NATURAL LANGUAGE PATTERNS FOR NATURAL LANGUAGE REQUIREMENTS SPECIFICATIONS

Sri Fatimah Tjong, BSc.

Technical Report submitted to the University of Nottingham

for the degree of Doctor of Philosophy

Aug 2006

Abstract

Studies have been continuously conducted to find effective approaches and techniques to better analyse Natural Language Requirements Specifications (NLRSs). NLRs are widely used in software development and they are highly prone to ambiguity and imprecision. We recognise the need of defining an approach that will solve the NLRS inherent problem in most domains.

This report presents an approach for reducing the problem of ambiguity and imprecision in NLRSs with the use of quality language patterns and guideline rules. To ensure the applicability of our approach, we studied different sets of requirements documents from several domains. We further validate our approach by rewriting the requirements statements derived from the requirements documents.

Table of Contents

List of Tables

List of Figures

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 resulted incorrectness, inconsistency, incompleteness, or 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 [Denger et. al., 2001]. 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.) [Lanman, 2002].

This report describes the works that have been done on analysing several sets of Requirements Document in the aim of producing sets of guiding rules and language patterns for the use of better analysis of natural language requirements specification (NLRSs). From our analysis, we find certain keywords such as “and”, “or”, “and/or”, “but”, “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.

The presentation of the report is organised as follows. Chapter 2 outlines a brief literature review of NLRSs. Chapter 3 describes guiding rules and language patterns for improving the quality and the recommendation in rewriting the original requirements process. Chapter 4 briefs on the continuing present and future work, and also the ultimate goal of the research work. Chapter 5 concludes the content of the report. Finally, chapter 6 encloses the list of supporting references used in the report.

2. Literature Review and Natural Language Requirements State of Practice

A survey has been conducted on identifying and classifying techniques and approaches that claim to reduce the inherent ambiguities in NLRSs [Denger et. al., 2001]. In general, these approaches can be classified into three categories:

2.1 Approaches that define linguistic rules and analytical keywords [Fabbrini et. al., 2000; Fabbrini et. al., 2002; Wilson et. al., 1996]

Wilson, Rosenberg and Hyatt [Wilson et. al., 1996] define the overall quality aspects of requirements specifications and requirements in general. Two quality aspects are distinguished as listed below:

- Quality Attributes that define aspects such as completeness, correctness, traceability, non-ambiguity, etc.
- Quality Indicators that associate to the quality of requirements document and individual specification statements such as imperatives, continuances, directives, options and weak phrases (refers to Table 1. more detailed classification)

These aspects are then implemented in an analysis tool called ARM (Automated Requirements Measurement), developed by the Software Assurance Technology Centre (SATC).

Fabbrini et. al. [Fabbrini et. al., 2000] distinguish the aspects between requirements sentences quality (RSQ) and requirements documents quality

(RDQ). Hence, the following is the list of Quality Indicators for both RSQ and RDQ (refers to Table 2. and 3. for detailed classification):

- RSQ related Indicators include implicit subject sentences, multiple sentence, optional sentences, subjective sentences, underspecified sentences, vague sentences and weak sentences.
- RDQ related Indicators include comment frequency, readability index, under-referenced sentences and unexplained sentences

From their findings, they develop an automatic tool called QuARS (Quality Analyser of Requirements Specifications) that will support the analysis and quality evaluation of requirements specifications.

To conclude, the above 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.

2.2 Approaches that define guideline-rules [Götz and Rupp, 1999; Juristo et. al., 2000]

Götz and Rupp develop a rule base that contains all rules needed to detect defects, ambiguities and weak phrases in requirements specifications. They distinguish three main transformation process used to model the original intention of a person (in communicating the requirements) which namely are:

- Deletion that reduces the perception of a person to a scope he or she can deal with

- Generalisation that leads to a detachment of an experience from its context and to assume that the experience is overall valid
- Distortion that is related to Nominalisation i.e. a noun stands for a complex process. Examples of nominalisation include: the recording, the playback, the take off, etc.

These processes form the base to detect defects in requirements specifications. For each process, there are rules developed to solve the problems of the related defects. The rules are believed to be efficient means for validating and specifying natural language requirements.

Juristo et. al. [Juristo et. al., 2000] classify requirements into static and dynamic requirements. Static requirements are then reformatted into the structure of the static utility language (SUL) and Dynamic requirements are reformatted into the structure of the dynamic utility language (DUL). Both SUL and DUL are composed of several natural language structures, which are formally described by grammars. Juristo et. al. also define distinct guidelines for static and dynamic requirements to be adapted in reformatting the requirements specifications.

In short, 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.

2.3 Approaches that define specific language patterns to be used in writing the NLRSs for different respective domains [Barr, 1999; Denger, 2002; Ohnishi, 1994; Rolland and Proix, 1992].

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 focus on the programming aspects, and even patterns for project management [Martinez et. al., 2004].

Ohnishi [Ohnishi, 1994] develop an X-JRDL analyser (for Japanese language) which is based on a concept called requirements frame model for the file system domain. The requirements frame model distinguishes between three different frames, namely Noun Frame, Case Frame, and Function Frame. Each of these frames restricts the vocabulary and the context of the requirements specifications. Ohnishi also states that with the requirements frame model, each requirements sentence can be transformed into an internal representation called CRD (Conceptual Requirements Description). Then each requirement description can be automatically analysed by the X-JRDL analyser.

The patterns defined by Rolland and Proix [Rolland and Proix, 1992] are derived from the Fillmore's case system [Fillmore, 1968] that tailored to the needs of the database development domain. During this adaptation of the process, the case system is extended in two directions. Firstly, the cases can be related not only to words, but also to clauses of sentences. Secondly, the classification of the case has also been modified. Furthermore, Rolland et. al. categorise several classes of verbs and distinguish two main linguistic

patterns. The linguistic patterns are a set of patterns that combine cases and classes of verbs. The patterns namely are:

- elementary patterns that allow associating cases to syntactic units of a clause
- sentence patterns that allow associating cases to clauses of a sentence

The ideas of this approach are then implemented in a tool called OICSI (which is based on the French natural language), to automate the support of the requirements engineer.

Both Barr and Denger [Barr, 1999; Denger, 2002] focus on the language patterns for embedded system domain. Barr identifies specification patterns (sentence patterns), which shall support the transformation of unstructured natural language requirements into a formal specification language. In his work, he distinguishes two different classes of patterns, which are:

- the if-then patterns described within the Rule-Scheme
- the patterns expressing an overall valid fact described within schemes for consequences without conditions

The difference between these patterns is that the consequence part of a rule is realised if and only if the condition evaluates to true.

Denger [Denger, 2002; Denger, 2003] develops an approach for reducing the problem of imprecision in NLRSs with the use of natural language patterns, authoring rules and document templates. He outlines several distinct patterns such as Sentence Patterns, Event Patterns, Reaction Patterns, Computation Pattern, etc. He even devises a metamodel for functional requirements-statements in the embedded system.

There is also work in specifying a controlled language for writing the requirements in an almost natural language. Fuch and Schwitter [Fuchs and Schwitter, 1996] define a Controlled English, which is a subset of natural language with restricted syntax and semantics of full natural language and a domain-specific vocabulary. It allows domain specialists to interactively formulate requirements specifications in domain concept. Fuch and Schwitter have developed a system, called Attempto that translates complete specification texts in Controlled English into discourse representation structures (a structured form of first-order predicate logic and optionally into Prolog).

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 such as the work done by [Firesmith, 2003; Hooks, 1994].

Hooks [Hooks, 1994] 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 [Firesmith, 2003] summarises a list of good-quality requirements characteristics and also the requirements' problem.

The work from Ambriola and Gervasi [Ambriola and Gervasi, 2003] 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.

Our work identifies 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 [DCS, 2002; EVLA, 2003; LAT, 2000; PESA, 2001; Bray, 2002], 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 guideline rules are built up on top of the Denger's authoring rules [Barr, 1999] by extending and adding more guideline 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. Guiding Rules and Language Patterns

This chapter includes the guiding rules and language patterns that are adaptable in most domains. Section 3.1 documents the guiding rules that requirements engineer or software developer shall adhere to in authoring the requirements statements. Section 3.2 presents the general language patterns along with associated examples of requirements statements. We validate our guiding rules and suggested language patterns by directly rewriting on the requirements. Before we rewrote the requirements statements, we reviewed the ambiguous requirements and requirements with defects. Whenever the original requirements statements violated a rule, the nature of violation was noted. Then, we rewrote the statements by adapting to our suggested language patterns.

3.1 Guiding Rules

Our guiding rules are built up on top of the authoring rules [Denger, 2002]. 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 [DCS, 2002; EVLA, 2003; LAT, 2000; PESA, 2001; Bray, 2002], with a few derived from the literature review discussed in Chapter 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. Following is the list of guiding rules:

[Rule 1]

Use simple affirmative declarative sentence that consists only 1 main verb [Juristo, et. al, 2000].

(E1) The system shall store 20 GB of processed data per day.

[Rule 2]

Avoid writing requirement sentences in passive form.

[Rule 3]

Rewrite sentence of the type "There should be X in Y" or "X should exist in Y" into "Y should have X" [Juristo, et. al., 2000].

[Rule 4]

Avoid requirement sentences that contain subjective option in realising the requirement (keywords "either", "whether", "otherwise"…).

(E2) The user shall either be trusted or not trusted [EVLA].

(E3) The system shall inform the user whether the new version is required or recommended [EVLA].

[Rule 5]

Avoid the use of "eventually", "at last", … in order to eliminate any possible disambiguation arisen.

(E4) When a client makes a one-way send, the server must eventually receive data.

Recommendation:

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

[Rule 6]

To eliminate the disambiguation caused by "maximum" and "minimum", Replace "maximum" with "at most" and "minimum" with "at least" followed by X data or time unit.

(E5) The system shall return minimum results to the user.

Recommendation:

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

[Rule 7]

Avoid the use of "/" in writing the requirement sentence. Alternatively, substitute the use of "/" with "or".

[Rule 8]

Avoid the use of "and/or" in writing requirement sentences. Alternatively, substitute the use of "and/or" with "or" because they carry the same logical interpretation (as proven in Table 4.).

(E6) An authorised user shall have the ability to edit and/or void a log entry.

Recommendation:

- An authorised user shall have the ability to edit or void a log entry.

[Rule 9]

Since "but" is just another way of saying "and", therefore substitute "but" with "and".

(E7) The LVL1 result will also provide secondary RoIs which did not pass the thresholds, but do pass lower thresholds [PESA].

Recommendation:

- The LVL1 result will also provide secondary RoIs which did not pass thresholds, and do pass lower thresholds [PESA].

[Rule 10]

Avoid the use of "both", since "both" is just simply "and", therefore discards "both".

(E8) The system should print reports for both users and clients.

Recommendation:

- The system should generate reports for users.
- The system should generate reports for clients.

[Rule 11]

Avoid the use of unnecessary conjunctions that work as additional commentary to the requirement sentence. The following conjunctions shall be avoided such as "not only", "but also" …

(E9) A reward system must be established not only for the individuals, but also for organisations and teams of employees.

Recommendation:

- A reward system must be established for the individuals.
- A reward system must be established for organisations.
- A reward system must be established for teams of employees.

[Rule 12]

Simplify requirement sentence that has more than 1-time occurrence of "and", "from", "for", ",".

(E10) All monitor points from weather-related equipment, for the array and for individual antennas, shall be available to the user [EVLA].

There are 2 recommendation of rewriting the above requirement and both of them carry different meaning:

Recommendation 1:

- All monitor points from weather-related equipment shall be available to the user.
- All monitor points for the array shall be available to the user.
- All monitor points for individual antennas shall be available to the user.

Recommendation 2:

- All monitor points from weather-related equipment for the array shall be available.
- All monitor points from weather-related equipment for individual antennas shall be available to the user.

[Rule 13]

Avoid the use of brackets or parentheses "( )" due to the ambiguity it carries. It is difficult to interpret whether the parentheses contain optional information or even multiple requirements. Requirement that comes with bracketed information will be taken as guided referenced information to the requirement.

(E11) The lift should never be allowed to move above the top floor or below the bottom floor. (There is an emergency shut down system that will stop the motor if the lift goes above the top floor or below the bottom floor by more

than 10 cm but this shut down system is beyond the scope of the lift control system.) [Bray, 2002].

Recommendation:

- The lift should never be allowed to move above the top floor or below the bottom floor.

[Rule 14]

Define a glossary to explain important terms and nominalisations that are used in the requirement. (refer to [Götz and Rupp, 1999]).

[Rule 15]

Define an acronym list to explain the used acronyms in the requirement. (refer to [Götz and Rupp, 1999]).

[Rule 16]

Define an abbreviation list to explain the used abbreviations in the requirement. (refer to [Götz and Rupp, 1999]).

[Rule 17]

Dependent requirement (requirement with mother and child relationship) should be group together.

(E12) The SDP shall provide the Level 1 data to the P1 sites in a manner of TBR. The Level 1 data should arrive at the sites no later than 24 hours (TBR) after completion of processing in the SDP. Then, the SDP may (TBR) provide the Level 0 data to the P1 sites.

[Rule 18]

To avoid the ambiguity of 'all' and 'any' [Berry and Kamsties, 2000; Berry et. al., 2003], rewrite the word 'all' and 'any' into 'each' or 'every'. We believe 'each', 'every', 'any' and 'all' should be considered as Universal Quantifiers.

(E13) The operator log will record *all* warning messages prompted by the system.

$\forall$ x record (x), where x is a warning message.

(E14) The operator log will record *any* warning messages prompted by the system.

$\forall$ x record (x), where x is a warning message.

(E15) The operator log will record *every* warning message prompted by the system.

$\forall$ x record (x), where x is a warning message.

Another example:

(E16) All output messages shall be categorised and reported via a common mechanism.

Recommendation to rewrite the requirement according to the rule and GAND pattern is:

- Each output message shall be categorized via a common mechanism.
- Each output message shall be reported via a common mechanism.

Further example:

(E17) The system should be possible to disable the monitoring system or at least any parts of it, which are resource-intensive.

Recommendation:

- The system should be possible to disable the monitoring system or at least every part of the system, which is resource-intensive.

[Rule 19]

Based on our studies on propositional logic and predicate logic [3, 9], we would classify "some", "many", "few", "a", "an", "the" as existential quantifiers ($\exists x$). Hence, one should rewrite the requirements statements that contain "some", "many", "few" into a specific measurable data unit.

[Rule 20]

Keywords such as 'meanwhile', 'whereas', 'on the other hand',… etc. are generally used to combine 2 or more related requirements. However, these keywords should be avoided as they tend to complicate the requirements.

(E18) Each officer can print the report by selecting an Associate *meanwhile* an Associate can only view the report which containing the payment details entered by the associate himself [Eng, 2005].

Recommendation:

- Each officer can print the report by selecting an Associate.
- An associate can only view the report which containing the payment details entered by the associate himself.

[Rule 21]

Avoid the use of vague adjectives such as "prompt", "fast", "routine",… in timing process(es) that happened/described in the requirements.

(E19) The Science Analysis Software performs *prompt* processing of Level 0 data to produce Level 1 event data.

Recommendation

- The Science Analysis Software performs 0.1 second processing of Level 0 data to produce Level 1 event data

[Rule 22]

When "between" or "among" is used in differentiating one action/process with another action/process described in the requirement, then the requirement should not be simplified by any of the language patterns.

(E20) Relationships between objects made at LVL2 must still be valid if they are passed on the EF or stored and retrieved in the offline environment.

[Rule 23]

Avoid vague adjectives such as "arbitrary", "ancillary", "relevant",… etc. as they are open for uncertain subjective decision in realising the requirement.

(E21) In support of high-level processing, the SAS extracts from the LAT and SC Level 0 data ancillary information relevant to event reconstruction and classification.

3.2 Language Patterns

We examined several case studies and real requirements documents [DCS, 2002; EVLA, 2003; LAT, 2000; PESA, 2001; Eng, 2005; Bray, 2002] and extracted NLRS writing schemes. Unlike previous works [Barr, 1999; Denger, 2002; Ohnishi, 1994; Rolland and Proix, 1992] 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.

An alphabetised list of definitions for special terms used in this report is listed as follows:

- COMPLEMENT- Noun, Noun_Phrase, Adverb, Adjective
- Modal Auxiliary Verb (MV)- can, may, shall, must, will, should
- Primary Auxiliary Verb (PV)- is, are, was, were
- Pattern- the unstructured model of language pattern generally used in writing the NLRs (as studied from the requirements documents)

The following notation convention is used in this report:

- A verdana term refers to textual element
- A bolded verdana term refers to the definition of the language pattern
- A capitalised verdana term refer to definition part of speech that should be in place
- A lower-case verdana term refers to the possible role of instantiations of a language pattern element
- A bolded Times-New-Roman Italic refers to occurrence of text elements in the pattern

- The Times-New-Roman term inside curly braces "{ }" and "[ ]" refers to optional pattern element

Let:
- R be the requirement statement, and R_1 **,** R_2 **,...** R_n are the requirements set.
- S be the reaction implied by R , and S_1 **,** S_2 **,...** S_n are the reactions set. (Each reaction should be written by adopting the **GP** pattern)
- C be the condition to R , and C_1 **,** C_2 **,...** C_n are the conditions set.

- **Generic Pattern (GP)**

  Generic Pattern is the pattern designed for writing simple and affirmative requirements statements.

GP:	NOUN_PHRASE (variable | actor | receiver) {MV | PV} [VERB] (action) COMPLEMENT

(E22) The user can enter details of: boat-class, boat, race, series, race-entry, series-entry.

Rewrite requirement according to the pattern:

- The user can enter details of boat-class.
- The user can enter details of boat.
- …
- The user can enter details of series-entry.

- **Generic Negation Pattern (GNP)**

  Generic Negation Pattern is the pattern designed for writing negated requirements statements.

GNP: NOUN_PHRASE (variable | actor | receiver) {MV | PV} ***not*** [VERB] (action) NOUN_PHRASE

(E23) The system should not remove the messages from POP server until the messages are retrieved successfully.

- **Event Condition Pattern (ECP)**

  The ECPs are designed to enable different ways of representing requirements that are caused by some events and conditions. Denger has also defined sets of event patterns in his work [Denger, 2002; Denger et. al., 2003]. 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 [Denger et. al., 2003]. 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.

Condition: {***Unless*** | ***If*** | ***When***} (conjunction) NOUN_PHRASE (variable | actor | receiver) [MV | PV] VERB (action) [COMPLEMENT]

ECP1: Condition, **GP**

(E24) If the boat's details are amended, the boat's details will not affect the outcome of any races.

ECP2: GP Condition

(E25) A boat may only be entered into a handicap race (series) if the boat's handicap type matches that of the race.

ECP3: GNP Condition

(E26) A boat's information can not be entered into the system unless the boat has a boat class.

- **"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. There also requirements statements which use comma "," to list down sets of requirements. Therefore, comma is commonly treated to be similar to "and".

Generic 'AND' Pattern (GAND)

Pattern: R_1, R_2 ,... and R_n

Theorem: $R_1 \wedge R_2 \wedge ... R_n$ can be simplified into:

GAND:	- R_1
	- R_2
	- …
	- R_n

(E27) The system shall have the ability to create, add, and delete a new account.

Recommendation:

- The system shall have the ability to create a new account.
- The system shall have the ability to add a new account.
- The system shall have the ability to delete a new account.

Compound AND Condition Pattern (CACP)

Pattern: $C_1 \ and \ C_2 \ ... \ and \ C_n$

$then \ S$

Theorem: $C_1 \wedge C_2 \wedge ... \wedge C_n$

$then \ S$, can be simplified into:

CACP1: $(C_1 \wedge C_2 \rightarrow S) \Leftrightarrow ((C_1 \rightarrow S) \vee (C_2 \rightarrow S))$

$\Leftrightarrow C_1$ ***then*** S **or** C_2 ***then*** S

(refer to Table 5. for a logical proof)

CACP1 describes several or compound conditions that should occur before the system can trigger a reaction or response in return.

(E28) When the message has been created and the user finished editing the message, then the message will be placed in the Outbox.

Recommendation to rewrite (E28) according to **CACP1**:

- When the message has been created, then the message will be placed in the Outbox or When the user finished editing the message, then the message will be placed in the Outbox.

Pattern: C

$then\ S_1\ and\ S_2$

Theorem: C

$then\ S_1\ \wedge S_2$, can be simplified into:

CACP2: $(C \rightarrow S_1 \wedge S_2) \Leftrightarrow ((C \rightarrow S_1) \wedge (C \rightarrow S_2))$

$\Leftrightarrow C\ \textbf{\textit{then}}\ S_1\ \textbf{and}\ C\ \textbf{\textit{then}}\ S_2$

(refer to Table 6. for a logical proof)

CACP2 describes the occurrence of a specific condition will cause the system triggers several or compound reactions or responses in return.

(E29) If the system's connection to the server is not available then the system will report an error and reconnect to the server.

Recommendation to rewrite (E29) according to **CACP2**:

- If the system's connection to the server is not available then the system will report an error.
- If the system's connection to the server is not available then the system will reconnect to the server

Pattern: $\{\textbf{\textit{Not}} \mid \textbf{\textit{Never}} \mid \textbf{\textit{Neither}}\}\ (C_1\ \{\textbf{\textit{and}} \mid \textbf{\textit{nor}}\}\ C_2)\ \textbf{\textit{then}}\ S$

Theorem Proof: $\neg(C_1 \wedge C_2) \rightarrow S \Leftrightarrow (\neg C_1 \vee \neg C_2) \rightarrow S$

Let $X = \neg C_1, Y = \neg C_2$ $\Leftrightarrow (X \vee Y) \rightarrow S$

(Derived from **C0CP1**) $\Leftrightarrow (X \rightarrow S) \wedge (Y \rightarrow S)$

$\Leftrightarrow (\neg C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$

CACP3: $\neg(C_1 \wedge C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$

$\Leftrightarrow \neg C_1$ ***then*** S **and** $\neg C_2$ ***then*** S

(refer to Table 7. for a logical proof)

CACP3 describes the compound of negated conditions in which when they occur, the system will trigger a specific reaction or action in return.

(E30) If the user has neither an unidentified nor unauthorised account, the system shall never grant an access to the database.

Recommendation to rewrite (E30) according to **CACP3**:

- If the user does not have an unidentified account, the system shall never grant an access to the database.
- If the user does not have an unauthorised account, the system shall never grant an access to the database.

Pattern: $\{$***Not*** $|$ ***Never*** $\}$ C_1 ***and*** C_2 ***then*** S

or

C_1 ***and*** C_2 $\{$***Not*** $|$ ***Never*** $\}$ ***then*** S

Theorem Proof**:** $(\neg C_1 \wedge C_2) \rightarrow S \Leftrightarrow (C_1 \vee \neg C_2 \wedge S)$

$\Leftrightarrow (C_1 \wedge S) \vee (\neg C_2 \wedge S)$

$\Leftrightarrow (\neg C_1 \rightarrow S) \vee (C_2 \rightarrow S)$

and so does

$$(C_1 \wedge \neg C_2) \rightarrow S \Leftrightarrow (C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$$

CACP4: $(\neg C_1 \wedge C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \vee (C_2 \rightarrow S)$

$\Leftrightarrow \neg C_1$ ***then*** S **or** C_2 ***then*** S

(refer to Table 8. for a logical proof)

CACP5: $(C_1 \wedge \neg C_2) \rightarrow S \Leftrightarrow (C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$

$\Leftrightarrow C_1$ ***then*** S **or** $\neg C_2$ ***then*** S

(refer to Table 9. for a logical proof)

CACP4 and **CACP5** describe the occurrence one particular negated condition will cause the system to trigger a specific reaction in return.

(E31) If the connection is not made to the server but is resetting network component to initial state, then the system shall log an error report.

Recommendation to rewrite (E31) according to **CACP4**:

- If the connection is not made to the server, the system shall log an error report or if the connection is resetting network component to initial state, then the system shall log an error report.

Elaborated AND Patterns (ECAND)

Pattern: $(C_1$ ***and*** C_2 ***and*** ... $C_n)$ ***then*** S

Theorem Proof: $(C_1 \wedge C_2 \wedge ...C_n) \rightarrow S$

$\Leftrightarrow \neg(C_1 \wedge C_2 \wedge ...C_n) \vee S$

$\Leftrightarrow (\neg C_1 \vee \neg C_2 \vee ....\neg C_n) \vee S$

$\Leftrightarrow (\neg C_1 \vee S) \vee (\neg C_2 \vee S) \vee ...(\neg C_n \vee S)$

$\Leftrightarrow (C_1 \rightarrow S) \vee (C_2 \rightarrow S) \vee ...(C_n \rightarrow S)$

ECAND1: $(C_1 \wedge C_2 \wedge ...C_n) \rightarrow S$

$\Leftrightarrow (C_1 \rightarrow S) \vee (C_2 \rightarrow S) \vee ...(C_n \rightarrow S)$

$\Leftrightarrow$ (C_1 ***then*** S) ***or*** (C_2 ***then*** S) ***or*** ...(C_n ***then*** S)

ECAND1 describes a set of conditions that are combined with conjunction "and" that must occur before the system can trigger a reaction or response in return.

(E32) If the generic, username, and password that are keyed in the login window are not validated, then the application shall not be initialised.

We refine (E32) according to our ECAND1 pattern:

- If the generic that is keyed in the login window is not validated, then the application shall not be initialised.
- If the username that is keyed in the login window is not validated, then the application shall not be initialised.
- If the password that is keyed in the login window is not validated, then the application shall not be initialised.

Pattern: C ***then*** (S_1 ***and*** S_2 ***and*** ... S_n)

Theorem Proof: $C \rightarrow (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow \neg C \vee (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow (\neg C \vee S_1) \wedge (\neg C \vee S_2) \wedge ...(\neg C \vee S_n)$

$\Leftrightarrow (C \rightarrow S_1) \wedge (C \rightarrow S_2) \wedge ...(C \rightarrow S_n)$

ECAND2: $C \rightarrow (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow (C \rightarrow S_1) \wedge (C \rightarrow S_2) \wedge ...(C \rightarrow S_n)$

$\Leftrightarrow$ (C ***then*** S_1) ***and*** (C ***then*** S_2) ***and ...*** (C ***then*** S_n)

ECAND2 describes the occurrence of one particular condition that causes the system to trigger a set of reactions, which are combined with conjunction "and" in return.

(E33) If the user logs in as a Project Manager, the Add, Update, and Delete buttons are enabled in the Requirements windows.

We refine (E33) according to ECAND2 pattern:

- If the user logs in as a Project Manager, then the Add button is enabled in the Requirements windows.
- If the user logs in as a Project Manager, then the Update button is enabled in the Requirements windows.
- If the user logs in as a Project Manager, then the Delete button is enabled in the Requirements windows.

Pattern: (C_1 ***and*** C_2 ***and ...*** C_n) ***then*** (S_1 ***and*** S_2 ***and ...*** S_n)

Theorem Proof: $(C_1 \wedge C_2 \wedge ...C_n) \rightarrow (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow \neg(C_1 \wedge C_2 \wedge ...C_n) \vee (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow (\neg C_1 \vee \neg C_2 \vee ....\neg C_n) \vee (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow ((\neg C_1 \vee \neg C_2 \vee ...\neg C_n) \vee S_1) \wedge ((\neg C_1 \vee \neg C_2 \vee ...\neg C_n) \vee S_2) \wedge ....$

$((\neg C_1 \vee \neg C_2 \vee ...\neg C_n) \vee S_n)$

$\Leftrightarrow (\neg(C_1 \wedge C_2 \wedge ...C_n) \vee S_1) \wedge (\neg(C_1 \wedge C_2 \wedge ...C_n) \vee S_2) \wedge ...$
$(\neg(C_1 \wedge C_2 \wedge ...C_n) \vee S_n)$

$\Leftrightarrow \left(\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow S_1\right) \wedge \left(\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow S_2\right) \wedge ...$
$\left(\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow S_n\right)$

ECAND3: $\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow \left(S_1 \wedge S_2 \wedge ... S_n\right)$

$\Leftrightarrow \left(\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow S_1\right) \wedge \left(\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow S_2\right) \wedge ...$
$\left(\left(C_1 \wedge C_2 \wedge ... C_n\right) \rightarrow S_n\right)$

$\Leftrightarrow$ (C_1 ***and*** C_2 ***and ...*** C_n) ***then*** S_1 ***and***

(C_1 ***and*** C_2 ***and ...*** C_n) ***then*** S_2 ***and ...***

(C_1 ***and*** C_2 ***and ...*** C_n) ***then*** S_n

refers to Table 10. for logical proof

ECAND3 describes a set of conditions that are combined with conjunction "and" that must occur before the system can trigger a set of reactions, which are combined with conjunction "and" in return.

(E34) When a message has been created and the user has finished editing it, the message is placed in the OutBox and marked as queued.

We refine the requirement by applying the ECAND3 pattern:

- When a message has been created and the user has finished editing the message, then the message is placed in the OutBox.
- When a message has been created and the user has finished editing the message, then the message is marked as queued.

(E35) For Logging and Chat tools, the system shall allow for 'Undo' and 'Redo' functions.

We refine the requirement by applying the **ECAND3** pattern:

- For Logging and Chat tools, the system shall allow for 'Undo' function.

- For Logging and Chat tools, the system shall allow for 'Redo' function.

Another example:

(E36) The system shall display summary on the numbers of payments and total amount for Direct Credit and Cheque on the footer of the report.

We refine the requirement by applying the **ECAND3** pattern:

- The system shall display summary on the numbers of payments and total amount for Direct Credit on the footer of the report.
- The system shall display summary on the numbers of payments and total amount for Cheque on the footer of the report.

- **"If and only if" Pattern (IFFP)**

Theorem Proof**:**

$$C \leftrightarrow S \Leftrightarrow (C \wedge S) \vee (\neg C \wedge \neg S)$$
$$\Leftrightarrow (C \vee \neg S) \wedge (\neg C \vee S)$$
$$\Leftrightarrow (\neg C \rightarrow \neg S) \wedge (C \rightarrow S)$$
$$\Leftrightarrow (C \rightarrow S) \wedge (\neg C \rightarrow \neg S)$$

IFFP:

$$C \leftrightarrow S \Leftrightarrow (C \rightarrow S) \wedge (\neg C \rightarrow \neg S)$$
$$\Leftrightarrow C \ \textit{then} \ S \ \textbf{and} \ \neg C \ \textit{then} \ \neg S$$

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.

(E37) The system shall print the customer data if and only if the customer data is inputted to it.

Recommendation to rewrite (E37) according to **IFFP**:

- If the customer data is inputted to the system then the system shall print the customer data.
- If the customer data is not inputted to the system then the system shall not print the customer data.

- **Unless Pattern**

Theorem Proof**:** $C \leftrightarrow \neg S \Leftrightarrow (C \wedge \neg S) \vee (\neg C \wedge S)$

$\Leftrightarrow (C \vee S) \wedge (\neg C \vee \neg S)$

$\Leftrightarrow (\neg C \rightarrow S) \wedge (C \rightarrow \neg S)$

UP: $C \leftrightarrow \neg S \Leftrightarrow (\neg C \rightarrow S) \wedge (C \rightarrow \neg S)$

$\Leftrightarrow \neg C$ ***then*** S ***and*** C ***then*** $\neg S$

refers to Table 11. for logical proof

Unless Pattern explicitly describes possible reactions triggered in response to each respective condition.

(E38) The system shall not log an error report unless an error message appears on the screen.

Refinement on the requirement statement can be done by applying the **UP** pattern:

- If an error message does not appear on the system's screen, then the system shall not log an error report.
- If an error message appears on the system's screen, then the system shall log an error report.

- **If-then-else and If-then-otherwise patterns**

  Referring to example taken from [Berry et. al., 2003]:

  (EX) If C_1, S_1. If C_2, S_2. If C_3, S_3. Otherwise S_4.

  There are several possible interpretations of patterns that can be derived due to the ambiguity caused by *otherwise*, which are:

  i. If we are to interpret (EX) from a compiler point of view, then we will define language patterns for every single *if* statement line-by-line:

  - If $(C_1 \rightarrow S_1)$
  - If $(C_2 \rightarrow S_2)$
  - If $(C_3 \rightarrow S_3) \wedge$
  - {Otherwise} $\neg(C_1 \rightarrow S_{10}) \rightarrow S_0$

  In this case, each if-statement will be treated as one separate requirement statement.

  ii. If (EX) is to be considered and elaborated as nested conditions, then (EX) should be written as [Berry et. al., 2003]:

  (EX) If C_1, S_1.

  If C_2, S_2.

  If C_3, S_3.

  Otherwise S_0.

  And the language pattern devised for (EX) is:

  - If $(C_1 \rightarrow S_1) \rightarrow (C_2 \rightarrow S_2)$
  - If $(C_2 \rightarrow S_2) \rightarrow (C_3 \rightarrow S_3)$
  - Otherwise If $\neg(C_3 \rightarrow S_3) \rightarrow S_0$

  Another possible interpretation from (EX) will be:

  (EX) If C_1, S_1.

If C_2, S_2.

If C_3, S_3.

Otherwise S_0.

And the language pattern devised for (EX) is:

- If $(C_1 \rightarrow S_1) \rightarrow (C_2 \rightarrow S_2)$
- If $(C_2 \rightarrow S_2) \rightarrow (C_3 \rightarrow S_3)$
- Otherwise If $\neg(C_1 \rightarrow S_1) \rightarrow S_0$

iii. Another possible interpretation:

- If $(C_1 \rightarrow S_1)$
- If $\neg(C_1 \rightarrow S_1)$ {then} $(C_2 \rightarrow S_2)$
- If $\neg(C_2 \rightarrow S_2)$ {then} $(C_3 \rightarrow S_3)$
- If $\neg(C_3 \rightarrow S_3)$ {then/otherwise} $(C_4 \rightarrow S_4)$

According to our COCP4 pattern [Tjong, 2006], "else" is treated similar to "otherwise".

(E39) The payments for the same Payee will be grouped together only if the "Individual Payment" is "No", else each payment will be treated separately.

In order to eliminate plural ambiguity [Berry, 2000; Schwertel, 2000] and to better illuminate the original requirement statement is written in an if-then-else pattern, we refine the requirement into:

(E40) If the "Individual Payment" is "No" then each payment for the same Payee will be grouped together, else each payment will be treated separately.

Recommendation to rewrite (E40) according to COCP4 pattern:

- If the "Individual Payment" is "No" then each payment for the same Payee will be grouped together or if the "Individual Payment" is not "No" then each payment will be treated separately.

- **"OR" Pattern**

  We observe 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. Hence, in our study, we introduce several "or" patterns to be adapted in writing the requirements statements.

  Generic 'OR' Pattern (GOR)

  Pattern: $R_1 \ or \ R_2 \ \ldots \ or \ R_n$

  Theorem: $R_1 \vee R_2 \vee \ldots \vee R_n$ will remains as such

GOR: R_1 ***or*** R_2 ***or*** $\ldots$ R_n

  (E41) The system should never allow the lift to move above the top floor or below the bottom floor.

  Compound OR Condition Pattern (COCP)

  Pattern: $C_1 \ or \ C_2 \ \ldots \ or \ C_n$

  $then \ S$

  Theorem Proof: $(C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg(C_1 \vee C_2) \vee S)$

  $\Leftrightarrow ((\neg C_1 \wedge \neg C_2) \vee S)$

  $\Leftrightarrow ((\neg C_1 \vee S) \wedge (\neg C_2 \vee S))$

$$\Leftrightarrow \left(\left(C_1 \rightarrow S\right) \wedge \left(C_2 \rightarrow S\right)\right)$$

COCP1: $\left(C_1 \vee C_2\right) \rightarrow S \Leftrightarrow \left(\left(C_1 \rightarrow S\right) \wedge \left(C_2 \rightarrow S\right)\right)$

$$\Leftrightarrow C_1 \textit{ then } S \textbf{ and } C_2 \textit{ then } S$$

(refer to Table 12. for a logical proof)

And according to GAND rule, C_1 ***then*** S **and** C_2 ***then*** S can be simplified to:

COCP1:
- C_1 ***then*** S
- C_2 ***then*** S
- …
- C_n ***then*** S

COCP1 describes several or compound conditions that should occur before the system can trigger a reaction or response in return.

(E42) If the book's ISBN or title is not in the system then the book is not available.

Recommendation:

- If the book's ISBN is not in the system, then the book is not available.
- If the book's title is not in the system, then the book is not available.

Pattern: C *then* S_1 *or* S_2

Theorem Proof: $C \rightarrow \left(S_1 \vee S_2\right) \Leftrightarrow \left(\neg C \vee \left(S_1 \vee S_2\right)\right)$

$$\Leftrightarrow \left(\left(\neg C \vee S_1\right) \vee \left(\neg C \vee S_2\right)\right)$$

$$\Leftrightarrow \left(\left(C \rightarrow S_1\right) \vee \left(C \rightarrow S_2\right)\right)$$

COCP2: $C \to (S_1 \vee S_2) \Leftrightarrow ((C \to S_1) \vee (C \to S_2))$

$\Leftrightarrow C$ ***then*** S_1 **or** C ***then*** S_2

(refer to Table 13. for a logical proof)

COCP2 describes the occurrence of a specific condition will cause the system to trigger several or compound reactions or responses in return.

(E43) If the user logs in to the system as a Project Manager, then the Add or Update buttons on the system's screen will be enabled.

Recommendation to rewrite (E43) according to COCP2:

- If the user logs in to the system as a Project Manager, then the Add button on the system's screen will be enabled or If the user logs in to the system as a Project Manager, then the Update buttons on the system's screen will be enabled.

Pattern: $C \to S$ { ***else*** | ***otherwise*** } $\overline{S}$

COCP3: $(C \to S_1) \vee (\overline{C} \to \overline{S})$

COCP3 describes the occurrence of a specific condition will cause the system to trigger appropriate reaction.

(E44) If the book's title is in the system, the user can borrow the book otherwise the user can't borrow the book.

Recommendation to rewrite (E44) according to **COCP3**:

- If the book's title is in the system, the user can borrow the book or If the book's title is not in the system, the user can't borrow the book.

Pattern: $C \rightarrow S_1$ { ***else*** | ***otherwise*** } S_2

COCP4:	$(C \rightarrow S_1) \vee (\overline{C} \rightarrow S_2)$

COCP4 describes if the condition is negated, the system must trigger another response or action.

(E45) If the book's title is in the system, the user can borrow the book otherwise the user will submit book's request to the librarian.

We refine (E45) according to **COCP4**:

- If the book's title is in the system, the user can borrow the book or if the book's title is not in the system, the user will submit book's request to the librarian.

COCP5, COCP6 and **COCP7** describe the combined occurrence of negated conditions will cause the system to trigger specific reaction in return.

Pattern: {***Not*** | ***Never*** | ***Neither***} (C_1 {***or*** | ***nor*** } C_2) ***then*** S

Theorem Proof: $\neg(C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg C_1 \wedge \neg C_2) \rightarrow S$

Let $X = \neg C_1, Y = \neg C_2$ $\Leftrightarrow (X \wedge Y) \rightarrow S$

(Derived from **CACP1**) $\Leftrightarrow (X \rightarrow S) \vee (Y \rightarrow S)$

$\Leftrightarrow (\neg C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$

COCP5:	$\neg(C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$
	$\Leftrightarrow \neg C_1$ ***then*** S **or** $\neg C_2$ ***then*** S
(refer to Table 14. for a logical proof)	

(E46) 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.

Recommendation to rewrite (E46) according to COCP5:

- If the system doesn't receive the requested data within 24 hours, then the system must automatically prompt an alert message to the user or 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.

Pattern: $\{\textit{Not} \mid \textit{Never}\}\ C_1\ \textit{or}\ C_2\ \textit{then}\ S$

or

$C_1\ \textit{or}\ C_2\ \{\textit{Not} \mid \textit{Never}\}\ \textit{then}\ S$

Theorem Proof**:** $(\neg C_1 \vee C_2) \rightarrow S \Leftrightarrow ((C_1 \wedge \neg C_2) \vee S)$

$$\Leftrightarrow (C_1 \vee S) \wedge (\neg C_2 \vee S)$$

$$\Leftrightarrow (\neg C_1 \rightarrow S) \wedge (C_2 \rightarrow S)$$

and so does

$$(C_1 \vee \neg C_2) \rightarrow S \Leftrightarrow (C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$$

COCP6: $(\neg C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg C_1 \rightarrow S) \wedge (C_2 \rightarrow S)$

$$\Leftrightarrow \neg C_1\ \textit{then}\ S\ \textbf{and}\ C_2\ \textit{then}\ S$$

(refer to Table 15. for a logical proof)

COCP7: $(C_1 \vee \neg C_2) \rightarrow S \Leftrightarrow (C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$

$$\Leftrightarrow C_1\ \textit{then}\ S\ \textbf{and}\ \neg C_2\ \textit{then}\ S$$

(refer to Table 16. for a logical proof)

(E47) If the user does not load data or enter illegal strings into the system, then an information window will pop-up.

Recommendation to rewrite (E47) according to COCP6:

- If the user does not load data into the system, then an information window will pop-up.
- If the users enter illegal strings into the system, then an information window will pop-up.

Elaborated OR Patterns (ECOR)

Pattern: $(C_1$ ***or*** C_2 ***or*** ... $C_n)$ ***then*** S

Theorem Proof: $(C_1 \vee C_2 \vee ...C_n) \rightarrow S$

$\Leftrightarrow \neg(C_1 \vee C_2 \vee ...C_n) \vee S$

$\Leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge ....\neg C_n) \vee S$

$\Leftrightarrow (\neg C_1 \vee S) \wedge (\neg C_2 \vee S) \wedge ...(\neg C_n \vee S)$

$\Leftrightarrow (C_1 \rightarrow S) \wedge (C_2 \rightarrow S) \wedge ...(C_n \rightarrow S)$

ECOR1: $(C_1 \vee C_2 \vee ...C_n) \rightarrow S$

$\Leftrightarrow (C_1 \rightarrow S) \wedge (C_2 \rightarrow S) \wedge ...(C_n \rightarrow S)$

$\Leftrightarrow$ $(C_1$ ***then*** $S)$ ***and*** $(C_2$ ***then*** $S)$ ***and*** ...$(C_n$ ***then*** $S)$

ECOR1 describes a set of possible conditions are combined with disjunction "or" that should occur before the system can trigger a reaction or response in return.

(E48) If the user skips or discontinues or cancels a login attempt during the login process, the system will return to the login page.

Recommendation to rewrite (E48) according to ECOR1:

- If the user skips a login attempt during the login process, the system will return to the login page.
- If the user discontinues a login attempt during the login process, the system will return to the login page.
- If the user cancels a login attempt during the login process, the system will return to the login page.

Pattern: C ***then*** $(S_1$ ***or*** S_2 ***or*** ... $S_n)$

Theorem Proof: $C \rightarrow (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow \neg C \vee (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (\neg C \vee S_1) \vee (\neg C \vee S_2) \vee ...(\neg C \vee S_n)$

$\Leftrightarrow (C \rightarrow S_1) \vee (C \rightarrow S_2) \vee ...(C \rightarrow S_n)$

ECOR2: $C \rightarrow (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (C \rightarrow S_1) \vee (C \rightarrow S_2) \vee ...(C \rightarrow S_n)$

$\Leftrightarrow$ $(C$ ***then*** $S_1)$ ***or*** $(C$ ***then*** $S_2)$ ***or*** ... $(C$ ***then*** $S_n)$

For an example, see (E43).

ECOR2 describes the occurrence of one particular condition that causes the system to trigger a set of reactions, which are combined with disjunction "or".

Pattern: $(C_1$ ***or*** C_2 ***or*** ... $C_n)$ ***then*** $(S_1$ ***or*** S_2 ***or***... $S_n)$

Theorem Proof:

$(C_1 \vee C_2 \vee ...C_n) \rightarrow (S_1 \vee S_2 \vee ...S_n) \Leftrightarrow (\neg(C_1 \vee C_2 \vee ...C_n)) \vee (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge ...\neg C_n) \vee (S_1 \vee S_2 \vee ...S_n)$

$$\Leftrightarrow (\neg C_1 \vee (S_1 \vee S_2 \vee ...S_n)) \wedge (\neg C_2 \vee (S_1 \vee S_2 \vee ...S_n)) \wedge ...(\neg C_n \vee (S_1 \vee S_2 \vee ...S_n))$$

$$\Leftrightarrow (C_1 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \wedge (C_2 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \wedge (C_n \rightarrow (S_1 \vee S_2 \vee ...S_n))$$

EC0R3: $(C_1 \vee C_2 \vee ...C_n) \rightarrow (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (C_1 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \wedge (C_2 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \wedge ...(C_n \rightarrow (S_1 \vee S_2 \vee ...S_n$

$\Leftrightarrow$ (C_1 ***then*** (S_1 ***or*** S_2 ***or*** ... S_n)) ***and*** (C_2 ***then*** (S_1 ***or*** S_2 ***or*** ... S_n)) ***and*** ***....*** (C_n ***then*** (S_1 ***or*** S_2 ***or*** ... S_n))

refers to Table 17. for a logical proof

ECOR3 describes a set of possible conditions that are combined with disjunction "or" that should occur before the system can trigger a set of reactions, which are combined with disjunction "or" in return.

(E49) If the user requests to close a window or exit the system after making uncommitted changes to a screen, the system shall prompt the user to commit or cancel those changes.

We refine the requirement by applying the **ECOR3** pattern:

- If the user requests to close a window after making uncommitted changes to the system's screen, the system shall prompt the user to commit or cancel the uncommitted changes.
- If the user requests to exit the system after making uncommitted changes to the system's screen, the system shall prompt the user to commit or cancel the uncommitted changes.

And according to **GAND** and **ECOR2** pattern, further pattern simplifications can be derived from the above theorem proof are:

- $C_1 \rightarrow (S_1 \vee S_2 \vee ...S_n) \Leftrightarrow (C_1 \rightarrow S_1) \vee (C_1 \rightarrow S_2) \vee ...(C_1 \rightarrow S_n)$

- $C_2 \rightarrow (S_1 \vee S_2 \vee ...S_n) \Leftrightarrow (C_2 \rightarrow S_1) \vee (C_2 \rightarrow S_2) \vee ...(C_2 \rightarrow S_n)$

- …

- $C_n \rightarrow (S_1 \vee S_2 \vee ...S_n) \Leftrightarrow (C_n \rightarrow S_1) \vee (C_n \rightarrow S_2) \vee ...(C_n \rightarrow S_n)$

- **Combination of AND and OR occurrence in the language patterns**

  We identify some complicated requirements that are using both "and" and "or" words. It can be observed from the examples shown below that these requirements are ambiguous and may cause misinterpretations. Requirements engineer needs to carefully address the options that are bonded with "and" and the options that are bonded with "or".

  ECOM1, ECOM2, ECOM3 and ECOM4 basically describe a set of conditions that are combined with conjunction "and" and disjunction "or" that must occur before the system can trigger a specific reaction in return.

  Pattern: $(C_p \ \textbf{\textit{and}} \ (C_1 \ \textbf{\textit{or}} \ C_2 \ \textbf{\textit{or}} \ ... \ C_n)) \ \textbf{\textit{then}} \ S$

  Theorem Proof:

  $(C_p \wedge (C_1 \vee C_2 \vee ...C_n)) \rightarrow S \Leftrightarrow (\neg(C_p \wedge (C_1 \vee C_2 \vee ...C_n))) \vee S$

  $\Leftrightarrow (\neg C_p \vee \neg(C_1 \vee C_2 \vee ...C_n)) \vee S$

  $\Leftrightarrow ((\neg C_p \vee S) \vee (\neg(C_1 \vee C_2 \vee ...C_n) \vee S))$

  $\Leftrightarrow (C_p \rightarrow S) \vee ((C_1 \vee C_2 \vee ...C_n) \rightarrow S)$

  ECOM1:

  $(C_p \wedge (C_1 \vee C_2 \vee ...C_n)) \rightarrow S \Leftrightarrow (C_p \rightarrow S) \vee ((C_1 \vee C_2 \vee ...C_n) \rightarrow S)$

  $\Leftrightarrow C_p \ \textbf{\textit{then}} \ S \ \textbf{\textit{or}}$

  $C_1 \ \textbf{\textit{or}} \ C_2 \ \textbf{\textit{or}} \ ... \ C_n \ \textbf{\textit{then}} \ S$

  refers to Table 18. for a logical proof

Pattern: (C_p ***or*** (C_1 ***and*** C_2 ***and*** ... C_n)) ***then*** S

Theorem Proof:

$$(C_p \vee (C_1 \wedge C_2 \wedge ...C_n))S \Leftrightarrow (\neg(C_p \vee (C_1 \wedge C_2 \wedge ...C_n))) \vee S$$

$$\Leftrightarrow (\neg C_p \wedge \neg(C_1 \wedge C_2 \wedge ...C_n)) \vee S$$

$$\Leftrightarrow ((\neg C_p \vee S) \wedge (\neg(C_1 \wedge C_2 \wedge ...C_n) \vee S))$$

$$\Leftrightarrow (C_p \rightarrow S) \wedge ((C_1 \wedge C_2 \wedge ...C_n) \rightarrow S)$$

ECOM2:

$$(C_p \vee (C_1 \wedge C_2 \wedge ...C_n)) \rightarrow S \Leftrightarrow (C_p \rightarrow S) \wedge ((C_1 \wedge C_2 \wedge ...C_n) \rightarrow S)$$

$$\Leftrightarrow C_p \ \textbf{\textit{then}} \ S \ \textbf{\textit{and}}$$

$$C_1 \ \textbf{\textit{and}} \ C_2 \ \textbf{\textit{and}} \ ... \ C_n \ \textbf{\textit{then}} \ S$$

refers to Table 19. for a logical proof

(E50) Associates are able to pick the Payee, GL Code, Branch Code or Standing Instruction details by picking from a predefined list.

We rewrite (EX) to eliminate the plural ambiguities:

(E51) Every associate is able to pick the Payee and GL Code and Branch Code or Standing Instruction details by picking from a predefined list.

Further refinement that can be made according to ECOM1 pattern:

- Every associate is able to pick the Payee and GL Code and Branch Code by picking from a predefined list.
- Every associate is able to pick the Standing Instruction details by picking from a predefined list.

Pattern: $((C_1 \textbf{ and } C_2 \textbf{ and } \ldots C_n) \textbf{ and } (C_p \textbf{ or } C_q \textbf{ or } \ldots C_z)) \textbf{ then } S$

Theorem Proof: $\big((C_1 \wedge C_2 \wedge \ldots C_n) \wedge (C_p \vee C_q \vee \ldots C_z)\big) \rightarrow S$

$\Leftrightarrow \big(\neg\big((C_1 \wedge C_2 \wedge \ldots C_n) \wedge (C_p \vee C_q \vee \ldots C_z)\big)\big) \vee S$

$\Leftrightarrow \big(\neg(C_1 \wedge C_2 \wedge \ldots C_n) \vee \neg(C_p \vee C_q \vee \ldots C_z)\big) \vee S$

$\Leftrightarrow \big((C_1 \wedge C_2 \wedge \ldots C_n) \rightarrow S\big) \vee \big((C_p \vee C_q \vee \ldots C_z) \rightarrow S\big)$

ECOM3: $\big((C_1 \wedge C_2 \wedge \ldots C_n) \wedge (C_p \vee C_q \vee \ldots C_z)\big) \rightarrow S$

$\Leftrightarrow \big((C_1 \wedge C_2 \wedge \ldots C_n) \rightarrow S\big) \vee \big((C_p \vee C_q \vee \ldots C_z) \rightarrow S\big)$

$\Leftrightarrow C_1 \textbf{ and } C_2 \textbf{ and } \ldots C_n \textbf{ then } S \textbf{ or}$

$C_p \textbf{ or } C_q \textbf{ or } \ldots C_z \textbf{ then } S$

Pattern: $((C_1 \textbf{ and } C_2 \textbf{ and } \ldots C_n) \textbf{ or } (C_p \textbf{ or } C_q \textbf{ or } \ldots C_z)) \textbf{ then } S$

Theorem Proof: $\big((C_1 \wedge C_2 \wedge \ldots C_n) \vee (C_p \vee C_q \vee \ldots C_z)\big) \rightarrow S$

$\Leftrightarrow \big(\neg\big((C_1 \wedge C_2 \wedge \ldots C_n) \vee (C_p \vee C_q \vee \ldots C_z)\big)\big) \vee S$

$\Leftrightarrow \big(\neg(C_1 \wedge C_2 \wedge \ldots C_n) \wedge \neg(C_p \vee C_q \vee \ldots C_z)\big) \vee S$

$\Leftrightarrow \big((C_1 \wedge C_2 \wedge \ldots C_n) \rightarrow S\big) \wedge \big((C_p \vee C_q \vee \ldots C_z) \rightarrow S\big)$

ECOM4: $\big((C_1 \wedge C_2 \wedge \ldots C_n) \vee (C_p \vee C_q \vee \ldots C_z)\big) \rightarrow S$

$\Leftrightarrow \big((C_1 \wedge C_2 \wedge \ldots C_n) \rightarrow S\big) \wedge \big((C_p \vee C_q \vee \ldots C_z) \rightarrow S\big)$

$\Leftrightarrow C_1 \textbf{ and } C_2 \textbf{ and } \ldots C_n \textbf{ then } S \textbf{ and}$

$C_p \textbf{ or } C_q \textbf{ or } \ldots C_z \textbf{ then } S$

Pattern: (C_1 ***and*** C_2 ***and*** ... C_n) ***then*** (S_1 ***or*** S_2 ***or ...*** S_n)

Theorem Proof: $(C_1 \wedge C_2 \wedge ...C_n) \rightarrow (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow \neg(C_1 \wedge C_2 \wedge ...C_n) \vee (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (\neg C_1 \vee \neg C_2 \vee \neg C_2) \vee (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (\neg C_1 \vee (S_1 \vee S_2 \vee ...S_n)) \vee (\neg C_2 \vee (S_1 \vee S_2 \vee ...S_n)) \vee ...(\neg C_n \vee (S_1 \vee S_2 \vee ...S_n))$

$\Leftrightarrow (C_1 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \vee (C_2 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \vee ...(C_n \rightarrow (S_1 \vee S_2 \vee ...S_n))$

ECOM5: $(C_1 \wedge C_2 \wedge ...C_n) \rightarrow (S_1 \vee S_2 \vee ...S_n)$

$\Leftrightarrow (C_1 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \vee (C_2 \rightarrow (S_1 \vee S_2 \vee ...S_n)) \vee ...(C_n \rightarrow (S_1 \vee S_2 \vee ...$

$\Leftrightarrow$ (C_1 ***then*** (S_1 ***or*** S_2 ***or*** ... S_n)) ***or***

(C_2 ***then*** (S_1 ***or*** S_2 ***or*** ... S_n)) ***or*** ...

(C_n ***then*** (S_1 ***or*** S_2 ***or*** ... S_n))

ECOM5 describes a set of conditions combined with conjunction "and" that must occur before the system can trigger a set of reactions, which are combined with disjunction "or" in return.

(E52) If the configuration file and the identity of the information processed are available in the system, the system shall be able to reconstruct or reprocess the data.

We refine the requirement by applying the ECOM5 pattern:

- If the configuration file is available in the system, the system shall be able to reconstruct or reprocess the data or If the identity of the information processed is available in the system, the system shall be able to reconstruct or reprocess the data.

Another example:

(E53) Associates and officers can print Payment Report by Staff ID or Import File name.

And again, we refine the requirement by applying the **ECOM4** pattern:

- An associate can print Payment Report by Staff ID or Import File Name or An officer can print Payment Report by Staff ID or Import File Name.

Pattern: $(C_1$ ***or*** C_2 ***or ...*** $C_n)$ ***then*** $(S_1$ ***and*** S_2 ***and*** ... $S_n)$

Theorem Proof: $(C_1 \vee C_2 \vee ...C_n) \rightarrow (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow \neg(C_1 \vee C_2 \vee ...C_n) \vee (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow (\neg(C_1 \vee C_2 \vee ...C_n) \vee S_1) \wedge (\neg(C_1 \vee C_2 \vee ...C_n) \vee S_2) \wedge ...$
$(\neg(C_1 \vee C_2 \vee ...C_n) \vee S_n)$

$\Leftrightarrow ((C_1 \vee C_2 \vee ...C_n) \rightarrow S_1) \wedge ((C_1 \vee C_2 \vee ...C_n) \rightarrow S_2) \wedge ...$
$((C_1 \vee C_2 \vee ...C_n) \rightarrow S_n)$

EC0M6: $(C_1 \vee C_2 \vee ...C_n) \rightarrow (S_1 \wedge S_2 \wedge ...S_n)$

$\Leftrightarrow ((C_1 \vee C_2 \vee ...C_n) \rightarrow S_1) \wedge ((C_1 \vee C_2 \vee ...C_n) \rightarrow S_2) \wedge ...$
$((C_1 \vee C_2 \vee ...C_n) \rightarrow S_n)$

By adopting the ECOR1 pattern, the pattern can be further simplified into:

$\Leftrightarrow (C_1 \rightarrow S_1) \wedge ...(C_1 \rightarrow S_n) \wedge (C_2 \rightarrow S_1) \wedge ...(C_2 \rightarrow S_n) \wedge ...(C_n \rightarrow S_1) \wedge ...$
$(C_n \rightarrow S_n)$

$\Leftrightarrow C_1$ ***then*** S_1 ***and ...*** C_1 ***then*** S_n ***and*** C_2 ***then*** S_1 ***and ...*** C_2 ***then*** S_n ***and*** C_n ***then*** S_1 ***and ...*** C_n ***then*** S_n

ECOM6 describes a set of possible conditions combined with disjunction "or" that should occur before the system can trigger a set of reactions, which are combined with conjunction "and" in return.

(E54) The user shall have the ability to redirect warning or error messages to the operator log and a printer.

We rewrite the requirement to lessen the missing and vague information as:

(E55) If a warning or error message appears on the system's screen, the user shall have the ability to redirect the appeared warning or error message to the operator log and printer.

Then we refine the requirement by applying the **ECOM6** pattern:

- If a warning message appears on the system's screen, the user shall have the ability to redirect the appeared warning message to the operator log.
- If a warning message appears on the system's screen, the user shall have the ability to redirect the appeared warning message to the printer.
- If an error message appears on the system's screen, the user shall have the ability to redirect the appeared error message to the operator log.
- If an error message appears on the system's screen, the user shall have the ability to redirect the appeared error message to the printer.

- **Time Pattern (TP)**

  TPs are the patterns to be adapted in writing requirements that concern with time.

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**

  Phrase(s) which is considered as clause will be taken as guided reference to the requirement(s) that it's tailored to. This will eliminate the informality or ambiguity caused by long sentences (due to the occurrence of clauses or phrases).

Subordinate Clause (Sub_Clause) 1:

{***that*** | ***which***} VERB [COMPLEMENT]

(E56) A popup box *that requires a response from the user* will remain in the system's foreground until the user clicks on the popup box.

Subordinate Clause (Sub_Clause) 2:

{***but*** | ***as*** | ***since*** | ***while*** | ***where*** } NOUN_PHRASE VERB [COMPLEMENT]

(E57) The DCS shall be able to display the status of ongoing projects, *where the DCS is maintaining information and the user has access* [DCS].

Subordinate Clause (Sub_Clause) 3:

{***in order to*** | ***in [the] case of*** | ***such that*** | ***regardless [of]*** | ***given that*** |...} NOUN_PHRASE VERB [COMPLEMENT]

(E58) The monitoring system should be lightweight, *such that when running online, the system's consumption of memory, CPU, and other resources of HLT processing node are small (~ 1%) compared to the demands of the software being monitored.*

4. Future Direction

This research work will further continue on developing a tool that incorporates all the defined language patterns. A system coined SREE (Systemised Requirements Engineering Environment), 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 NLR 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 NLRs to be incorporated in SREE:

- First, the requirements document is analysed, sorted and rewritten into a set of structured requirements sentences by applying the authoring 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.

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.

5. Conclusion

This report addresses the current research work in defining guideline rules and 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 NLR state of practice.

We believe that by adopting the language patterns while writing the NLRSs, the level of ambiguity and possible introduction of imprecision can be reduced. Furthermore, the language patterns are specifically designed not only to be adaptable in one particular domain, but in most general domains. On the other hand, the rules work as guideline that assists the requirements engineer in authoring the NLRs writing. Therefore, the rules and language patterns should be used in combination to achieve maximum reduction of ambiguity and imprecisions.

To validate the usefulness and adaptable of the guideline rules and language patterns, we have conducted studies on several sets of requirements and extracted some real industrial requirements to be presented as examples. From there, we have rewritten the ambiguous requirements sentences by applying both the language patterns and rules.

6. References

[Ambriola and Gervasi, 2003]

Ambriola, V. and Gervasi, V.: *The CIRCE approach to the systematic analysis of NL requirements,* 2003*,* Technical Report: TR-03-05, Università Di Pisa

[Barr, 1999]

Barr V., Identifikation von Spezifikationsmustern im Echtzeitentwurf anhand der Fallstudie Antiblockiersystem, 1999, Diplomarbeit der Universitaet Oldenburg, Fachbereich Informatik

[Denger et. al., 2001]

Denger C., Jőrg D., Kamsties E., *QUASAR A Survey on Approaches for Writing Precise Natural Language Requirements*, 2001, Fraunhofer IESE

[Denger, 2002]

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, 2002 Germany

[Denger et. al., 2003]

Denger C., Berry D.M., Kamsties E., *Higher Quality Requirements Specifications through Natural Language Patterns*, 2003, Proceedings of the IEEE International Conference on Software – Science, Technology & Engineering (SwSTE'03)

[Fabbrini et. al., 2000]

Fabbrini F., Fusani M., Gnesi G. and Lami G., *Quality Evolution of Software Requirements Specifications*, 2000, Proceedings Software and Internet Quality Week 2000 Conference, San Fransisco, CA

[Fabbrini et. al., 2002]

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*, 2002, Proceedings of the 26th Annual NaSA Goddard Software Engineering Workshop (SEW'01)

[Fillmore, 1968]

Fillmore C.J., *The Case for Case*, 1968, in Universals in Linguistic Theory

[Firesmith, 2003]

Firesmith D., *Specifying Good Requirements*, 2003, Journal of Object Technology 2

[Fliedl et. al., 2000]

Fliedl G., Kop C., Mayerthaler W., Mayr H.C., Winkler C., *Linguistic Aspects of Dynamics in Requirements Specifications & Linguistically Based Requirements Engineering,*, 2000, Data & Knowledge Engineering, Netherlands

[Fuchs and Schwitter, 1996]

Fuchs N.E. and Schwitter R., *Controlled English for Requirements Specification*, 1996, IEEE Computer Special Issue on Interactive Natural Language Processing

[Götz and Rupp, 1999]

Götz R. and Rupp C., *Regelwerk Natürlichsprachliche Methode*, 1999, Sophist, Nürnberg, Germany

[Hooks, 1994]

Hooks I., *Writing Good Requirements*, 1994, Vol. 2, pp. 197-203, Proceedings of the Fourth International Symposium of the NCOSE 2, San Jose, CA

[Juristo et. al., 2000]

Juristo N., Moreno A.M. and Lopez M., *How to Use Linguistic Instruments for OOA*, 2000, pp. 80-89 Proceedings of the IEEE International Conference on Software

[Lami, 2005]

Lami Giuseppe., *QuARS: A Tool for Analysing Requirements*, September 2005, Carnegie Mellon University

[Lanman, 2002]

Lanman J. T., *Using Formal Methods in Requirements Engineering Version 1.0*, 2002, Department of Computing and Mathematics, Embry-Riddle Aeronautical University

[Martinez et.al., 2004]

Martinez A., Pastor O., Estrada H., A pattern language to join early and late requirements, 2004, pp 51-64 Anais do WER04 - Workshop em Engenharia de Requisitos, Tandil, Argentina

[Ohnishi, 1994]

Ohnishi A., *Customizable Software Requirements Languages*, 1994, Proceedings of the Eighth International Computer Software and Application Conference (COMPSAC), IEEE Computer Society, Los Alamitos, CA

[Rolland and Proix, 1992]

Rolland C. and Proix C., *A Natural Language Approach for Requirements Engineering*, 1992, pp. 257-277 in Proceedings of Conference on Advanced Informations Systems Engineering, CAiSE, Manchester UK

[Wilson et. al., 1996]

Wilson W.M., Rosenberg L.H. and Hyatt L.E., *Automated Quality Analysis of Natural Language Requirements Specifications*, 1996, NASA Software

Assurance Technology Center, The Software Assurance Technology Center (SATC), NASA Goddard Space Flight Center (GSFC), Greenbelt, MD

Requirements Documents

[DCS, 2002] available at

http://www.astro.ucla.edu/~shuping/SOFIA/Documents/DCS_SRD_Rev1.pdf

Nelbach F.J., *Software Requirements Document For the Data Cycle System (DCS) Of The SOFIA Project*, 2002, Universities Space Research Association

[EVLA, 2003] available at

http://www.aoc.nrao.edu/evla/techdocs/computer/workdocs/array-sw-rqmts.pdf

Moeser R., Perley P., *EVLA Operations Interface, Software Requirements*, EVLA-SW-003 Revision: 2.5, 2003

[LAT, 2000] available at

http://www-glast.slac.stanford.edu/IntegrationTest/DataHandling/docs/LAT-SS-00020-06.pdf

Dubois R., *Large Area Telescope (LAT) Science Analysis Software Specification*, 2000, GE-0000X-DO

[PESA, 2001] available at

http://www.pp.rhul.ac.uk/atlas/newsw/requirements/1.0.2/

George S., *PESA High-Level Trigger Selection Software Requirements*, 2001

[Eng, 2005]

Eng Chee Siong, "Batch Poster System, Detailed Business Requirements", EDS MySC Malaysia, 2005

[Bray, 2002]

Bray, I.K., *An Introduction To Requirements Engineering*, 2002, © Pearson Education Limited

- The yacht racing results (YRR) case study, p. 337-340
- The lift controller case study, p. 357-358

7. Appendices

7.1 List of Tables

- Table 1. Standard ARM Indicators [Wilson, et. al, 1996]

	Imperative	**Continuance**	**Directive**	**Option**	**Weak Phrases**	**Incompletes**
INDICATORS	shall	below:	e.g.	can	adequate	TBD
	must	as follows:	i.e.	may	as appropriate	TBS
	is required to	following:	for example	optionally	be able to	TBE
	are applicable	listed:	figure		be capable of	TBC
	are to	in particular:	table		capability of/to	not defined
	respon-sible for	support:	note:		easy to	not determined
	will	and			effective	but not limited to
	should	:			as required	as a minimum
					normal	
					provide for	
					timely	

- Imperatives are words and phrases that command something must be provided.
- Continuances are phrases and words used to introduce the specification of requirements at a lower level.
- Directives are category of words and phrases that point to illustrative information within the requirements document
- Options are category of words that give the developer latitude in satisfying the specification statements containing them.
- Weak Phrases are category of clauses that are apt to cause uncertainty and leave room for multiple interpretations.

- Table 2. QUARS Indicators [Fabbrini et. al, 2000]

	Implicit Sentence	**Multiple Sentence**	**Optional Sentence**	**Weak Sentence**
INDICATORS	Demonstrative Adjective: this, these, that, those	>1 subject	possibly	can
	Pronouns: it, they	> 1 main verb	eventually	could
	Preposition: above, below,…	>1 direct complement	in case of	may
	Adjective: previous, next, last, first, following, …	>1 indirect complement	if possible	
			if appropriate	
			if needed …	

- Table 3. More QUARS Indicators [Fabbrini et. al, 2000]

	Subjective Sentence	**Vague Sentence**	**Underreferenced Sentence**
I	Having in mind	Easy	According to
D	Take (into) account	Strong	On (the) basis of
I	Take into consideration	Good	Relatively to
C	Similar	Bad	Compliant with
A	Similarly	Useful	Conformant to
T	Better	Significant	…
O	Worse	Adequate	
R	As [adjective] as possible	recent	
S	…		

- Table 4. Logical Representation of ((A $\vee$ B) $\vee$(A $\wedge$ B)) is similar to (A $\vee$ B)

A	B	$A \vee B$	$A \wedge B$	$(A \vee B) \vee (A \wedge B)$
0	0	**0**	0	**0**
0	1	**1**	0	**1**
1	0	**1**	0	**1**
1	1	**1**	1	**1**

- Table 5. Logical Representation of $(C_1 \wedge C_2 \rightarrow S)$ is similar to $((C_1 \rightarrow S) \vee (C_2 \rightarrow S))$

C_1	C_2	S	$(C_1 \wedge C_2 \rightarrow S)$	$((C_1 \rightarrow S) \vee (C_2 \rightarrow S))$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

- Table 6. Logical Representation of $(C \rightarrow S_1 \wedge S_2)$ is similar to $((C \rightarrow S_1) \wedge (C \rightarrow S_2))$

C	S_1	S_2	$(C \rightarrow S_1 \wedge S_2)$	$((C \rightarrow S_1) \wedge (C \rightarrow S_2))$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- Table 7. Logical Representation of $\neg(C_1 \wedge C_2) \rightarrow S$ is similar to $(\neg C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$

C_1	C_2	S	$\neg(C_1 \wedge C_2) \rightarrow S$	$(\neg C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- Table 8. Logical Representation of $(\neg C_1 \wedge C_2) \rightarrow S$ is similar to $(\neg C_1 \rightarrow S) \vee (C_2 \rightarrow S)$

C_1	C_2	S	$(\neg C_1 \wedge C_2) \rightarrow S$	$(\neg C_1 \rightarrow S) \vee (C_2 \rightarrow S)$
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- Table 9. Logical Representation of $(C_1 \wedge \neg C_2) \rightarrow S$ is similar to $(C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$

C_1	C_2	S	$(C_1 \wedge \neg C_2) \rightarrow S$	$(C_1 \rightarrow S) \vee (\neg C_2 \rightarrow S)$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- Table 10. Logical Representation of $(C_1 \wedge C_2) \rightarrow (S_1 \wedge S_2)$ is similar to $((C_1 \wedge C_2) \rightarrow S_1) \wedge ((C_1 \wedge C_2) \rightarrow S_2)$

C_1	C_2	S_1	S_2	$(C_1 \wedge C_2) \rightarrow (S_1 \wedge S_2)$	$((C_1 \wedge C_2) \rightarrow S_1) \wedge ((C_1 \wedge C_2) \rightarrow S_2)$
0	0	0	0	1	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1

0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	1

- Table 11. Logical Representation of $C \leftrightarrow \neg S$ is similar to $(\neg C \rightarrow S) \wedge (C \rightarrow \neg S)$

C	S	$C \leftrightarrow \neg S$	$(\neg C \rightarrow S) \wedge (C \rightarrow \neg S)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

- Table 12. Logical Representation of $(C_1 \vee C_2) \rightarrow S$ is similar to $((C_1 \rightarrow S) \wedge (C_2 \rightarrow S))$

C_1	C_2	S	$(C_1 \vee C_2) \rightarrow S$	$((C_1 \rightarrow S) \wedge (C_2 \rightarrow S))$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1

1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

- Table 13. Logical Representation of $C \to (S_1 \vee S_2)$ is similar to $((C \to S_1) \vee (C \to S_2))$

C_1	C_2	S	$C \to (S_1 \vee S_2)$	$((C \to S_1) \vee (C \to S_2))$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- Table 14. Logical Representation of $\neg(C_1 \vee C_2) \to S$ is similar to $(\neg C_1 \to S) \vee (\neg C_2 \to S)$

C_1	C_2	S	$\neg(C_1 \vee C_2) \to S$	$(\neg C_1 \to S) \vee (\neg C_2 \to S)$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1

1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- Table 15. Logical Representation of $(\neg C_1 \vee C_2) \rightarrow S$ is similar to $(\neg C_1 \rightarrow S) \wedge (C_2 \rightarrow S)$

C_1	C_2	S	$(\neg C_1 \vee C_2) \rightarrow S$	$(\neg C_1 \rightarrow S) \wedge (C_2 \rightarrow S)$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

- Table 16. Logical Representation of $(C_1 \vee \neg C_2) \rightarrow S$ is similar to $(C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$

C_1	C_2	S	$(C_1 \vee \neg C_2) \rightarrow S$	$(C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0

1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

- Table 17. Logical Representation of $(C_1 \vee C_2) \rightarrow (S_1 \vee S_2)$ is similar to $(C_1 \rightarrow (S_1 \vee S_2)) \wedge (C_2 \rightarrow (S_1 \vee S_2))$

C_1	C_2	S_1	S_2	$(C_1 \vee C_2) \rightarrow (S_1 \vee S_2)$	$(C_1 \rightarrow (S_1 \vee S_2)) \wedge (C_2 \rightarrow (S_1 \vee S_2))$
0	0	0	0	1	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

- Table 18. Logical Representation of $(C_p \wedge (C_1 \vee C_2)) \rightarrow S$ is similar to $(C_p \rightarrow S) \vee ((C_1 \vee C_2) \rightarrow S)$

C_p	C_1	C_2	S	$(C_p \wedge (C_1 \vee C_2)) \rightarrow S$	$(C_p \rightarrow S) \vee ((C_1 \vee C_2) \rightarrow S)$
0	0	0	0	1	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	1

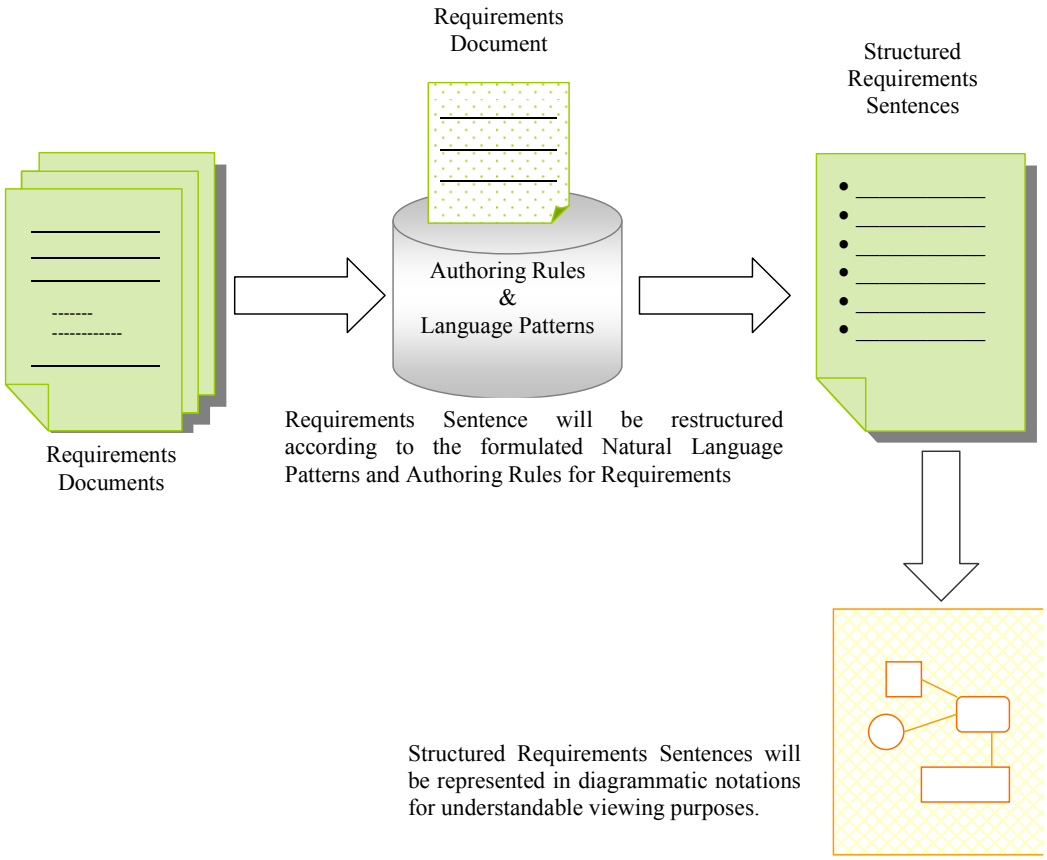
- Table 19. Logical Representation of $(C_p \vee (C_1 \wedge C_2)) \rightarrow S$ is similar to $(C_p \rightarrow S) \wedge ((C_1 \wedge C_2) \rightarrow S)$

C_p	C_1	C_2	S	$(C_p \vee (C_1 \wedge C_2)) \rightarrow S$	$(C_p \rightarrow S) \wedge ((C_1 \wedge C_2) \rightarrow S)$
0	0	0	0	1	1
0	0	0	1	1	1

0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	1

7.2 Figures

- Figure 1. Transformation of Higher Quality Natural Language Requirements Specifications through Natural Language Requirements Pattern


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.