Integrating Requirements Engineering Techniques and Formal Methods

María Virginia Mauco
Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina

Daniel Riesco
Universidad Nacional de San Luis, Argentina

INTRODUCTION

Formal methods help to develop more reliable and secure software systems, and they are increasingly being accepted by industry. The RAISE[1] Method (George et al., 1995), for example, is intended for use on real developments, not just toy examples. This method includes a large number of techniques and strategies for formal development and proofs, as well as a formal specification language, the RAISE Specification Language (RSL) (George et al., 1992), and a set of tools (George et al., 2001).

Formal specifications may be used throughout the software lifecycle, and they may be manipulated by automated tools for a wide variety of purposes such as model checking, deductive verification, formal reuse of components, and refinement from specification to implementation (van Lamsweerde, 2000a). However, they are not easily accessible to people who are not familiar or comfortable with formal notations. This is particularly inconvenient during the first stages of system development, when interaction with the stakeholders is very important. In common practice, the analysis of a problem often starts from interviews with the stakeholders, and this source of information is heavily based on natural language.

System requirements must be described well enough so that an agreement can be reached between the stakeholders and the system developers on what the system should and should not do. A major challenge with this is that the stakeholders must be able to read and understand the results of requirements capture. To meet this challenge we must use the language of the stakeholders to describe these results (Jacobson, Booch & Rumbaugh, 1999). Stakeholder-oriented requirements engineering techniques help to improve communication among stakeholders and software engineers as they ease the development of a first specification of a system which could be validated with stakeholders and which could be the basis for a formal development. Thus, we could take advantage of both techniques to improve the final product.

Among the techniques proposed to formalize requirements elicitation and modeling, Leite's Requirements Baseline can be mentioned (Leite et al., 1997). Two of its models are the Language Extended Lexicon (LEL) and the Scenario Model. LEL and scenarios provide a detailed description of an application domain, and as they are written in natural language, they are closer to stakeholder's world. However, an important point is how to fruitfully use all this information during the software development process.

To address the problems stated above, we have been working in the integration of stakeholder-oriented requirements engineering techniques with formal methods in order to take advantage of the benefits of both of them. In particular, our work is focused on the Requirements Baseline and the RAISE Method. We have proposed a three-step process to help in the definition of an initial formal specification in RSL of a domain starting from natural language models such as LEL and scenarios. These steps are the derivation of types, the derivation of functions, and the definition of modules. We have developed a preliminary set of heuristics that show how to derive types and functions, and how to structure them in modules by using LEL and scenarios information. We have also proposed to represent the hierarchy of RSL modules obtained using a layered architecture. This layered architecture is then the basis to start applying the steps of the RAISE Method.

BACKGROUND

In spite of the wide variety of formal specification languages and modeling languages, such as the Unified Modeling Language (UML) (Jacobson et al., 1999), natural language is still the method chosen for describing software system requirements (Jacobson et al., 1999; Sommerville & Sawyer, 1998; van Lamsweerde, 2000a). However, the syntax and semantics of natural language, even with its flexibility and expressiveness power, is not formal enough to be used directly for prototyping, implementation, or verification of a system. Thus, the requirements document written in natural language has to be reinterpreted by software engineers into a more formal design on the way to a complete implementation. Some

recent works (Lee, 2001; Lee & Bryant, 2002; Moreno Capuchino, Juristo & Van de Riet, 2000; Nuseibeh & Easterbrook, 2000, van Lamsweerde, 2000b) present different strategies for mapping requirements to, for example, object-oriented models or formal specifications.

When using the RAISE Method, writing the initial RSL specification is the most critical task because this specification must capture the requirements in a formal, precise way (George, 2002). RSL specifications of many domains have been developed by starting from informal descriptions containing synopsis (introductory text that informs what the domain is about), narrative (systematic description of all the phenomena of the domain), and terminology (list of concise and informal definitions, alphabetically ordered). Others also include a list of events. They can be found in UNU/IIST´s Web site *(*www.iist.unu.edu*)*. The gap between these kind of descriptions and the corresponding RSL formal specification is large, and thus, for example, it is difficult and not always possible to check whether the formal specification models what the informal description does and vice versa.

As we had some experience in using the Requirements Baseline, and we knew it had been used as the basis to an object conceptual model (Leonardi, 2001), we consider the possibility of using it as the first description of a domain from which a formal specification in RSL could be later derived.

THREE-STEP PROCESS TO DERIVE A FORMAL SPECIFICATION

As an attempt to reduce the gap between stakeholders and the formal methods world, we propose a technique to derive an initial formal specification in RSL from requirements models, such as LEL and scenarios that are closer to stakeholders' language. The derivation of the specification is structured in three steps: Derivation of Types, Derivation of Functions, and Definition of Modules. They are not strictly sequential; they can overlap or be carried out in cycles. For example, function definitions can indicate which type structures are preferable.

Derivation of Types

This step produces a set of abstract as well as concrete types that model the relevant terms in the domain. We perform the derivation of the types in two steps. First we identify the types, and then we decide how to model them. This way of defining types follows one of the key notions of the RAISE Method (George et al., 1995): the step-wise development.

The main goal of the identification step is to determine an initial set of types that are necessary to model the different entities present in the analyzed domain. This initial set will be completed, or even modified, during the remaining steps of the specification derivation. For example, during the Definition of Modules Step, it may be necessary to define a type to reflect the domain state. Also, when defining functions, it may be useful to define some new types to be used as result types of functions. The LEL is the source of information during this step, as LEL subjects and some objects represent the main components or entities of the analyzed domain. In general, LEL subjects and objects will correspond to types in the RSL specification. In some cases, LEL verbs may also give rise to the definition of more types, as when they represent an activity that has its own data to save. However, in order to define just the relevant types, we have suggested some heuristics that can be found in Mauco, Riesco, and George (2001a).

Once a preliminary set of types is defined and in order to remove under-specification, we propose to return to the information contained in the LEL and the Scenario Model. In particular, the analysis of the notion, and sometimes the behavioral response of each symbol that motivated the definition of an abstract type, can help to decide if the type could be developed into a more concrete type. All the developments we suggest satisfy the implementation relation. In Mauco et al. (2001a), some heuristics to assist in this task can be found. During this step, it might be necessary to introduce some type definitions that do not correspond to any entry in the LEL. They appear, in general, when modeling components of some other type. Symbols without an entry in the LEL may represent an omission or a symbol considered outside the application domain language. When an omission is detected, it is necessary to return to the LEL to add the new definition, and update the Scenario Model to maintain the consistency between its vocabulary and the LEL itself.

Definition of Modules

This step helps to organize in modules all the types produced by the Derivation of Types Step in order to obtain a more legible and maintainable specification. These modules would be later completed with the definition of functions in the next step, and probably they will be completed with more type definitions.

A summary of the heuristics we propose to define for the modules can be found in Mauco et al., (2001b). In defining these heuristics, we closely followed the features RSL modules should have according to the RAISE Method (George et al., 1995; George, 2002). For example, each module should have only one type of interest, defining the appropriate functions to create, modify, and observe values of the type, and the collection of modules should be, as far as possible, hierarchically structured.