Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns

Sri Fatimah Tjong Nasreddine Hallam Michael Hartley
University of Nottingham Malaysia Campus
{kcx4sfj, Nasreddine.Hallam, Michael.Hartley}@nottingham.edu.my

Abstract

This paper presenst an approach for reducing the problem of ambiguity and imprecision in natural language requirements specifications with the use of language quality patterns and guiding rules. To ensure the applicability of our approach, we study different sets of requirements documents from several domains. We further validate our approach by rewriting the requirements statements derived from these requirements documents.

Keywords: Natural Language Requirements Specifications, Guiding Rules, Language Patterns

1. Introduction

Requirements engineering, being the core of software development, is concerned with identifying the purpose of a software system and the contexts in which it will be used. It also facilitates effective communication of the requirements among different stakeholders, users, and clients. In general, some requirements are not properly communicated and documented, which results in incorrectness, inconsistency, incompleteness, and even misinterpretation. More importantly, the inherent ambiguity of natural language is another issue of requirements represented in natural language.

To reduce the ambiguity in natural language, several authors have proposed the use of different modeling techniques and methods as summarised in QUASAR [3]. Some have even developed a controlled language for specifying requirements in an almost natural language [10]. These methods are either the formal languages expressing the requirements or a set of procedures formalising the requirements. Formal languages use precise mathematical notations to eliminate ambiguity (such as Z and B, VDM, LOTOS, Petri Nets, etc.) [15].

This paper describes the works that have been done on analysing several sets of requirements documents in the aim of producing sets of guiding rules and language patterns for the use of better analysis of natural language requirements specifications (NLRSs). From our analysis, we find that certain keywords such as "and", "or", "and/or", "but" and "both" have occasionally contributed to the introduction of ambiguity and defects. Therefore, the rules and language patterns are hoped to aid the writing of better quality requirements with less linguistic inaccuracies and defects.

This paper is organised as follows. Section 2 is a brief literature review of NLRSs. Section 3 describes rules and language patterns for improving the quality of NLRSs. Section 4 briefs on the analysis and rewriting the original requirements process. Finally, Section 5 concludes the content of the report and future work to be done.

2. Related Work

A survey has been conducted on identifying and classifying techniques and approaches that claim to reduce the inherent ambiguities in NLRSs [3]. In general, these approaches can be classified into three categories:

- Approaches that define linguistic rules and analytical keywords [6, 7, 19].
  The approaches present Quality Attributes, Model and Indicators used in evaluating the quality of the existing NLRSs. Frequently used keywords, phrases and sentence structures that cause imprecision are grouped and counted by computer programs. They are thought to be effective in detecting defects and ambiguous NLRSs found in the requirements document.
- Approaches that define guideline-rules [11, 13].
  These approaches summarise rules and guidelines to be adapted in preparing NLRSs. The guidelines avoid incorrect constructions of NLRSs by detecting the potential defects and ambiguities in NLRSs. The definition of rules can be used as a checklist by a requirement engineer to decide the

correctness of the written NLRSs. This would avoid the introduction of natural language ambiguities by restricting the level of freedom in preparing or writing NLRSs.

- Approaches that define specific language patterns to be used in writing the NLRSs for different respective domains [2, 4, 17, 18].

  A pattern language is a devised description of language in a more restricted way. There are several types of patterns such as architectural patterns that show the high level architectures of a software system, design patterns that are focused on the programming aspects, or even patterns for project management [16]. The patterns defined by Ohnishi and by Rolland and Proix [17, 18] concentrate to lessen the NLRSs imprecision in the domain of information system and database. Both Barr and Denger [2, 4] focused on the patterns for embedded system. Denger even devises a metamodel for requirements-statements in the embedded system.

It is worth noting that besides the above three approaches, others discuss the use of quality characteristics that are necessary in writing the well-defined NLRSs [12, 8].

Hooks [12] raises the common problem found in producing the requirements and defined the ways to prevent them. Moreover, she also conducts an in-depth survey on the principal sources of defects in NLRSs and the associated risks. Firesmith [8] summarises a list of good-quality requirements characteristics and also the requirements' problem.

The work from Ambriola and Gervasi [1] concentrates on achieving high-quality of NLRSs through *CIRCE (Cooperative Interactive Requirement-Centric Environment)*. *CIRCE* is based on the concept of successive transformations that are applied to the requirements, in order to obtain concrete (i.e., rendered) views of models extracted from the requirements.

In this work, we identify general language patterns that are sufficient enough to reduce the informality, imprecision and ambiguity of the NLRSs. We focus on language patterns for sentence parts (phrases or clauses), and also for complete-sentence patterns. Based on the studies and analysis on the imprecise and ambiguous requirements statements in the requirements documents [20, 21, 22, 23, 24], we produce a set of language patterns along with their corresponding transformation process in order to reduce the requirements defects. The idea is to transform some ambiguous NLRSs into more simplistic (in terms of less ambiguous) ones. We use rules of inferences to prove the reliability of the transformations. On the other hand, our guiding rules are built up on top of the Denger's authoring rules [2] by extending and adding more guiding rules to be used along with the language patterns. The rules basically describe how to use natural language in writing the requirements whereas the patterns restrict the writing freedom in the purpose of reducing imprecision and ambiguity of NLRSs.

3. Language Patterns and Guiding Rules

This section presents the excerpt of the language patterns and guiding rules that are applicable in general writing of NLRSs in most domains.

3.1 Language Patterns

We examined several case studies and real requirements documents [20, 21, 22, 23, and 24] and extracted NLRS writing schemes. Unlike previous works [2, 4, 17, 18] that concentrate on specific domains, we develop general standardised language patterns that are applicable in most domains and adaptable in the process of writing NLRSs.

In our studies of real requirements documents, we notice the frequent use of "and", "or", "and/or", "but" and "both" has caused the introduction of inherent ambiguity in requirements statements. To facilitate our explanation and due to space limitation, we concentrate on describing a variety of "and" and "or" patterns and a few other general patterns in this paper. The full set of language patterns can be found in the first author's technical report [25].

3.1.1 AND Pattern. The word "and" is generally and always used to represent several combinations of requirements in one requirements statement. From the analysis on the requirements documents, the use of "and" in requirements statements have occasionally made the requirements implicitly ambiguous. For instance:

E.g. An authorised user shall be able to connect to the database server at ease and at will.

There are also requirements statements that use comma "," to list down sets of requirements. Therefore, comma is commonly treated to be similar to "and". The use of "but" in writing the requirements statement may cause different interpretations as well. For instance:

E.g. The LVL1 result will also provide secondary RoIs which did not pass the thresholds, but do pass lower thresholds.

Hence, we suggest to avoid the use of "but". Instead of "but", use "and" since "but" is just another way of saying "and" [25].

Table 1 summarises the sets of "and" and "if and only if" language patterns. The table's left-hand-side column describes the patterns generally adapted in existing requirements documents. The table's middle column illustrates our language patterns transformation. The table's right-hand-side column gives our language patterns.

The following abbreviations are used in this table: GAND = Generic AND Pattern, CACP = Compound AND Condition Pattern, and IFFP= If and Only If Pattern. It is worth noting that CACP3, CACP4, and CACP5 are patterns for Negation AND.

Let: - R be the requirement statement, and $R_1, R_2, \ldots R_n$ are the requirements set.
- S be the reaction implied by R, and $S_1, S_2, \ldots S_n$ are the reactions set.
- C be the condition to R, and $C_1, C_2, \ldots C_n$ are the conditions set.

Table 1. AND and IFF Language Patterns

Patterns found in Requirements Documents	**Suggestion to the Language Patterns Transformation**	**Trivial**
$R_1, R_2, \ldots$ *and* R_n	**GAND**:- R_1 - R_2 - … - R_n	**GAND**:- R_1 - R_2 - … - R_n
C_1 *and* C_2 … *and* C_n *then* S	**CACP1:** C_1 *then* S or C_2 *then* S	$(C_1 \wedge C_2 \rightarrow S) \Leftrightarrow ((C_1 \rightarrow S) \vee (C_2 \rightarrow S))$ $\Leftrightarrow C_1$ *then* S or C_2 *then* S
C *then* S_1 *and* S_2	**CACP2**: C *then* S_1 and C *then* S_2	$(C \rightarrow S_1 \wedge S_2) \Leftrightarrow ((C \rightarrow S_1) \wedge (C \rightarrow S_2))$ $\Leftrightarrow C$ *then* S_1 and C *then* S_2
{*Not* | *Never* | *Neither*} (C_1 {*and* | *nor* } C_2) *then* S	**CACP3**: $\neg C_1$ *then* S and $\neg C_2$ *then* S	$\neg(C_1 \wedge C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$ $\Leftrightarrow \neg C_1$ *then* S and $\neg C_2$ *then* S
{*Not* | *Never* } C_1 *and* C_2 *then* S	**CACP4**: $\neg C_1$ *then* S or C_2 *then* S	$(\neg C_1 \wedge C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \vee (C_2 \rightarrow S)$ $\Leftrightarrow \neg C_1$ *then* S or C_2 *then* S
C_1 *and* C_2 {*Not* | *Never* } *then* S	**CACP5**: C_1 *then* S or $\neg C_2$ *then* S	$(C_1 \wedge \neg C_2) \rightarrow S \Leftrightarrow (C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$ $\Leftrightarrow C_1$ *then* S or $\neg C_2$ *then* S
C *if and only if* S	**IFFP**: C *then* S and $\neg C$ *then* $\neg S$	$C \leftrightarrow S \Leftrightarrow (C \rightarrow S) \wedge (\neg C \rightarrow \neg S)$ $\Leftrightarrow C$ *then* S and $\neg C$ *then* $\neg S$

CACP1 describes several or compound conditions that should occur before the system can trigger a reaction or response in return. **CACP2** describes the occurrence of a specific condition will cause the system triggers several or compound reactions or responses in return. **CACP3** describes the compound of negated conditions in which when they occur, the system will trigger a specific reaction or action in return. **CACP4** and **CACP5** describe the occurrence one particular negated condition will cause the system to trigger a specific reaction in return. **IFFP** describes if the condition is true then the system will trigger a specific reaction in return. If the negated condition occurs, then the system will trigger another reaction to it.

3.1.2 OR Pattern. We observe that the improper uses of "or", "/", or "and/or" in writing requirements statements have also contributed in introducing the inherent ambiguity in NLRSs. They occasionally cause an open and subjective interpretation in realising the requirement. For instance:

E.g. The user shall either be trusted or not trusted.

Besides that, the use of "and/or" should also be avoided in writing requirements statements due to its inherent defects it may cause. For instance:

E.g. An authorised user shall have the ability to edit and/or void a log entry.

Since "and/or" is logically the same with "or" [25], therefore a proper use of "or" is preferable instead of "and/or".

Table 2 summarises the sets of "or" language patterns. The table's left hand side column describes the patterns generally adapted in existing requirements documents. The table's middle column illustrates our language patterns transformation. and the table's right hand column devises our language patterns. The following abbreviations are used in this table: GOR = Generic OR Pattern and COCP = Compound OR Condition Pattern. It is worth noting that COCP5, COCP6, and COCP7 are patterns for Negation OR.

Table 2. OR Language Patterns

Patterns found in Requirements Documents	**Suggestion to the Language Patterns Transformation**	**Trivial**
R_1 *or* R_2 ... *or* R_n	**GOR**: R_1 *or* R_2 *or* ... R_n	R_1 *or* R_2 *or* ... R_n
C_1 *or* C_2 ... *or* C_n *then* S	**COCP1**: - C_1 *then* S - C_2 *then* S - ...	$(C_1 \vee C_2) \rightarrow S \Leftrightarrow ((C_1 \rightarrow S) \wedge (C_2 \rightarrow S))$ $\Leftrightarrow C_1$ ***then*** S **and** C_2 ***then*** S And according to GAND rule, it can be simplified to: - C_1 ***then*** S - C_2 ***then*** S -...
C *then* S_1 *or* S_2	**COCP2**: C *then* S_1 **or** C *then* S_2	$C \rightarrow (S_1 \vee S_2) \Leftrightarrow ((C \rightarrow S_1) \vee (C \rightarrow S_2))$ $\Leftrightarrow C$ ***then*** S_1 **or** C ***then*** S_2
$C \rightarrow S$ { *else* | *otherwise* } $\overline{S}$	**COCP3**: $(C \rightarrow S_1) \vee (\overline{C} \rightarrow \overline{S})$	$(C \rightarrow S_1) \vee (\overline{C} \rightarrow \overline{S})$
$C \rightarrow S_1$ { *else* | *otherwise* } S_2	**COCP4**: $(C \rightarrow S_1) \vee (\overline{C} \rightarrow S_2)$	$(C \rightarrow S_1) \vee (\overline{C} \rightarrow S_2)$
{*Not* | *Never* | *Neither*} (C_1 {*or* | *nor* } C_2) *then* S	**COCP5**: $\neg C_1$ *then* S **or** $\neg C_2$ *then* S	$\neg(C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$ $\Leftrightarrow \neg C_1$ ***then*** S **or** $\neg C_2$ ***then*** S
{*Not* | *Never* } C_1 *or* C_2 *then* S	**COCP6**: $\neg C_1$ *then* S **and** C_2 *then* S	$(\neg C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \wedge (C_2 \rightarrow S)$ $\Leftrightarrow \neg C_1$ ***then*** S **and** C_2 ***then*** S
C_1 *or* C_2 {*Not* | *Never* } *then* S	**COCP7**: C_1 *then* S **and** $\neg C_2$ *then* S	$(C_1 \vee \neg C_2) \rightarrow S \Leftrightarrow (C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$ $\Leftrightarrow C_1$ ***then*** S **and** $\neg C_2$ ***then*** S

COCP1 describes several or compound conditions that should occur before the system can trigger a reaction or response in return. **COCP2** describes the occurrence of a specific condition will cause the system to trigger several or compound reactions or responses in return. **COCP3 and COCP4** describe the occurrence of a specific condition will cause the system to trigger appropriate reaction. However, if the condition is negated, the system must trigger another response or action. **COCP5, COCP6** and **COCP7** describe the combined occurrence of negated conditions will cause the system to trigger specific reaction in return.

3.1.3 Some Other Patterns. Two of the general sentence patterns can be found in Table 3. These patterns should be adapted as guidelines in preparing the NLRSs. GP (Generic Pattern) is meant for writing a simple and affirmative requirements statement whereas GNP (Generic Negated Pattern) is for a negated requirement.

ECP1, ECP2 and ECP3 (Event Condition Pattern) are patterns designed to enable the different ways of representing requirements that are caused by some events and conditions. Denger has also defined sets of event patterns in his work [4, 5]. He clarifies that there is a difference between an event and a condition. An event is a change in the value of a variable in the system state; whereas a condition concerns the value of that variable[5]. Nevertheless, we design the ECP1, ECP2, and ECP3 that are commonly adaptable to writing requirements statement that is caused by either an event or a condition.

TP1, TP2, TP3, and TP4 (Time Pattern) are the patterns to be adapted in writing requirements that concern with time.

We suggest that a clause(s) or a phrase(s) should be taken as guided reference tailored to the requirements. This will eliminate the informality or ambiguity caused by long sentences (due to the occurrence of clauses or phrases) [25].

The following notation conventions are used: sans serif term refers to textual element, a boldface sans serif term refers to the definition of the language pattern, capitalised sans serif term refers to the definition part of speech that should be in place, a lower-case sans serif term refers to the possible role of instantiations of a language pattern element, an italic boldface serif refers to occurrence of text elements in the pattern. The serif term inside curly braces "{ }" and "[ ]" refers to optional pattern element.

Table 3. Some Other Patterns

Generic Patterns	**GP**: NOUN_PHRASE (variable | actor | receiver) {MV | PV} [VERB] (action) COMPLEMENT **GNP**: NOUN_PHRASE (variable | actor | receiver) {MV | PV} ***not*** [VERB] (action) NOUN_PHRASE
Event Condition Patterns	**Condition**: {***Unless*** | ***If*** | ***When***} (conjunction) NOUN_PHRASE (variable | actor | receiver) [MV | PV] VERB (action) [COMPLEMENT] **ECP1**: **Condition, GP** **ECP2**: **GP Condition** **ECP3**: **GNP Condition**
Time Patterns	**TP1**: {***within***} TIME_UNIT **TP2**: [***for***] {***at least*** | ***at most***} [DATA_UNIT] **TP3**: {***as soon as*** | ***as long as …***} ADJECTIVE **TP4**: {***for*** | ***of***} { [***not*** | ***no***] {***more*** | ***less***} ***than*** } TIME_UNIT
Clause Patterns	**Subordinate Clause (Sub_Clause) 1:** {***that*** | ***which***} VERB [COMPLEMENT] **Subordinate Clause (Sub_Clause) 2:** {***but*** | ***as*** | ***since*** | ***while*** | ***where*** } NOUN_PHRASE VERB [COMPLEMENT] **Subordinate Clause (Sub_Clause) 3:** {***in order to*** | ***in [the] case of*** | ***such that*** | ***regardless [of]*** | ***given that*** |…} NOUN_PHRASE VERB [COMPLEMENT]

3.2 Guiding Rules

Our guiding rules are built up on top of the authoring rules [4]. We add more rules to be used together with the language patterns. The majority of the rules are produced based on the analysis on different sets of requirements documents [20, 21, 22, 23, 24], with a few derived from the literature review discussed in Section 2.

The rules are intended to be used along with the language patterns in order to maximise the reduction of ambiguity and possible introduction of imprecision in writing the NLRSs. Therefore, the requirements writer must consider these rules when applying the language patterns.

Due to the space restriction, we present only some of the guiding rules (a more detailed description can be found in [25]) as below:

[Rule 1] Use simple affirmative declarative sentence that consists only 1 main verb [13].

[Rule 2] Avoid writing requirement sentences in passive voice [11].

[Rule 3] Rewrite sentence of the type "There should be X in Y" or "X should exist in Y" into "Y should have X" [13]

[Rule 4] Avoid requirement sentences that contain subjective option in realising the requirement (keywords such as "either", "whether", "otherwise", etc. shall be avoided) [25]

[Rule 5] Avoid the use of keywords such as "eventually", "at last", etc. [25].

[Rule 6] Avoid the use of keywords such as "maximum", "minimum", etc. Alternatively, use "at most", "at least" followed by specific X data or time unit (as defined in TP2) [25].

[Rule 7] To reduce the possibility of arising ambiguity, avoid the use of "both", since "both" is just simply "and" [25].

[Rule 8] Avoid the use of keyword "but". Since "but" is just another way of saying "and", therefore use only "and" instead of "but" [25].

[Rule 9] Avoid the use of "/" in writing the requirements statements. Alternatively, use only "or" instead of "/" [25].

[Rule 10] Avoid the use of "and/or" in writing requirements statements. Alternatively, use only "or" instead of "and/or" because they carry the same logical interpretation [25].

[Rule 11] Avoid the use of unnecessary conjunctions that work as additional commentary to the requirements statements. The following conjunctions shall be avoided such as "not only", "but also", etc. [25].

[Rule 12] Avoid the use of brackets or parentheses "( )", "{ }", "[ ]" due to the ambiguity it may cause. It is generally difficult to interpret whether the parentheses contain optional information or even multiple requirements [25].

[Rule 13] Define a glossary to explain important terms and nominalisations that are used in the requirements statements [11].

[Rule 14] Define an acronym list to explain the used acronyms in the requirements statements [11].

[Rule 15] Define an abbreviation list to explain the used abbreviations in the requirements statements [11].

We realise the aspects as listed in "[Rule 13]", "[Rule 14]", and "[Rule 15]" should be followed so that the requirements writer can differentiate between real requirements and referenced information tailored to the requirements.

4. Validation: Analysis and Rewriting the Original Requirements

We validated the guiding rules and our suggested patterns by rewriting existing requirements documents using the guiding rules and patterns.

Before we rewrote the requirements, the requirements statements from the original requirements documents and case studies were studied to identify the possible defects and imprecision. The original requirements were reviewed by referring to the guiding rules. Whenever the original requirements statement violated a rule, the nature of the violation was then noted. Then, we rewrote the statements by adapting to our suggested language pattern. The following are some examples of the original requirements statements:

R1. When a client makes a one-way send, the server must eventually receive data.

The requirement does not specify the type of data (one-way send or other type of send) and exactly when the server must receive. As long as the server receives the data, then the requirement is fulfilled. Unfortunately, the client won't be aware when the server will be receiving the data. The recommendation to rewrite the requirements statement is:

R1. When a client makes a one-way send, the server must receive the one-way send data.

Or alternatively:

R1. When a client makes a one-way send, the server must receive the sent data.

R2. The system shall return minimum results to the user.

The above example shows a vague and ambiguous requirement caused by keyword "minimum". The system will never be able to decide itself the number of results and the type of results it has to return to the user. Adapting to “[Rule 6]”, GP and GAND patterns, the recommendation to rewrite the requirements statement is:

R2. The system shall return at least 1 search-result to the user.

R3. A reward system must be established not only for the teams of employees, but also for organisations.

Since the conjunctions serve only as additional commentary to the requirements statement, they should be discarded. Adapting to “[Rule 11]”, GP and GAND patterns, the recommendation to rewrite the requirements statement according to the rules and language pattern is:

R3.1. A reward system must be established for the teams of employees.

R3.2. A reward system must be established for the organisations.

R4 Neither if the system receives the requested data nor the one-way sent data within 24 hours, then the system must automatically alert the user.

We rewrote the statement to be more precise by adapting to COCP5 pattern as shown below:

R4.1 If the system doesn’t receive the requested data within 24 hours, then the system must automatically prompt an alert message to the user

R4.2 If the system doesn’t receive the one-way data within 24 hours, then the system must automatically prompt an alert message to the user

Due to the space limitation, we can only describe the validation and rewritten process by using 4 examples of ambiguous requirements found on the existing requirements documents and case studies [20, 21, 22, 23, 24]. More examples and rewritten requirements can be found in the first author’s Technical Report [25].

5. Conclusion and Future Direction

We address the current research work in defining guiding rules and our suggested language patterns to be used in writing the NLRSs. The introduced rules and language patterns are developed from the studies of several sets of requirements documents and series of literature reviews and NLRS state of practice.

We validate the usefulness and applicability of the guiding rules and suggested language patterns by rewriting the ambiguous requirements sentences applying both the language patterns and rules.

This research work will further continue on developing a system coined SREE (Systemised Requirements Engineering Environment). SREE that incorporates all our suggested language patterns, is mainly designated as an environment for the analysis of natural language requirements. SREE is expected as a work companion for the requirements engineer or software developers that reads NLRSs as inputs and in anticipation, produces views on different aspects of requirements (such as requirements specification to be presented in diagrammatic ways). One may also think of SREE as Requirements Management tool.

Overview of the transformation of Higher Quality NLRSs to be incorporated in SREE:

- First, the requirements document is analysed, sorted and rewritten into a set of structured requirements sentences by applying the guiding rules and language patterns
- Next, the produced structured and unambiguous requirements will be parsed and tagged by an automated tool.
- The parsed attributes of the requirements will be represented in diagrammatic notations as modeling aid.

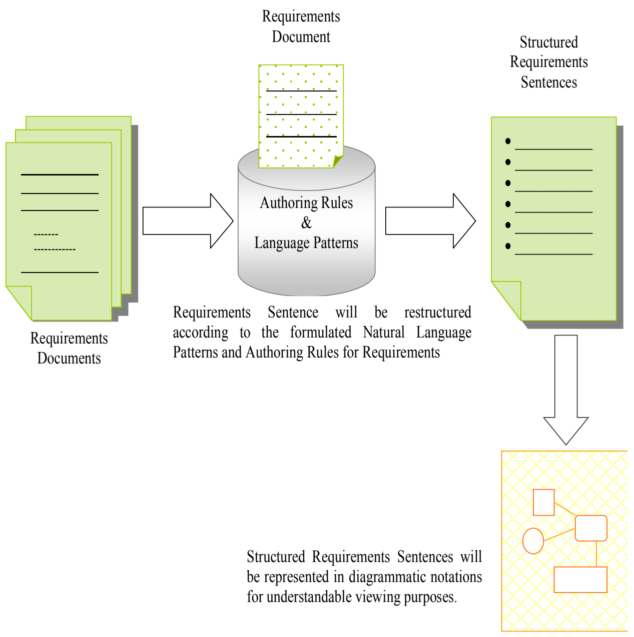
Requirements Document
Structured Requirements Sentences
Authoring Rules & Language Patterns
Requirements Documents
Requirements Sentence will be restructured according to the formulated Natural Language Patterns and Authoring Rules for Requirements
Structured Requirements Sentences will be represented in diagrammatic notations for understandable viewing purposes.

Figure 1. Improving the qualtiy of NLRSs through natural language requirements patterns

SREE is also expected to be highly adaptable to different applications domains and requirements. It will later be tested in a new product development environment so that the effects on the process can be monitored. We expect that the combination use of the tool and requirements engineer as the human inspectors will achieve the best maximum result of engineering the software requirements.

Acknowledgement

The authors thank D. M. Berry for his invaluable and significant advises, comments and reviews.

10. References

[1] Ambriola, V. and Gervasi, V., *"The CIRCE approach to the systematic analysis of NL requirements",* Technical Report: TR-03-05, Università Di Pisa, 2003
[2] Barr V., *"Identifikation von Spezifikationsmustern im Echtzeitentwurf anhand der Fallstudie Antiblockiersystem"*, Diplomarbeit der Universitaet Oldenburg, Fachbereich Informatik, 1999
[3] Denger C., Jőrg D., Kamsties E., *"QUASAR A Survey on Approaches for Writing Precise Natural Language Requirements"*, Fraunhofer IESE, 2001
[4] Denger C., *"High Quality Requirements Specifications for Embedded Systems through Authoring Rules and Language Patterns", M.Sc. Thesis*, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 2002
[5] Denger C., Berry D.M., Kamsties E., *"Higher Quality Requirements Specifications through Natural Language Patterns"*, Proceedings of the IEEE International Conference on Software – Science, Technology & Engineering (SwSTE'03), 2003
[6] Fabbrini F., Fusani M., Gnesi G. and Lami G., *"Quality Evolution of Software Requirements Specifications"*, Proceedings Software and Internet Quality Week 2000 Conference, San Fransisco, CA, 2000
[7] Fabbrini F., Fusani M., Gnesi G. and Lami G., *"The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool"*, Proceedings of the 26th Annual NaSA Goddard Software Engineering Workshop (SEW'01), 2002
[8] Firesmith D., *"Specifying Good Requirements"*, Journal of Object Technology 2, 2003
[9] Fliedl G., Kop C., Mayerthaler W., Mayr H.C., Winkler C., *"Linguistic Aspects of Dynamics in Requirements Specifications & Linguistically Based Requirements Engineering"*, Data & Knowledge Engineering, Netherlands, 2000
[10]Fuchs N.E. and Schwitter R., *"Controlled English for Requirements Specification"*, IEEE Computer Special Issue on Interactive Natural Language Processing, 1996
[11] Götz R. and Rupp C., *"Regelwerk Natürlichsprachliche Methode"*, Sophist, Nürnberg, Germany, 1999
[12] Hooks I., *"Writing Good Requirements"*, Vol. 2, pp. 197-203, Proceedings of the Fourth International Symposium of the NCOSE 2, San Jose, CA, 1994
[13] Juristo N., Moreno A.M. and Lopez M., *"How to Use Linguistic Instruments for OOA"*, pp. 80-89 Proceedings of the IEEE International Conference on Software, 2000
[14] Lami Giuseppe., *"QuARS: A Tool for Analysing Requirements"*, Carnegie Mellon University, September 2005
[15] Lanman J. T., *"Using Formal Methods in Requirements Engineering Version 1.0"*, Department of Computing and Mathematics, Embry-Riddle Aeronautical University, 2002
[16] Martinez A., Pastor O., Estrada H., *"A pattern language to join early and late requirements"*, pp 51-64 Anais do WER04 - Workshop em Engenharia de Requisitos, Tandil, Argentina, 2004
[17] Ohnishi A., "*Customizable Software Requirements Languages"*, Proceedings of the Eighth International Computer Software and Application Conference (COMPSAC), IEEE Computer Society, Los Alamitos, CA, 1994
[18] Rolland C. and Proix C., "*A Natural Language Approach for Requirements Engineering"*, pp. 257-277 in Proceedings of Conference on Advanced Informations Systems Engineering, CAiSE, Manchester UK, 1992
[19] Wilson W.M., Rosenberg L.H. and Hyatt L.E., *"Automated Quality Analysis of Natural Language Requirements Specifications"*, NASA Software Assurance Technology Center, The Software Assurance Technology Center (SATC), NASA Goddard Space Flight Center (GSFC), Greenbelt, MD, 1996
[20] Nelbach F.J., *"Software Requirements Document For the Data Cycle System (DCS) Of The SOFIA Project"*, Universities Space Research Association, 2002 http://www.astro.ucla.edu/~shuping/SOFIA/Documents/DCS_SRD_Rev1.pdf
[21] Moeser R., Perley P., *"EVLA Operations Interface, Software Requirements"*, EVLA-SW-003 Revision: 2.5, 2003 http://www.aoc.nrao.edu/evla/techdocs/computer/workdocs/array-sw-rqmts.pdf
[22] Dubois R., "*Large Area Telescope (LAT) Science Analysis Software Specification"*, GE-0000X-DO, 2000 http://www-glast.slac.stanford.edu/IntegrationTest/DataHandling/docs/LAT-SS-00020-06.pdf
[23] George S., "*PESA High-Level Trigger Selection Software Requirements"*, 2001 http://www.pp.rhul.ac.uk/atlas/newsw/requirements/1.0.2/
[24] Bray, I.K., "*An Introduction To Requirements Engineering"*, 2002, © Pearson Education Limited
[25] Tjong, S.F., "*Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns*", Technical Report, University of Nottingham Malaysia Campus, March 2006