NATURAL LANGUAGE INTERFACES FOR REQUIREMENTS ENGINEERING

Sri Fatimah Tjong, BSc.

Technical Report submitted to the

University of Nottingham

for the degree of Doctor of Philosophy

March 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]. Some have even developed a controlled language for specifying requirements in an almost natural language [Fuchs and Schwitter, 1996]. 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 Technical 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 that 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 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:

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

  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.

	Imperative	**Continuance**	**Directive**	**Option**	**Weak Phrases**	**Incompletes**
I	shall	below:	e.g.	can	adequate	TBD
N	must	as follows:	i.e.	may	as appropriate	TBS
D	is required to	following:	for example	optionally	be able to	TBE
I	are applicable	listed:	figure		be capable of	TBC
C						
A						
T						

O	are to	in particular:	table		capability of/to	not defined
R	respon-sible for	support:	note:		easy to	not determined
S	will	and			effective	but not limited to
	should	:			as required	as a minimum
					normal	
					provide for	
					timely	

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

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

Table 2. 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 3. More QUARS Indicators [Fabbrini et. al, 2000]

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

  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 [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 are focused on the programming aspects, or even patterns for project management [Martinez et. al., 2004]. The patterns defined by Ohnishi and by Rolland and Proix [Ohnishi, 1994; Rolland and Proix, 1992] concentrate to lessen the NLRSs imprecision in the domain of information system and database. Both Barr and Denger [Barr, 1999; Denger, 2002] 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 [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.

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 [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 guiding rules are built up on top of the Denger's authoring rules [Barr, 1999] 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. 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. 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, first, 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].

E.g. 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"…).

E.g. The user shall either be trusted or not trusted [EVLA].

E.g. 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.

E.g. 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.

E.g. 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.).

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

E.g. 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".

E.g. 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".

E.g. 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" …

E.g. 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", ",".

E.g. 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.

E.g. 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.

E.g. 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 [LAT].

3.2 Language Patterns

We examined several case studies and real requirements documents [DCS, 2002; EVLA, 2003; LAT, 2000; PESA, 2001; 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)**

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

E.g. 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)**

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

E.g. 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**

E.g. If the boat's details are amended, the boat's details will not affect the outcome of any races.

ECP2: GP Condition

E.g. 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

E.g. 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 that 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

  E.g. 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: $C_1 \wedge C_2 \wedge \ldots \wedge C_n$

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.

E.g. When the message has been created and the user finished editing the message, then the message will be placed in the Outbox.

Recommendation:

- 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.

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

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.

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

Recommendation:

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

Refinement:

- 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.

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

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

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

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

(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.

E.g. If the user has neither an unidentified nor unauthorised account, the system shall never grant an access to the database.

Recommendations:

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

Refinement:

- 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.

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

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

or

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

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

$\Leftrightarrow \neg C_1\ \textit{then}\ S$ **or** $C_2\ \textit{then}\ S$

(refer to Table 8. for a logical proof)

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

$\Leftrightarrow C_1\ \textit{then}\ S$ **or** $\neg C_2\ \textit{then}\ S$

(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.

E.g. 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.

Recommendations:

- 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.

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

- **"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.

E.g. The system shall print the customer data if and only if the customer data is inputted to it.

Recommendations:

- 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.

- **"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

E.g. 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:
$$(C_1 \vee C_2) \rightarrow S \Leftrightarrow (\neg(C_1 \vee C_2) \vee S)$$
$$\Leftrightarrow ((\neg C_1 \wedge \neg C_2) \vee S)$$
$$\Leftrightarrow ((\neg C_1 \vee S) \wedge (\neg C_2 \vee S))$$
$$\Leftrightarrow ((C_1 \rightarrow S) \wedge (C_2 \rightarrow S))$$

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

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

(refer to Table 10. 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

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

E.g. 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.

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

Pattern: $C \ then \ S_1 \ or \ S_2$

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

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

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

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

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

(refer to Table 11. 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.

E.g. 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:

- 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.

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 11. Logical Representation of $C \to (S_1 \vee S_2)$ is similar to $((C \to S_1) \vee (C \to S_2))$

Pattern: $C \to S \ \{ \textbf{\textit{else}} \mid \textbf{\textit{otherwise}} \} \ \overline{S}$

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.

E.g. If the book's title is in the system, the user can borrow the book otherwise the user can't borrow the book.

Recommendation:

- If the book's title is in the system, the user can borrow the book.
- 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.

E.g. 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.

Recommendation:

- 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.

Refinement:

- If the book's title is in the system, the user can borrow the book.
- If the book's title is not in the system, the 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 12. for a logical proof)

E.g. 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.

Recommendations:

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

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

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

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

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

(refer to Table 13. for a logical proof)

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

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

(refer to Table 14. for a logical proof)

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

Recommendations:

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

Refinement:

- 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.

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 13. 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	$(C_1 \vee \neg C_2) \rightarrow S$	$(C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$
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 14. Logical Representation of $(C_1 \vee \neg C_2) \rightarrow S$ is similar to $(C_1 \rightarrow S) \wedge (\neg C_2 \rightarrow S)$

- **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) that 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]

E.g. 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]

E.g. 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]

E.g. 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.

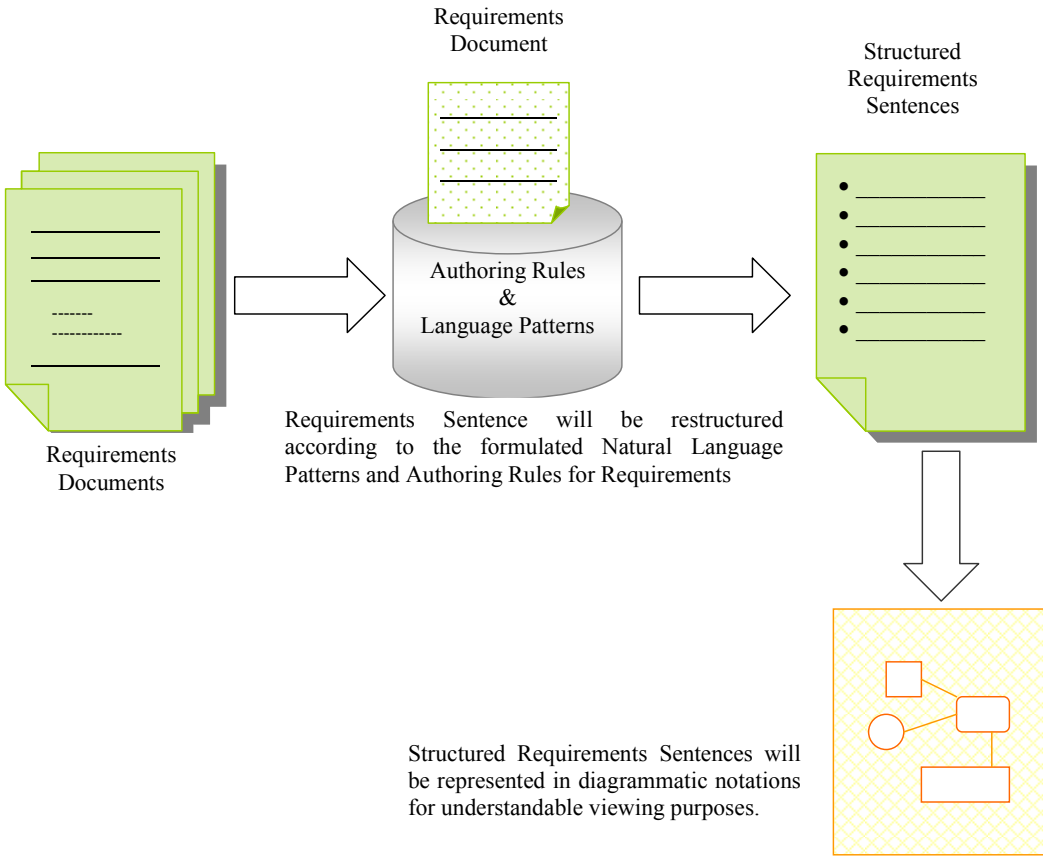

Figure 1. Transformation of Higher Quality Natural Language Requirements Specifications through Natural Language Requirements Pattern

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 guiding 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 works as guidelines that assist 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 guiding 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)

[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

[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