SAREL: An Assistance System
for Writing Software Specifications
in Natural Language

Núria Castell
Àngels Hernández

Report LSI-94-17-R
SAREL: An Assistance System for writing Software Specifications in Natural Language

N. Castell, A. Hernández
Departament de Llenguatges i Sistemes Informàtics
e-mail: castell@lsi.upc.es
email: ahernandez@lsi.upc.es

ABSTRACT

The specification phase is one of the most important and least supported part of software development process. In order to improve the quality control of software at this earlier phase, it is important to identify the quality properties that would be accomplished by a set of requirements forming a specification. Among the well-known quality properties, we are interested in five of them that we consider the most important: consistency, completeness, traceability, verifiability and modifiability.

Following a brief description of three systems focused on the quality of the specification phase, we present our system (SAREL). Its purpose is to assist engineers in the creation of software specifications written in natural language. The assistance process takes into account the writing norms and the above mentioned quality properties. To do so, the SAREL system relies on a variety of controls that validate the set of requirements using lexical, syntactic, semantic and domain information. SAREL distinguishes two kinds of knowledge: the Knowledge Base, which contains domain representation, and the Requirements Base, which contains requirements representation. Once a requirement has been validated and is correct, its conceptual representation is added to the Requirements Base.

RESUM

La fase d’especificació és una de les parts menys controlades en el procés de desenvolupament de software. Per a controlar la qualitat del software en aquesta fase és important identificar les propietats de qualitat que haurien de ser presents en el conjunt de requeriments que formen l’especificació. Entre les diferents propietats de qualitat conegudes, considerem les més importants: consistència, complitesa, traçabilitat, verificabilitat i modificabilitat.

Després d’una breu descripció de tres sistemes que incideixen en la qualitat a fase d’especificació, presentem el nostre sistema SAREL. El seu objectiu és ajudar a l’enginyer en la creació d’especificacions de software escrites en llenguatge natural. Aquest procés d’ajut està basat en les normes de redacció i en les propietats de qualitat mencionades anteriorment. El sistema SAREL realitza una varietat de controls que validen el conjunt de requeriments utilitzant informació lèxica, sintàctica, semàntica i del domini. SAREL diferencia dos tipus de coneixement: la Base de Coneixements que conté la representació del domini i la Base de Requeriments que conté la representació dels requeriments. Una vegada un requeriment ha estat validat i és correcte, la seva representació conceptual és afegeta a la Base de Requeriments.
1 Introduction

The software development process starts generally with the specification phase. In this stage it is very important to control the quality of the specifications in order to detect possible mistakes as earlier as possible. The correction of errors on the development and implementation phases implies spending more time and efforts than doing on the specification phase. This is the reason why the developers increasingly try to identify the possible mistakes on the early phases of software development.

During the initial phase, the software specifications for complex systems (such as those in the aerospace and nuclear fields) result in bulky documents since they are often written in natural language. Even after being formalized, this original documentation may also serve during later phases, and during the functioning and maintenance of the developed computer system. Documentation writing is guided by the norms which define the linguistic restrictions required to satisfy the specifications. These norms are of two types: those relating to the use of natural language in general (e.g., [4] and [2]); and those that are based on terminological restrictions related to a particular domain (e.g., the ESA - European Space Agency - norms). Both of them restrict the use of natural language through a set of rules which limit various irregularities (polysemy, paraphrase, ambiguity, vagueness) which occur during the interpretation of natural language. Even though the norms define linguistically precise restrictions, the frequent failure to observe them makes it difficult for the consequence of such breaches to be detected afterwards. In addition to linguistic restrictions, the norms also include Software Engineering constraints related to the quality factors of the specifications, such as consistency, completeness, traceability, modifiability and verifiability.

In order to assist the engineer in writing specifications in natural language our system takes into account the writing norms and the software quality properties.

In this paper we describe, in section 2, a set of systems which tackle the problems associated with the specification phase. This description serves as a reference frame of our research work. Later, in section 3, we present our system (SAREL). Its purpose is to assist engineers in the creation of software specifications written in natural language. In section 4, we analyse the common aspects between SAREL and the systems described in section 2. This paper concludes with a discussion of the future goals of our research.

2 An overview of different systems

In this section we describe three systems supporting the specification phase. The Requirements Apprentice (RA) assists a human analyst in the creation and modification of software requirements. The FRORL development system facilitates the specification, analysis and development of a software system. The ISLET system supports the construction of
formal specifications and their incremental refinement into verified implementations. Although the above system goals are even different from SAREL system goal, they form a good set of references to place our research work.

We also want to mention the NATURE project [13], although it is not described in this paper. It is a more ambitious project which goal is to develop a set of interacting theories that relate knowledge representation, domain analysis and process support aspects of requirements engineering. We would notice that NATURE has several common aspects with LESD project, which contains the SAREL system, in spite of LESD not being so ambitious as NATURE.

2.1 RA : Automated Assistance for Requirements Acquisition

The focus of the RA [7], [8], [9] is on the formalization phase that bridges the gap between an informal and formal specification. Figure 1 shows the role of the RA in relation to other agents involved in the software process.

![Figure 1. The Requirements Apprentice](image)

RA is composed of three modules: Cake, Executive and Cliché Library. Cake is a knowledge-representation and reasoning system, which supports the reasoning abilities of the RA. Each layer of Cake builds on facilities provided by the more primitive layer below.

- The Propositional layer of Cake provides four principal facilities: it performs simple one-step logical deductions, supports a general mechanism for the pattern-directed invocation of demons, detects a certain class of shallow contradictions, and it acts as a recording medium for dependencies.

- The Algebraic layer of Cake contains special-purpose decision procedures for equality, common algebraic properties of operators (such as commutativity, associativity, and transitivity), partial functions, and algebra of sets.
- The Frames layer of Cake supports the standard frame notions of inheritance, slots, and instantiation. It is used to represent the clichés in the requirements cliché library.

- The Plan Calculus layer of Cake supports an internal representation for algorithms that is well suited for automated reasoning and other manipulations.

In general Cake provides basic facilities for propositions deduction (including the detection of contradictions), reasoning about equalities, maintenance of dependencies between deduced facts, and incremental retraction of previously asserted facts.

The Cliché Library is a declarative repository of information relevant to requirement in general and to domains of particular interest. When creating a requirement, the information unique to the particular problem comes from the analyst. However the bulk of general information about the domain comes from the cliché library.

The Executive handles interaction with the analyst and provides high-level control of the reasoning performed by Cake. The analyst communicates with the Executive by issuing commands. Each command provides fragmentary information about an aspect of the requirement being specified. The immediate implications of a command are processed by Cake, added to the RKB (Requirements Knowledge Base) and checked for consistency. If the analyst makes a change in the description the Executive incrementally incorporates this modification into the RKB, retracting invalidated deductions and replacing them with new deductions.

The RA checks the requirement for consistency, completeness, ambiguity and provides specific support for evolutionary modification of the requirement.

2.2 FRORL Development System

The software development methodology described in [5], based on frames and rules, applies the concepts of top-down design and step by step construction to develop a software system. The main idea of this methodology is that a system can be constructed by dealing with the most general and important concepts in the application domain first, and then proceeding with subsystems through more particularization. Using this methodology to construct specifications, one can create higher-level frames to represent the main features and uses lower level frames to describe the minute demands. The user and the developer can understand and validate the main functions of the developed system at an early stage of software development. The language FRORL is based on this methodology, which is composed of the following steps:

- Identify subject and themes.

- Define object frames.
- Define objects abstract inheritance relations.
- Define object attributes.
- Identify activity frames.
- Define actions and communications.

A frame-and-rule based software system has been developed to support software development using FRORL. This system plays an important role in assisting the user to bridge the gap between informal conceptual model and formal specification. The development process guides the users of FRORL in systematically constructing specifications which model the problem domain at hand. Figure 2 gives an overview of this software system.

Figure 2. Overview of software development using FRORL

This software development system starts from the formation of informal requirements by the clients. The developers, henceforth, do an initial analysis on this informal requirements, and then build a conceptual model. By using the frame-and-rule based development methodology, the developers can therefore create a FRORL specification from the conceptual model. Once a syntactically correct FRORL specification is obtained, this
specification can be subject to various analysers which attempt to verify several software properties: reachability, reversibility, liveness, consistency, synchronic distance and bounded fairness. The output is an analysis report, which shows the structural properties of the specification. After an iterative sequence checking, review, and modification, a prototype can be derived. At the prototype evaluation phase, the client and the developer cooperate in evaluating the prototype. There is another iterative sequence of prototype generation, prototype evaluation, and report review. The report generated in this phase shows the run-time behavior of the specification. After the prototype specifications have been satisfied, implementation of the system should proceed by application of correctness-preserving transformations. All these activities are supported using a knowledge base.

2.3 ISLET: A Prototype Tool to support the Gries/Dijkstra method

Unlike the above systems, ISLET [10] deals with formal specifications and their incremental refinement into verified implementations. Some refinement steps generate verification conditions that must be true for the step to be correct.

![Diagram of ISLET architecture](image)

Figure 3. Architecture of ISLET

The system operates on a set of symbol tables that are stored in a module data base. The user interacts with three subsystems: the editing tool allows programs to be
entered and modified, the proof tool supports the mechanical certification of verification conditions, and the cliche tool gives access to a library of cliches representing solutions to common programming problems. All of these subsystems use an algebraic simplifier to reduce logical formulae to more compact forms.

This system supports simple stepwise refinement with proof by automatically generating and mechanically certifying verification conditions. In addition, through ISLET the programmer has access to a library of pre-verified cliches that can be used to create programs more easily.

3 SAREL: An Assistance System for writing Software Specifications in Natural Language

The SAREL system is part of a more ambitious program of research and development called LESD (Linguistic Engineering for Software Design). This project [6], [12] aims to develop computational tools which will (1) allow conceptual interpretation of functional or preliminary software aerospace specifications written in English; (2) permit evaluation of quality factors by means of reasoning algorithms applied to the conceptual representation;
and (3) help the engineers handle documentation.

![Diagram of validation process by SAREL](image)

Figure 4. Validation process by SAREL

In this context, the main goal of SAREL is to assist an engineer in the creation of software specifications written in natural language. The assistance process, decomposed in several steps (see figure 4), validates every requirement introduced by the engineer taking into account the writing norms (for instance [2], [4]) and the quality properties [11]. This process incrementally constructs a conceptual representation of the specification. The controls showed in figure 4 can be grouped into two modules: the Style Refinement Module and the Conceptual Refinement Module. The first module analyses the quality of the requirement itself, and the second one analyses the quality of the requirement taking into account the set of requirements represented in the Requirements Base. Once
a requirement has been checked and is correct, its conceptual representation is added to the Requirements Base. Next, we describe in detail these two modules as well as the different analysis and controls that are being implemented.

3.1 Style Refinement Module

This module controls the requirement according the writing norms and is composed of four steps: lexical analysis, syntactic-semantic analysis, ambiguity control and simplicity control. The style refinement module controls at first the lexicon used in a requirement, then analyses its coherence and finally validates its surface form. The examples we present in this paper are taken from an aerospace system documentation.

3.1.1 Lexical Analysis

This analysis carries out the control of lexicon used in the specification. Given a requirement, the lexical analysis verifies that the words belong to the application domain lexicon. To do so, SAREL uses three different lexicons:

- The extended lexicon includes all words related to the application domain.

- The allowed lexicon is an extraction of the previous one. Every word in the allowed lexicon represents a set of synonymous on the extended one.

- The ground lexicon contains the basic words in the English language (prepositions, articles, ...).

The input of this analysis is a sentence composed of words that belong to the extended and ground lexicons. If the requirement contains at least one word not belonging neither to the extended nor to the ground lexicons, it will be refused. The output is the same requirement described using the allowed and ground lexicons. Next we present an example of lexical refinement:

Requirement : “The IOIGS shall manage the computer o.n-board the space-vehicle”

Answer : “The word manage is not allowed. The suggested expression for this requirement is: The IOIGS shall monitor the computer on-board the space-vehicle”

The lexicon division offers the engineer some kind of writing flexibility, because he can use an extended lexicon which will be refined on to the allowed lexicon. Moreover, the change of application domain implies only the change of extended and allowed lexicon, because the lexical validation process is the same.
3.1.2 Syntactic-Semantic Analysis

In this step, the ALVEY parser [3] is used. This parser has been adapted to our domain. Although semantic rules are associated to syntactic ones, the GDE (Grammar Development Environment) is not able to capture the whole meaning of a requirement and a conceptual analysis must be further performed. The output is a tree-like semantic representation. Figure 5, from [12], shows the semantic representation for the requirement: "The IOIGS shall monitor the status of the space vehicle"

![Semantic Representation Diagram]

Figure 5. Semantic Representation

3.1.3 Ambiguity Control

It is possible to obtain more than one semantic representation from a requirement. This fact appears when there exists some kind of ambiguity in the requirement. The function of the ambiguity controller is to distinguish the representation which corresponds to the engineer idea. For example from the requirement "The IOIGS shall monitor the computer on-board and the status of the space-vehicle" we can get two possible interpretations:

a) "The IOIGS shall monitor the computer on-board of the space-vehicle and the IOIGS shall monitor the status of the space vehicle"

b) "The IOIGS shall monitor the computer on-board and the IOIGS shall monitor the status of the space vehicle"
The ambiguity controller applies some rules to the set of semantic representations in order to qualify them. It then cooperates with the engineer to decide which the correct semantic representation is.

3.1.4 Simplicity Control

The function of this control is to analyse the simplicity of the requirement. This is one of the properties expressed by the writing norms. From the semantic representation of a requirement, the simplicity controller detects whether the structure of the sentence is simple or compound. The following requirement does not accomplish this property.

Requirement: "The IOIGS shall control the computer on-board of the space-vehicle and the IOIGS shall control the automatic systems of the flight configuration"

Answer: "The requirement is a conjunction of two simple requirements"

If the requirement is composed of two simple requirements, there exists the possibility that the process continues with these two requirements in a sequential way.

3.2 Conceptual Refinement Module

This module carries out a group of controls that validate the requirement in relation to the Requirements Base (RB). At first it obtains a conceptual representation using the Knowledge Base (KB). Both RB and KB use a frame-based formalism [12]. From de conceptual representation this module controls the duplicity of information and finally it validates the requirement activating the reasoning tools developed in LESD to control the quality properties (completeness, traceability, consistency, verifiability and modifiability). We have studied so far, the traceability and the completeness properties.

3.2.1 Conceptual Analysis

This analysis identifies in the Knowledge Base the concepts involved on the semantic representation. From this information it constructs a conceptual representation of the requirement. Figure 6, from [12], shows the conceptual representation of the requirement:
"The IOIGS shall monitor the computer on board the space-vehicle"

Figure 6. Conceptual Representation

3.2.2 Duplicity Control

The function of this control is to validate that the requirement introduced by the engineer contains new information. To do so, the Duplicity Controller matches the Conceptual Representation of a requirement to the Requirements Base in order to discover possible duplicities. The following example shows two equivalent requirements since the system obtains the same conceptual representation for both.

Req1: "The IOIGS shall monitor the computer on-board the space-vehicle"

Req2: "The computer on-board the space-vehicle shall be monitorized by the IOIGS"

The activity in both is monitor, the agent which performs the action is the IOIGS and the object, which suffers the actions in both, is the computer on-board the space-vehicle. If this Controller detects duplicity in the introduced requirement it will offer to the analyst the possibility to refine the requirement.
3.2.3 Analysis of Traceability

The goal here is offering information to the engineer about the traceability links of the requirement. To do so, the system activates a set of algorithms already developed which control the traceability in order to show the relationships between the introduced requirement and a subset of requirements of the RB, either more specific or more general. From this information the engineer knows the relationships between requirements introduced up to this point. From the next requirement:

Req1. : "The IOIGS shall receive the data of the space vehicle"

the system displays, among others, the following more general requeriment:

Req2. : "The system shall process the data of the space vehicle"

The second requirement is more general than the first because IOIGS is an instance of system and receive is a subclass of process.

3.2.4 Analysis of Completeness

Also in this step the system activates the reasoning mechanisms of LESD. In order to control the completeness, a general hierarchy of actions-subactions that can be associated to any set of specifications related to a aerospace system [1] is necessary. An exemple of this hierarchy is present in the activity monitor, which is composed of three subactivities: receive, analyse and display. Given the next requirement:

Req. : "During de launch phase, the IOIGS shall analyse and display the status of the space vehicle"

Taking into account the hierarchy described above, the system analyses the relationships amongst requirements in the Requirements Base. If there is no requirement containing the monitor activity, it will inform the engineer. In the same way, if the system notices that there is a requirement that contains the monitor activity but there is no requirement containing the subactivities on the hierarchy, it will suggest the engineer that the current specification is incomplete.

Once a requirement has been validated and is correct it will be added to the Requirements Base. This process incrementally constructs the conceptual representation of the specification written in natural language.

4 A Comparative Analysis

In this section we present a comparative analysis of the three systems we have described in section 2, in order to place our research work.
4.1 RA & SAREL

The Requirement Apprentice (RA) is the closest system to SAREL. The input for RA is a set of commands while SAREL accepts specifications in natural language, therefore RA does not consider linguistic problems as SAREL do. Nevertheless, we have identified a set of common elements between these two systems:

a) The Library Cliché corresponds to the Knowledge Base since the library contains information about the application domain.

b) The Requirements Knowledge Base in RA is related to the Requirements Base in SAREL.

c) The Cake module corresponds to the Conceptual Refinement Module, because it provides basic facilities in order to validate the set of requirements.

According quality control, the RA checks the requirements taking into account four properties: disambiguity, consistency, completeness and modifiability. On the other hand, the conceptual validation in SAREL is more extensive and is based on writing norms and the five mentioned properties: traceability, completeness, consistency, verifiability and modifiability.

4.2 FRORL Development System & SAREL

To compare these systems, we keep in mind that their goals are different. The FRORL Development System aims at supporting the software development. Starting from a set of informal requirements, the system achieves its corresponding Prolog code. The goal of SAREL is to obtain a validated conceptual representation from a specification written in natural language. In spite of that, we have detected some similarities between SAREL and the early stage of the architecture of FRORL Development System:

a) The start point in both systems is a set of informal requirements. FRORL development system begins from a requirement document while the input of SAREL is a specification written in natural language. However, in the first system the developer translates the requirements into FRORL sentences, while SAREL supports a really true natural language processing.

b) The Requirements Analysis in FRORL Development System corresponds to the Style Refinement Module and the construction of the Knowledge Base in SAREL. On the one hand the developers using the FRORL Development System identify the concepts that characterize the application domain. On the other hand they solve the problems related to the writing style (ambiguity, simplicity, ...).
c) The FRORL development methodology is related with the conceptual analysis, because it describes the specifications in a fixed form susceptible to be checked.

d) The Specification Analysis corresponds to the set of controls performed on the Conceptual Refinement Module. The output in both systems is a report which shows the properties accomplished by the specification.

The set of properties to be validated is different from one system to the other. The Specification Analysis verifies software properties and the Conceptual Refinement Module validates requirements properties.

4.3 ISLET & SAREL

In spite of the differences between these two systems, the idea of refinement is implicit in both. The design derivation process in ISLET uses stepwise refinement to transform pre and post-condition specifications into verified programs. The assistance system SAREL uses stepwise refinement to transform a specification written in natural language into its validated conceptual representation.

5 Conclusion and Future Research

Our research deals with the control of quality in specifications written in natural language. Since the specification phase is performed by a human analyst, we have designed an assistance system for writing software specifications in natural language. The quality of the specification is studied from two points of view: writing norms (style refinement) and quality properties (conceptual refinement). The quality properties we are interested in are: traceability, completeness, verifiability, consistency and modifiability.

The assistance process is divided into two modules: the Style Refinement Module and the Conceptual Refinement Module. Once a requirement has been validated by these two modules, it is added to the Requirements Base. During this process we incrementally obtain a conceptual representation of the specification.

Some of our future research directions are the global implementation of controls contained in both modules and the development of new controls related to the verifiability, consistency and modifiability properties.

References


Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1994


LSI-94-2-R “Degrees and reducibilities of easy tally sets”, Montserrat Hermo.

LSI-94-3-R “Isothetic polyhedra and monotone boolean formulae”, Robert Juan-Arinyo.

LSI-94-4-R “Una modelización de la incompletitud en los programas” (written in Spanish), Javier Pérez Campo.


LSI-94-6-R “Construction of the Face Octree model”, Núria Pla-Garcia.

LSI-94-7-R “On the expected depth of boolean circuits”, Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI-94-8-R “A transformation scheme for double recursion”, José L. Balcazar.


LSI-94-10-R “Relative knowledge and belief: SKL preferred model frames”, Natías Alvarado.

LSI-94-11-R “A top-down design of a parallel dictionary using skip lists”, Joaquim Gabarró, Conrado Martínez, and Xavier Messeguer.

LSI-94-12-R “Analysis of an optimized search algorithm for skip lists”, Peter Kirschchenhofer, Conrado Martínez, and Helmut Prodinger.

LSI-94-13-R “Bases de dades bitemporals” (written in Catalan), Carme Martín and Jaume Sistac.

LSI-94-14-R “A volume visualization algorithm using a coherent extended weight matrix”, Daniela Tost, Anna Puig, and Isabel Navazo.


LSI–94–18–R "Medición del factor modificabilidad en el proyecto LESD" (written in Spanish), Núria Castell and Olga Slávкова.

LSI–94–19–R "Algorismes paral·lels SIMD d'extracció de models de fronteres a partir d'arbres octals no exactes" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI–94–20–R "Una paral·lelització SIMD de la conversió d'objectes codificats segons el model de fronteres al model d'octrees clàssics" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secret@lsi.upc.es