# Deriving Formal Specifications from Natural Language Requirements

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

**María Carmen Leonardi**
*Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina*

**Daniel Riesco**
*Universidad Nacional de San Luis, Argentina*

## INTRODUCTION

Formal methods have come into use for the construction of real systems as they help to increase software quality and reliability, and even though their industrial use is still limited, it has been steadily growing (Bowen & Hinchey, 2006; van Lamsweerde, 2000). When used early in the software development process, they can reveal ambiguities, incompleteness, inconsistencies, errors, or misunderstandings that otherwise might only be discovered during costly testing and debugging phases.

A well-known formal method is the RAISE Method (George et al., 1995), which has been used on real developments (Dang Van, George, Janowski, & Moore, 2002). One tangible product of applying a formal method is a formal specification. A formal specification serves as a contract, a valuable piece of documentation, and a means of communication among stakeholders and software engineers. 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, animation, test data generation, formal reuse of components, and refinement from specification to implementation (van Lamsweerde, 2000). However, one of the problems with formal specifications is that they are hard to master and not easily comprehensible to stakeholders, and even to non-formal specification specialists. This is particularly inconvenient during the first stages of system development when interaction with stakeholders is very important. In practice, the analysis often starts from interviews with the stakeholders, and this source of information is heavily based on natural language as stakeholders must be able to read and understand the results of requirements capture. Then specifications are never formal at first. A good formal approach should use both informal and formal techniques (Bjorner, 2000).

The requirements baseline (Leite, Hadad, Doorn, & Kaplan, 2000), for example, is a technique proposed to formalize requirements elicitation and modeling, which includes two natural language models, the language extended lexicon (LEL) and the scenario model, which ease and encourage stakeholders' active participation. However, specifying requirements in natural language has some drawbacks related to natural language imprecision.

Based on the previous considerations, we proposed a technique to derive an initial formal specification in the RAISE specification language (RSL) from the LEL and the scenario model (Mauco, 2004; Mauco & Riesco, 2005a; Mauco, Riesco, & George, 2004). The technique provides a set of manual heuristics to derive types and functions and structure them in modules taking into account the structured description of requirements provided by the LEL and the scenario model. But, for systems of considerable size this manual derivation is very tedious and time consuming and may be error-prone. Besides, maintenance of consistency between LEL and scenarios, and the RSL specification is a critical problem as well as tracking of traceability relationships.

In this article, we present an enhancement to this technique, which consists in the RSL-based formalization of some of the heuristics to derive RSL types from the LEL. The aim of this formalization is to serve as the basis for a semiautomatic strategy that could be implemented by a tool. More concretely, we describe a set of RSL-based derivation rules that will transform the information contained in the LEL into abstract and concrete RSL types. These derivation rules are a useful starting point to deal with the great amount of requirements information modeled in the LEL, as they provide a systematic and consistent way of defining a tentative set of RSL types. We also present some examples of the application of the rules and discuss advantages and disadvantages of the strategy proposed.

## BACKGROUND

In spite of the availability of other notations such a tables, diagrams, and formal notations, natural language is still

chosen for describing the requirements of a software system (Berry, Bucchiarone, Gnesi, Lami, & Trentani, 2006; Bryant et al., 2003; van Lamsweerde, 2000).

The language extended lexicon (LEL) and the scenario model are two well known natural language requirements models used and accepted by the requirements engineering community (Leite et al., 2000). The LEL aims at registering significant terms in the Universe of Discourse (UofD). Its focus is on the application domain language, rather than the details of the problem. It unifies the language allowing the communication with stakeholders. LEL is composed by a set of symbols that represent words or phrases that stakeholders repeat or emphasize. Each entry in the LEL has a name (and possibly a set of synonyms), and two descriptions: notion, which describes what the symbol is, and behavioral response, which describes how the symbol acts upon the system. Each symbol is classified as object, subject, verbal phrase, or state. Figure 1 shows an example of an object LEL symbol taken from the LEL of the milk production system (Mauco, 2004). Underlined words or phrases correspond to other LEL symbols. The scenario model contains a set of scenarios where each scenario describes a situation in the UofD. Scenarios are naturally linked to the LEL. This link is reflected by underlying LEL symbols every time they appear in a scenario description. Figure 2 shows an example of a scenario with all its components.

In order to take profit of natural language requirements specifications, it would be necessary to look at ways for mapping the conceptually richer world of requirements engineering to more formal designs on the way to a complete implementation (Nuseibeh & Easterbrook, 2000). Many works aiming at reducing the gap between the requirements step and the next steps of software development process have been published. Some of them describe, for example, different strategies to obtain object oriented models or formal specifications from requirements specifications (Bryant et al., 2003; Díaz, Pastor, Moreno, & Matteo, 2004; Juristo, Moreno, & Lopez, 2000; Lee & Bryant, 2002).

The RAISE method includes a large number of techniques and strategies for doing formal development and proofs, as well as a formal specification language, RSL, and a set of tools to help writing, checking, printing, storing, transforming, and reasoning about specifications (George, 2001, 2002; George et al., 1995). Usually the first RSL specification is

*Figure 1. Field LEL symbol*

**FIELD**
*Notion*
- Land where cows eat pastures.
- It has an identification.
- It has a precise location in the dairy farm.
- It has a size.
- It has a pasture.
- It has an hectare loading.
- It is divided into a set of plots.
- It has a list of previous plots.

*Behavioral Response*
- A dairy farmer divides it into a set of plots, separated by electric wires.
- Many different groups of cows can be eating in it simultaneouslv.

*Figure 2. Feed a group*

**TITLE:** Feed a group
**GOAL:** Register the daily ration given to a group of cows.
**CONTEXT:** It is done once a day. Pre: Group is not empty.
**RESOURCES:** Group Date Quantity of corn silage Quantity of Hay Quantity of concentrated food Feeding form
**ACTORS**: Dairy farmer
**EPISODES**:
- COMPUTE RATION.
- The dairy farmer records, in the Feeding form, the date and the quantities of corn silage, hay and concentrated food given to each cow in the group.
- COMPUTE PASTURE EATEN.