the sub-components of this level are inductively defined by the *identify* operator, whose syntaxis is:

identify (*Sp*, {RCS_1, RCS_2,...,RCS_n})

where *Sp* is a RSL specification and RCS_i are reusable sub-components. Also, "j: 1<=j<=n: Root (RCS_j) 'is implemented by' *Sp*).

Realization Level: the sub-components of this level are inductively defined by the *realize* operator, whose syntaxis is:

realize (*S*, {RCR_1, RCR_2,...,RCR_n})

where *S* is a RSL specification and RCR_i are reusable components whose roots are realizations of S . This means ("j: 1<=j<=n: Sp 'is implementation of' Root (RCR_j)).

Each realization, can be view like a reusable component like *realize*(S,{}).

The definition of reusable sub-components in this level is related to the development of a sequence of concrete RAISE modules.

Code Level: the sub-components of this level are inductively defined by the *codify* operator, whose syntaxis is:

codify (*S*, {C_1, C_2,...,C_n})

where S is a complete specification and each C_j (1<=j<=n) is a concrete class in a programming language corresponding to the abstract specification of S.

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. Component retrieval is made efficient by layering a faceted classification scheme above specifications. 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 each of the retrieved components and the requirements specification. The adaptation phase serves [3] 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 [11]. 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 main idea is to transform the incomplete RSL specification into complete imperative specification by reusing existing components. The method has the following steps: *decomposition, identification, adaptation* and *composition* depicted in figure 3.

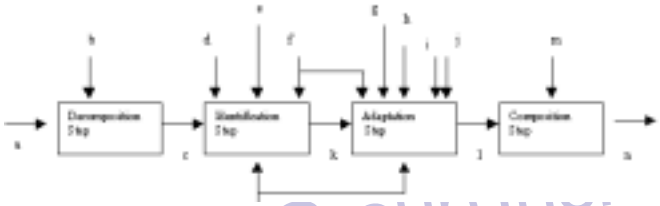
In Decomposition step the decomposition of a goal specification Eg into sub-specifications E1, E2, ..., En is formalized.

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.

In the next section, we only describe the Identification step because of space reasons.

Figure 3: The method



a: Incomplete RSL Specifiication
b: Composition Constructors
c: Set of Incomplete RSL components Ci
d: Reuse S-Operators
e: Specialization level
f: User
g: Realization level
h: Code level
i: Reuse I-Operators
j: Reuse R-Operators
k: Set of identified components Ci
l: Set of implemented components Ci
m: Complete Imperative RSL Specifications

RC Identification

In this section the use of specification matching to identify RC components is described. In the identification process, we search for all RC-components that satisfy a given query.

It must be able to find the component faster than user could build it, so, to address this problem it is necessary a classification scheme and then provide manual or automated retrieval techniques. We propose a classification based on:

- *functionality*: describing the function of the component.
- *operations* performed
- *component structure*: modules involved (schemes and objects).
- *relationships* to another component ('implements' relations and composition of components)
- *component specification style* (applicative sequential, imperative sequential, imperative sequential, applicative concurrent or imperative concurrent and abstract and concrete styles).

Figure 4 shows briefly the classification of a reusable component whose complete definition can be found in [8]. Milk Production System is a component belonging to the Agricultural System Infrastructure.

Localized the possible components, the objective is to compare a component with the query. This process has two essential steps: signa-

Figure 4: Milk production system component classification

- *functionality*: describe Milk Production System
- *operations*: records the events where animals participate. E.g.: births, milking, feedings,etc.
- *component structure:*
- *schemes:* {Dairy_Farm Cow_Gropus, Fields, Bulls, D_Farmers, Cow_Group, Cows, Field, Bull, D_Farmer ,Cow, Plots, Plot, History,Constants, Event_Info, Gropu_Event, Cow_Event, General_Types, Date}
- *objects:* {GH, GE, CH, CE, K, GT, D}
- *relationship*: -
- *specification style:* concrete applicative sequential

ture 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 [11], even though they were adapted to the identification of RC components.

The signature of a specification consists of a set of sorts and a set of operations, each operation being equipped with a particular functionality.

Let L=<SL, FL > be the signature of a library specification and Q=<SQ,FQ> the signature of a query specification where SL and SQ are set of sorts and FL and FQ are set of operation symbols, the signature matching is defined as follows:

Signature–Matching:

Query-Signature X RC-Library X Match-Predicate ®Set-of RC-Components

Signature-Matching(Q,C,P) = { c ∈ C/ P(Q,C)}

This means that given a query specification Q, a RC library C and a predicate P, it gives back the RC components that satisfy P. The signature matching is based on operations matching. Different kinds of operation matching can be applied. They are the exact, generalized and specialized matchings of operations. They can be extended for signatures of RSL specifications. This matching requires a specifications signature matching (sorts and operations) and the axioms proofs between pairs of operations.

CONCLUSIONS

In this paper a strategy to classify and select a reusable component is presented. We define the RC model 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. Therefore, 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.

In addition to this, we would like to mention that exists the possibility to automatically translate the specifications at code level in C++ language, using the code generator component of the RAISE toolset.

Many works proved that software reusability could be addressed from formal descriptions. Taking into account the four dimensions mentioned by Krueger [7], and comparing it with our proposal, we can say:

Decomposition and *Identification* steps correspond to Abstraction and Selection dimensions, because Abstraction is the essential feature in any reuse technique and Decomposition is related to obtain sub-specifications of a modular style of specification. On the other hand, Selection is related to locate, compare and select the reusable component and the Identification step is related to the component identification, its selection and the matching application.

Adaptation and *Composition* steps correspond to Specialization and Integration dimensions. Rigorously speaking, we can consider the Identification step into Specialization dimension too, because here we apply the operators to adapt the reusable component with rename, hide and extend operators. When we speak about Integration, we think of the combination of a collection of selected and specialized artifacts, in Composition step, the sub-specifications and their implementation are composed.

REFERENCES

[1] Bjorner, D. (2000) “Software Engineering: A New Approach”; Lecture Notes, Technical University of Denmark.

[2] Chen Yonghao and 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).

[3] 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.

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

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

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

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

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

[9] Meyer B (1997) “Object-Oriented Software Construction”, 2nd Edition, P.Hall.

[10] Penix, J. (1999) “REBOUND: A Framework for Automated Componet Adaptation”, WIR’99, Position Paper.

[11] 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.