ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

Automatic Conflict Detection in Non-functional requirement Analysis using a Conceptual Graph

Taweewat Luangwiriya[1] and Rachada Kongkachandra[2]

ABSTRACT

Non-functional requirement (NFR) is a complex problem. The conflict of NFRs is a significant obstacle in the requirement analysis phase of software engineering because they can frequently contradict or interfere with other NFRs. Another aspect of the problem derives from the main character of NFRs, since many of them can be related to the same quality attribute. Our research has the objective of representing the NFR by introducing an automatic knowledge representation detection method using conceptual graphs to discover conflicts among NFRs. It consists of extracting the NFR from the input software requirement specification document using NFR XML frame and then trying to detect any disputes between it and other NFRs. A billing system case study from a telecommunications company in Thailand yielded 353 NFR XML frames that could be extracted from the software specification document. These divide into Valid frames from complete NFRs, Incomplete NFR frames, and Invalid NFR frames. The accuracy of the extracting process resulted in a precision of 0.36, a recall of 0.92, and an F measure of 0.52. Moreover, these NFR XML frames revealed representative NFR conflicts for 20 NFRs; and detected two NFRs as an NFR conflict, and reported two NFRs as a potential NFR conflict, otherwise known as a "bad smell NFR" to the system analyst for revising these concerns before submitting them the next development phase.

Article information:

Keywords: Knowledge Representation, Non-functional Requirement Representation, Requirement Conflict Detection, Conceptual Graph, Semantic Graph

Article history:
Received: May 20, 2022
Revised: July 2, 2022
Accepted: November 6, 2022
Published: February 25, 2023 (Online)

DOI: 10.37936/ecti-cit.2023171.248531

1. INTRODUCTION

The conflict problem is a significant problem in every domain. Conflict, basically, is a disagreement between people with opposing opinions or principles. Additionally, if beliefs, needs, facts, etc. conflict, it means that they are very different and cannot easily exist together or both be true. Similarly, in software requirement engineering, conflicts can occur anytime between the software requirements. It is also a huge problem for the requirement engineer and the development team to identify the requirement conflicts before submitting them to the next phase of the software development life cycle. In this section, we provide the basis of software requirement engineering, Non-functional requirement representation, and the problem of software requirement conflict in subsections 1.1, 1.2, and 1.3, respectively.

1.1 Software requirement engineering

Software requirement engineering has as its objective the eliciting of stakeholder needs and then developing them into an agreed set of requirement specifications that can describe a basis for subsequent development activities. For this reason, this process needs to completely clarify problems present in the initial stage of the process and subsequently confirm that their solutions are correct, reasonable, and effective [1]. The requirement analysis phase is an essential and early phase of the system development project – it is then when the analysts have to work closely with the users to identify the user requirement expectations of the system under development. Therefore, the analysts have to analyse the system functions with other quality-oriented characteristics such as availability, modifiability, security, etc and present

[1] The author is with Department of Computer Science, Faculty of Science and Technology, Thammasat University, Patum Thani,12121, Thailand, E-mail: taweewatlu@gmail.com
[2] The author is with Data Science and Innovation Program, College of Interdisciplinary Studies, Thammasat University, Patum Thani,12121, Thailand, E-mail: krachada@tu.ac.th

them as acceptable to the user. As a result of this phase, the requirement specifications are considered fundamental to the rest of the development project. Since the requirement specifications are necessary for user acceptance, it is vital to confirm that they are consistent, complete, and correct.

In requirement engineering, software requirements divide into 'functional' and 'Non-functional.' A functional requirement is one that considers a system's function or unit. It is a specification of the relation between inputs and outputs. On the other hand, a Non-functional requirement is concerned with criteria that can be used to assess a system's overall operation rather than just specific behaviours. There is a broad interest in applying a varied model for the representation technique to represent the requirement visualization, especially in NFR, since it represents the quality of the requirement specification such as security, usability, availability, and so on. However, one of the main problems in requirement engineering is there remains no consensus on the nature and representation of Non-functional requirements nor how to manage them [2].

Table 1: *NFR representation techniques.*

Ref.	Technique	Objective	Result
[3]	Standard template	To defines and recommends using the software requirement	IEEE standard 830-1998
[5]	Goal-oriented modeling	To represent various types of NFR	The framework for representing NFR
[6]	Goal-oriented modeling	Proposed the i*, a well-known framework for an NFR representation model	i* framework
[7]	Ontology	Proposed ElicitO, a tool used for requirement-elicitation activities	ElicitO tool
[8]	Ontology	Proposed a method for checking SRS quality	Requirement Ontology
[9]	UML and UML extension	applying Aspect-Oriented Software Development (AOSD) to software quality requirements	REASQ model
[10]	NFR Extraction	introduced requirement frame structures, and a language to capture NFRs	The method for verifying NFR
[11]	NFR Extraction	Present a technique for using semantic model to understand web or desktop application quality measurements	Software Quality Product Attributes application (SWQAs)
[12]	NFR Extraction	Proposed a rule-based natural language technique for discovering and classifying the requirements	An automated NL requirements classifier
[13]	NFR Extraction	Introduced the framework to identify and analyze Non-functional requirements from text using pattern-based rules	The approach to identify NFRs based on extracting multiple features

1.1 Non-functional requirement representation

Table 1 shows several pieces of research that explore software requirement representations. The table presents the approach to NFR representation surveyed by this paper, which consists of a standard template, goal-oriented modelling, ontology, UML and UML extensions, and NFR extraction.

The first approach to representing NFR in our survey is the standard template. This approach takes its cues from 'IEEE standard 830-1998' "IEEE Recommended Practice for Software Requirements Specifications" (1998) [3]. It is the standard that defines and recommends using the software requirement specification (SRS) template. However, many of the items in the SRS template are meant to separate NFR from functional requirement – an endeavour that causes system analysts difficulty in understanding the relations between NFR and functional requirements - especially the relations between an NFR and other NFRs. In attempts to resolve this problem, various techniques for the representation of NFRs have been propose in order to make NFRs more accessible [4].

Because the NFR understandability is a problem for the first approach, therefore an alternative method for representing NFRs is through 'goal-oriented modelling.' The comprehensive framework proposed by Mylopoulos et al. [5], is a technique for NFR representation which is used during the software engineering process. Mylopoulos et al. introduced five components for NFR representation in the form of 'interrelated goals.' The objective for using them is that they confer the ability to refine and evaluate the software requirements while determining the degree to which a set of NFRs is supported by a particular design. However, Mylopoulos et al. accepted that their framework required further development to deal with the various types of NFRs. Theoretical fundamentals are needed for reasoning through and representing the NFRs. Moreover, semantics is a crucial notion which should be incorporated into this framework along with NFRs and efficient algorithms.

Another work in 'goal-oriented modelling' approach, L. Chung et al. mentioned that even though requirement engineering determined the requirements in case of a functional requirement or NFR, still, many requirement specifications or modeling languages need a practical process for specifying quality - even though the usual approach of goal-oriented requirement engineering is to scrutinize the importance of quality characteristics as a whole [4]. In addition, the existing NFR framework takes the high level of abstraction of the Non-functional requirements for all of the whole system into consideration. It sets the conflicts in NFRs as the main task in system development.

Additionally, i* is a well-known framework for an NFR representation model - when used together with a goal-oriented method [6]. This framework is adopted and used by many research groups and may apply in accordance with their particular concerns. In software development, various uses of representation techniques are being investigated as interesting. Conceptualization as in the i* framework is a way of understanding a level of abstraction of representation taken from the real world. In addition, various representation techniques reason logically and can use as target domains of interest. However, there is also much discussion about how they can apply in order to enhance the requirement engineering process especially at the level of sematic.

Ontology is another NFR representation. This approach variously uses in many works for requirement engineering, such as by Elicito [7]. Their work used ontology as a representation model for NFRs using ElicitO, a tool used for requirement-elicitation activities. It aims to provide software analysts with a knowledge-based repository by capturing precise NFRs through elicitation interviews. This work approaches the issue using functional and Non-functional domain ontologies (quality ontologies), using them to support elicitation activities. This tool helps analysts by giving them computerized aids to structure elicitation-interviews, suggesting that analysts use important quality aspects relevant to a particular class of applications, and provide precise requirements based on criteria available in quality model standards for software development. Dang et al. have also introduced a requirement ontology for the requirement-analysis phase of specific domains. The semantic hierarchy of the functions, and the relationship with entities such as complementary functions, and exclusion functions, are important model structures within their proposed ontology. Because of that, they also proposed a method for checking SRS quality, particularly in cases in which they want to confirm completeness and correctness while employing their ontology [8].

Another approach to NFR representation is a technique that uses UML, and UML extensions, to describe both functional and Non-functional requirements. In this case, NFRs can be considered an inherent aspect of the system. Research by Castillo et al., showed them applying Aspect-Oriented Software Development (AOSD), (ISO/IEC25030), to software quality requirements together with classic requirements of engineering notions to analyze the requirement engineering process [9]. Their work culminated in a conceptual model known as REASQ (REquirements, Aspects and Software Quality), and incorporates UML notation. For this research, all the modeling concepts (aspect orientation, software quality and a classic requirement engineering notation) are integrated into the related ontologies to visualize quality requirements in requirement engineering.

Additionally, another serious problem in NFR representation is requirement verification. To verify the NFR in the software requirement specification, Yuuma et al. introduced an NFR specification verification technique and applied it to usability and time-response requirements using a 'requirement frame model' [10]. In this paper, the author introduced requirement frame structures, and a language they called X-JXDL, which they used for detecting incorrect noun types and missing but indispensable cases in NFRs. They described a method for extracting the main verb and noun from requirement specifications, and then matching them to pre-defined roles to construct a required sentence. These results in a requirement frame that can be used to fulfill both functional and non- functional software requirements.

Finally, there are many works that introduce techniques to detect and identify NFRs within the requirement specification. Some of them apply textual natural language processing techniques to identify the requirement from the input-source in the form of text. A. Kayed, et al. present a technique for using ontology to understand web or desktop application quality measurements. The result of their work was to classify the quality attribute and show that the semantics of these attributes can be condensed into a smaller set of concepts [11]. R. Vlas, et al. proposed a rule-based natural language technique for discovering and classifying the requirements in open-source development projects. In order to discover and classify these requirements, they applied the ontology-based information extracting mechanism to parse and tag them [12]. Sharma, et al. introduced the framework to identify and analyze Non-functional requirements from text using pattern-based rules at the level of syntactic and semantic patterns, in order to analyze the input text and identify the Non-functional requirements and their categories as an output [13]. However, many of these works identify the need for a consensus in defining NFRs more clearly.

1.2 Software requirement conflict

"Conflict" is defined as a misunderstanding between people with opposing opinions or principles. In general, conflict is a problem that leads people to misunderstand facts, make wrong decisions, and finally even make mistakes or produce incorrect results. In order to manage a conflict in the software requirement, a conceptual model employing requirement representation is a means by which both functional and Non-functional requirements can be manifested. Since Non-functional requirements may change at any and all times, especially as they are evolving, they may cause requirement conflicts to pop up during any version of the specifications. Of course, this adds overhead to the engineering process. For example, it may lead to implementation errors or misunderstandings in subsystem communication. Cases in which a possible defect may exist will cause development difficulty and boost the costs for the project. Therefore, software requirement conflicts become a significant problem in software engineering [14]. NFRs tend to conflict, interfere, and contradict each other. These occurs especially in general-specific contradictions between various types of NFRs.

In this regard, the objective of this work is to offer a technique for NFR representation. This paper proposes a method for the detection of conflicts and potential conflicts using a conceptual graph. Section 2 provides details about NFR extraction and conflict detection. Section 3 describes an experimental case study and subsequent evaluation of results. Section 4 discusses the results and Section 5 offers a short conclusion.

2. NFR EXTRACTION AND CONFLICT DETECTION

In this section, we present the method to represent the conceptual model for NFRs using conceptual graph and quality attribute concept in subsection 2.1 and 2.2. Then describe in detail for the requirement conflict detecting using conceptual graph together with reasoning technique using logical inference rule evaluation in subsection 2.3.

2.1 Conceptual graph

The conceptual graph is an integrated knowledge representation joining semantic networks and existential logics as proposed by J. Sowa [15]. It can be used as a tool for supporting reasoning and automating computation. F. Harmelen, et al. explain the use of the conceptual graph by defining it as an existential logical representation. There are many examples that use conceptual graphs in development projects in order to enhance semantic model understanding [16]. For example, they are used in the analysis and design phases of relational database systems such as those described in [17] and [18]. In addition, as far as semantic-based applications go, conceptual graphs are a kind of knowledge representation that is referred to by many other works, such as those by [18], [19], and [20].

Similarly, the conceptual graph is a graph with two node types: a square node that stands for the concept, and an oval node that symbolizes the relation. Figure 1 shows an example of a conceptual graph for the following natural language statement: "An internal user who sends an online request to delete a record in the database should be authenticated by a security requirement." This example shows the use of the quality attributes portion as a template for the conceptual graph to represent NFRs as a security type. The details of using quality attributes together within the conceptual graph will be described further in section 2.3.1

2.2 Quality attribute

In the requirement analysis phase, there are both functional and NFR concerns. The NFR concerns include availability, modifiability, security, interoperability, testability, performance, and usability. In the driven design method of L. Bass, et al., their idea was to focus on NFRs that can be described in terms

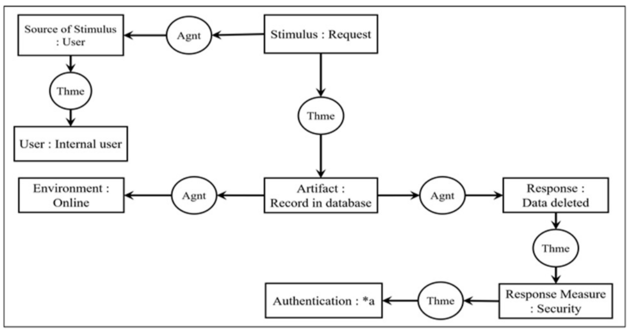


Fig.1: *The example of NFR conceptual graph.*

of a system quality attribute. They also explained a fundamental aspect of system quality attributes, and promoted structural patterns that support these quality attributes [21]. Figure 2 illustrates the characteristics of a quality attribute in terms of software architecture.

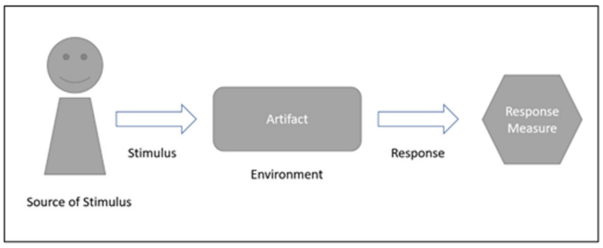


Fig.2: *Quality attribute with scenario portions.*

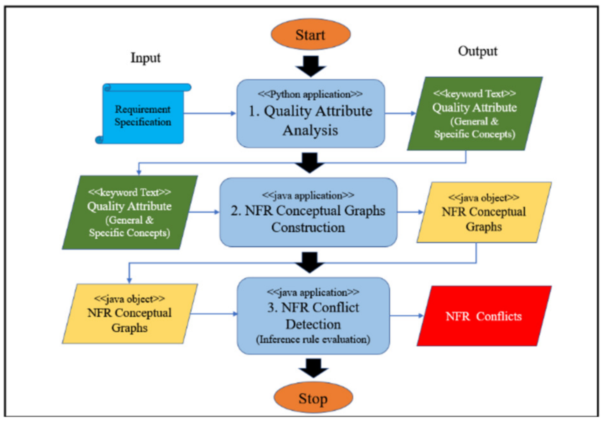
In order to define quality attributes as a testable and unambiguous feature, L. Bass, et al. applied an approach using a general form to represent them as types of software requirements. It appears to be a good way to emphasize the commonalities of all quality attributes, but it may be too rigid an approach for some cases during system development, according to [22]. According to a quality attribute, their approach divided the quality attribute into scenario portions in accordance with [22].

1. Source of stimulus - an entity that activates the stimulus.
2. Stimulus - a request that is activated for a response when it is found at a target system.
3. Environment - the condition of the state where the stimulus occurs.
4. Artifact - the thing that can be stimulated.
5. Response - the result of the activation of the stimulus.
6. Response measure - the evaluation of the response that occurred.

According to the quality attribute, it is defined as a set of general scenarios and reflected into the requirements for the system. The quality attribute requirements in this scope are composed of seven NFR types which are availability, interoperability, modifiability, performance, security, testability, and usability [21].

2.3 NFR conflict and potential conflict detection process

This section describes our proposed method for extracting NFRs using a conceptual graph, and for detecting NFR conflicts using existential logic and the rules of inferential logical. To demonstrate the proposed method, we developed a prototype called NFR conflict and potential conflict detection and have evaluated its use with a billing system requirement specification in a case study. The details of the research experiment and case study will be discussed in the next section. Figure 3 shows the three steps of our method. Quality attribute analysis. The first step begins with analysing the NFRs that may be found in requirement specifications, and extracting them into quality attribute scenario portions, as defined by [22] in the form of NFR XML frames. The NFR XML frame defines the NFR element in accordance with a quantitative model which defines general vs specific relations to separate general concepts from specific concepts.



Start
Input
Output
Requirement Specification
<<Python application>> 1. Quality Attribute Analysis
<<keyword Text>> Quality Attribute (General & Specific Concepts)
<<keyword Text>> Quality Attribute (General & Specific Concepts)
<<java application>> 2. NFR Conceptual Graphs Construction
<<java object>> NFR Conceptual Graphs
<<java object>> NFR Conceptual Graphs
<<java application>> 3. NFR Conflict Detection (Inference rule evaluation)
NFR Conflicts
Stop

Fig.3: *NFR conflict and potential conflict detection method.*

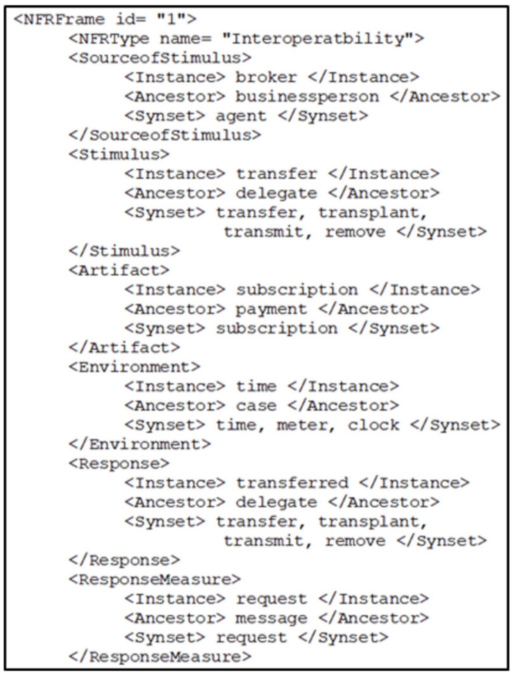
1. Conceptual graph construction. The second step is to construct a conceptual graph from the NFR XML frame using the NFR conceptual template. The NFR conceptual graph constructed in this step is a representation of NFRs extracted from the input software requirement specification.
2. NFR conflict detection. In this step, we already have the NFRs in the form of a conceptual graph. The advantage of the conceptual graph is that it can be a logical representation in itself. Applying the rules of inferential logic this automatically proves the presence of the NFR conflict within a requirement specification.

2.3.1 Quality attribute analysis

This paper proposes using quality attribute scenario portions of NFRs as a group of NFR concepts. In this regard, we defined the objective of the first step to be the extraction of the NFRs from an input document, that usually presents in the form of natural language, to an NFR XML frame, and then reassemble them in the form of an NFR conceptual template together with the NFR dictionaries which provided the NFR keywords collected from the expert of particular system. During our research, we have created an algorithm which analyses and extracts the NFRs according to the following steps:

1. It elicits requirements from the input requirement specification document, separates them into a list of sentences, then analyses them to see if they fit the subject-verb-object (SVO) pattern.
2. When considering the stimulus in portion scenario for each quality attribute, the verb of each requirement sentence is looked upon as the main element.
3. The subject of the sentence is then deemed a source of stimulus, and the object as an artifact of the NFR quality attribute.
4. For the response portion, our system checked for correspondence against all the keywords in the dictionary. If there was no available mapping of keywords to the keyword in the requirement specification, our system re-framed the stimulus in the past tense.
5. The environment portion scenario in this research is defined as the adjective or noun phrase that modifies the artifact of the requirement specification.
6. The response measure arose when a keyword existed in the NFR dictionary. Our system maps the requirement specification keyword to the keyword defined in the NFR dictionary.
7. We have defined the NFR examples as being one of seven types according to the quality attribute types of Len Bass [21]. They are described as partaking of one of the following modalities: availability, modifiability, security, testability, performance, usability and interoperability. The NFR type of each frame was classified according to the number of matching keywords in each frame as well as the keyword defined in the NFR dictionary. The criteria used to justify the NFR type assigned to each NFR frame was the greatest number of matching keywords in the NFR frame and the NFR dictionary.

In order to extract the NFR from the input document, we needed to design the NFR requirement frame as an XML structure. It is a data structure that can represent the NFR in form of the quality attribute proportion model of [22]. An example of the NFR frame shows in figure 4. The example presents the NFR structure's steps in the following order: system, module, NFR frame, NFR type, source of stimulus, stimulus, artifact, environment, response and response measure. The main objective of our work is to devise a prototype system that read the requirement document to extract the output in the form of NFR Frames and prepare them as inputs for the next step in the process. We describe the experiment's design in the next section.



```
<NFRFrame id= "1">
     <NFRType name= "Interoperatbility">
     <SourceofStimulus>
          <Instance> broker </Instance>
          <Ancestor> businessperson </Ancestor>
          <Synset> agent </Synset>
     </SourceofStimulus>
     <Stimulus>
          <Instance> transfer </Instance>
          <Ancestor> delegate </Ancestor>
          <Synset> transfer, transplant,
                   transmit, remove </Synset>
     </Stimulus>
     <Artifact>
          <Instance> subscription </Instance>
          <Ancestor> payment </Ancestor>
          <Synset> subscription </Synset>
     </Artifact>
     <Environment>
          <Instance> time </Instance>
          <Ancestor> case </Ancestor>
          <Synset> time, meter, clock </Synset>
     </Environment>
     <Response>
          <Instance> transferred </Instance>
          <Ancestor> delegate </Ancestor>
          <Synset> transfer, transplant,
                   transmit, remove </Synset>
     </Response>
     <ResponseMeasure>
          <Instance> request </Instance>
          <Ancestor> message </Ancestor>
          <Synset> request </Synset>
     </ResponseMeasure>
```

Fig.4: *Example of NFR XML frame.*

2.3.2 Conceptual graph construction

We divided quality attributes into six portions. According to our conceptual graph, six NFR concepts best represent the requirement concept and their relations at general and specific levels. An example of an NFR conceptual graph is given in Figure 5.

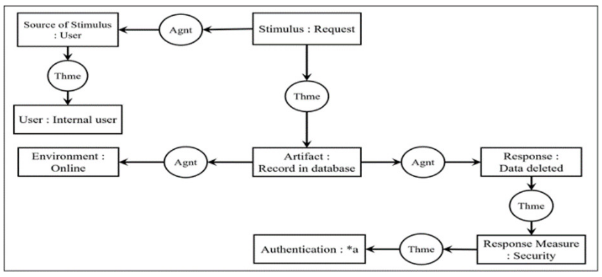


Fig.5: *Example of NFR conceptual graph.*

This paper designed an NFR template composed of six NFR concepts to represent six general scenario portions of the NFR. Each NFR concept in the six concepts template has a type and name to represent its semantic aspects. The NFR type is defined as the NFR conceptual template in order to represent the six scenario portions of the quality attribute of the NFR. However, the concept name is a concept reference that represents the instance of each concept. For the conceptual graph, the acceptable values of the type can be derived from the existential logic to represent the instance of a concept.

In addition, to construct the NFR conceptual graph, the work uses the NFR conceptual graph template as a pre-defined structure for setting up the NFR concept type and the NFR concept name that is defined in the NFR dictionary for each NFR type. The input software requirement specifications are that it consists of text-based documentation. For example: R1 is a security requirement of the system:

R1: The internal user who sends the online request to delete a record in the database should be authenticated as part of the security requirement.

Figure 5 shows the security requirement represented by the conceptual graph with the pre-defined NFR conceptual graph template to represent the NFR. The example shows that the source of stimulus in this case is the 'user', the stimulus is the 'request', the artifact is the 'record in the database', with the defined environment being 'online'. The response for this case will be: 'data deleted', and the response measure is the 'security authentication'. In addition, the example also shows that the user and the response measure may have modifiers consisting of an internal user and an authentication method. As regards conflict detection, more details of the general-specific relation are given below.

2.3.3 NFR Conflict detection

For the NFR conflict detection, we want to emphasize NFR representation with NFR quality attribute visualization by applying a conceptual graph. Because the conceptual graph is also a form of semantic model, NFRs can verify their completeness by invoking the corresponding completeness of the general scenario portion.

In order to prove the conflict between two or more NFRs, our conflict detection approach applies existential logic and the rules of inference. The concept we invoke is the quantifier inference rule, composed of four sub-rules that can be proved independently. The quantifier inference rules are defined as follows:

1. Universal Instantiation - the inference from a proposition stating that all things are thus and so to an instance, stating that some particular is thus and so. In classical formal logic, it is also known as universal quantifier elimination [23]. The universal instantiation statement shows as statement one.

$$\frac{\forall x P(x)}{\therefore P(c)} \tag{1}$$

2. Universal Generalization - an inference rule to promote the generalization for all things in a participate set. The quantified statement shows as statement two.

$$\frac{P(c)\ for\ an\ arbitary\ c}{\therefore \forall x P(x)} \tag{2}$$

3. Existential Generalization – this permits you to introduce an existential quantifier and is sometimes called an 'existential introduction'. It allows you to infer an existential generalization (a ∃ sentence) from any instance of that generalization, as shows in statement 3.

$$\frac{P(c)\ for\ an\ arbitary\ c}{\therefore \exists x P(x)} \tag{3}$$

4. Existential Instantiation - the last rule used in this paper is existential instantiation. Suppose we can prove B from a particular proposition 'Fa' and confirm that 'a' is not mentioned in any premises used in this argument. So, it is as if 'a' is an arbitrary example. The rule allows to infer B from the weaker premise that (∃x)Fx. Provided that something is F, there is going to be something that could function as a counter, just as 'a' does. Blackburn (2008). The existential instantiation shows in statement four.

$$\frac{\exists x P(x)}{\therefore P(c) for\ som\ element\ c} \tag{4}$$

The current paper uses these four rules for the detection of any potential NFR conflicts that may occur in the requirement specification document. In so doing, we extract the NFR using an NFR XML Frame and an NFR dictionary, as described in the previous stepsp7/57-58. Here, in the detail of the detection step, we define the entity and its structure in the billing system in the form of billing system vocabulary and a hierarchy of its vocabulary, i.e. its keywords. This structure represents the relations between the entities used by the NFR conflicts and potential conflict detection to determine the general and specific relation between the comparative NFRs and detects the conflict and the potential conflict which may occur between them. An example of the entity and the hierarchical structure shows in figure 6.

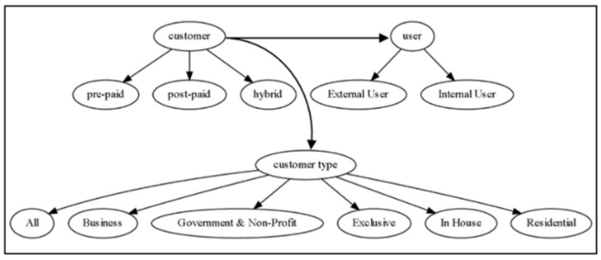


Fig.6: *Example of the entity and the hierarchy structure of the billing system.*

Figure 6 shows an example of the entity and the hierarchy structure of the billing system. It represents the semantic structure of business entities existing in the billing system of the telecommunications company used as a business case in this paper. The structure describes the semantic hierarchy which the NFR conflict and potential conflict detection can use for proving the potential conflict using existential inference rules.

3. AN EXPERIMENT AND CASE STUDY

In this research, we developed an NFR conflict and potential conflict detection prototype in order to conduct an experiment using a billing system as a research case study. The end we had in view was to demonstrate the feasibility of our proposed method. The main functions of the billing system, as named, were invoice calculation and preparation. The requirements of this system represents within the company's software specification document. This requirement specification document illustrates that the billing system can divide into 15 modules or subsystems: Rated XDR process, One-time charge, Recurring charge, Bill run, Suppression, Bill sampling, Bill Termination, Negative bill, Hot bill, Suppress, Topup, Revenue code, Override, SO tools and Housekeeping.

For each module, the requirement specification document organized the requirements as topics for the functions. This work will read the requirements topic by topic, analyse their content, extract the NFRs, then process them as inputs for the NFR conflicts and detection of potential conflicts. The experimental results and evaluation of our system are discussed in the next subsection.

3.1 Experimental results and evaluation

This section describes details from the NFR detection experiments. We began with the NFR classification process that focuses on the mechanism to extract the NFRs from the requirement specification document to the NFR XML frame, then reconstructs them as an NFR conceptual graph, and then processes the NFR conflict and potential conflict detection to detect our systems' outputs. The output is represented by the complete NFR XML frames, the incomplete NFR XML frames, the NFR potential (bad smell) conflicts, and the NFR conflicts.

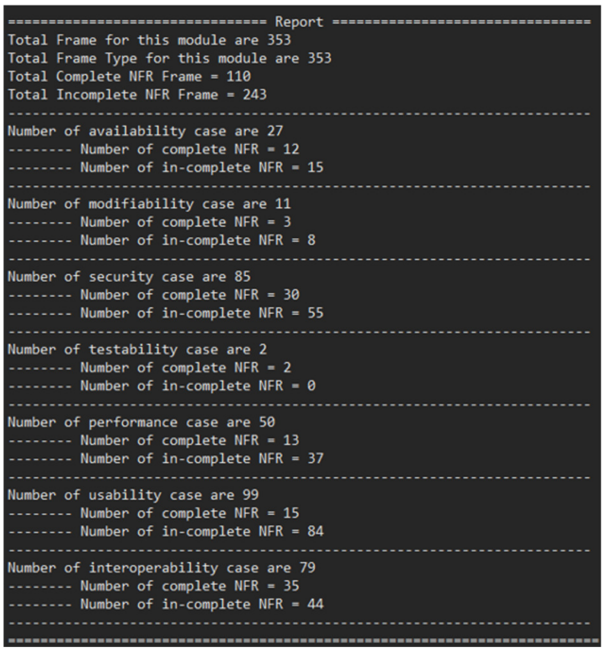
3.1.1 The experiment results

The results of the experiment show in figure 7. The system report shows the output data as follows:

1. Total frames for the system module. One of the outputs is the total frames for the system module. It is equivalent to the number of NFR XML frames extracted from the software requirement specification document. For each NFR XML frame, this work defined a frame structure as the data structure of the NFR which applied the quality attribute scenario portions to represent a knowledge graph. These knowledge graphs will be an input to the conceptual graph construction process and the NFR conflict and potential conflict detection process of the pro-

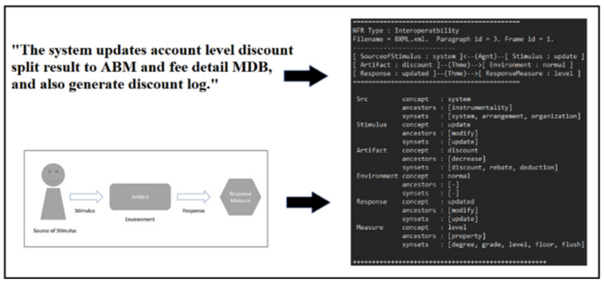
posed method. Figure 8 presents the example of the complete NFR XML frame.

2. Total complete NFR XML frames. The complete NFR frames are one of the main outputs. They are equivalent to the NFR XML frames which fully possess the six quality attribute portions based on the seven NFR dictionaries for all types of NFRs. According to Figure 7, the billing system requirement document for all the 15 modules is given by the user in the telecommunication company, and the system extracts the NFR from its content. From the input document, the system found 110 complete NFR XML frames. It shows that many NFRs can be found and extracted. However, there are also a number of NFR XML frames that could not fill the quality attribute portions.



```
================================ Report ================================
Total Frame for this module are 353
Total Frame Type for this module are 353
Total Complete NFR Frame = 110
Total Incomplete NFR Frame = 243
------------------------------------------------------------------------
Number of availability case are 27
-------- Number of complete NFR = 12
-------- Number of in-complete NFR = 15
------------------------------------------------------------------------
Number of modifiability case are 11
-------- Number of complete NFR = 3
-------- Number of in-complete NFR = 8
------------------------------------------------------------------------
Number of security case are 85
-------- Number of complete NFR = 30
-------- Number of in-complete NFR = 55
------------------------------------------------------------------------
Number of testability case are 2
-------- Number of complete NFR = 2
-------- Number of in-complete NFR = 0
------------------------------------------------------------------------
Number of performance case are 50
-------- Number of complete NFR = 13
-------- Number of in-complete NFR = 37
------------------------------------------------------------------------
Number of usability case are 99
-------- Number of complete NFR = 15
-------- Number of in-complete NFR = 84
------------------------------------------------------------------------
Number of interoperability case are 79
-------- Number of complete NFR = 35
-------- Number of in-complete NFR = 44
------------------------------------------------------------------------
========================================================================
```

Fig.7: *The overview report of the experiment results.*



"The system updates account level discount split result to ABM and fee detail MDB, and also generate discount log."

```
--------------------------------------------
NFR Type : Interoperatbility
Filename = 8XML.xml.  Paragraph id = 3. Frame id = 1.
----------------------------
[ SourceofStimulus : system ]<--(Agnt)--[ Stimulus : update ]
[ Artifact : discount ]--(Thme)-->[ Environment : normal ]
[ Response : updated ]--(Thme)-->[ ResponseMeasure : level ]
============================================

 Src         concept   : system
             ancestors : [instrumentality]
             synsets   : [system, arrangement, organization]
 Stimulus    concept   : update
             ancestors : [modify]
             synsets   : [update]
 Artifact    concept   : discount
             ancestors : [decrease]
             synsets   : [discount, rebate, deduction]
 Environment concept   : normal
             ancestors : [-]
             synsets   : [-]
 Response    concept   : updated
             ancestors : [modify]
             synsets   : [update]
 Measure     concept   : level
             ancestors : [property]
             synsets   : [degree, grade, level, floor, flush]

++++++++++++++++++++++++++++++++++++++++++++++
```

Fig.8: *The complete NFR XML frame for the NFR in software specification document.*

3. Total incomplete NFR XML frames. Incomplete NFR frames is one of the most important outputs of our system. It counts the number of frames in which the system cannot capture all six quality attribute portions based on the seven NFR dictionaries for all types of NFRs. These incomplete NFR XML frames are important because they suggest that there NFRs will potentially occur in the software specification document but that these potential NFRs are still incomplete. It is a good sign for software analysts or requirement engineers who would like to take action on potential NFRs to complete the developing system's software specification document requirements.

4. The number of NFR XML frames for each quality attribute type. One of the main objectives of this paper is to classify the software requirement specification for each NFR type. Our proposed method and implemented system, as described in section 3, used the software requirement specification document as an input for the detection process. Outputs were then analysed and classified with the NFR dictionary by mapping each keyword with one of the keywords gathered from the requirement specification document. The number of keyword mappings was used to classify the NFR type by incorporating a voting technique for all of the NFR XML frames that we extracted from the input document. Figure 7 shows the outputs of the newly classified NFR types. According to figure 7, the NFR conflict and potential conflict detection gets the billing system requirement document from the user in the telecommunication company for all the 15 modules, and then extracts the NFR from the content from all of the requirement specifications. Then, the NFR conflict and potential conflict detection process found that within this document, there were 353 NFR frames. These divide into 110 complete NFR XML frames and 243 incomplete NFR XML frames.

5. The results of NFR conflicts and potential (bad smell) NFR conflicts. One of the main objectives of this paper is not only to detect the conflicts but also the potential conflicts that may appear in the software requirement specification. As we have described in our proposed method in section 2.3, it reads the requirements in the input document, extracts the NFR XML frames, and constructs the NFR conceptual graph using the proposed NFR template. Then, in order to analyse and prove the conflict using existential logic quantitative inference rules it matches all of the constructed NFR conceptual graphs with each other to detect the general-specific relation of the quality attribute portions between the pairs of NFR XML frames. If the implemented system found that all of the quality attribute portions in a pair of NFR XML frames have a general-specific relation, then it justifies that these pair of NFR

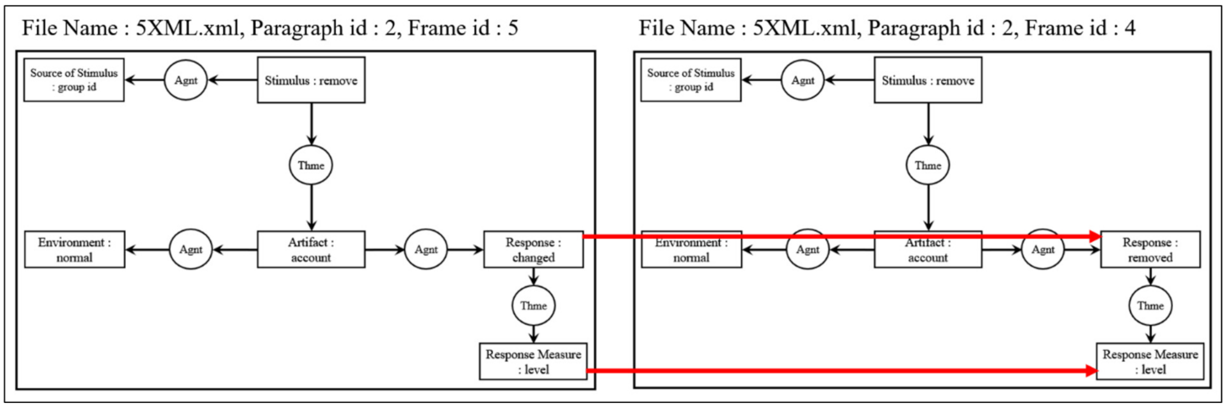

File Name : 5XML.xml, Paragraph id : 2, Frame id : 5
Source of Stimulus : group id
Agnt
Stimulus : remove
Thme
Environment : normal
Artifact : account
Response : changed
Response Measure : level
File Name : 5XML.xml, Paragraph id : 2, Frame id : 4
Response : removed

Fig.9: *The NFR conceptual graph conflict detected in the billing system .*

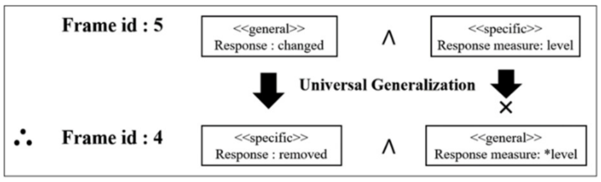

Frame id : 5
<<general>> Response : changed
<<specific>> Response measure: level
Universal Generalization
Frame id : 4
<<specific>> Response : removed
<<general>> Response measure: *level

Fig.10: *The NFR conceptual graph conflict in case of universal generalization.*

frames is a match with the NFR XML frame scenario. It then presents these pairs as possessing the exact software requirements. In this billing system case study, 20 matching NFR XML frames were found within the requirement specification document.

From the matching NFR XML frames, the NFR conflict and potential conflict detection checked all pairs of the matching NFR XML frames using the quantitative inference rules at the level of existential logic to prove whether there was a conflict between these matching NFR XML frames. The results of the NFR conflict and potential conflict detection shows in Figures 9 and 10.

In addition, during the processing of the NFR conflicts and the potential conflict detection, it is possible that the matching NFR XML frames may conflict – however, it cannot describe them. These scenarios in which there are potential but indescribable NFR conflicts are termed 'bad smell' cases. In this paper, we prove these potential conflicts exist by using three quantitative inference rules: universal instantiation, existential instantiation and existential generalization, as explained in section 2.3.3.

As Figure 11 shows, the NFR potential conflict detection gets the billing system requirement document from the user for all the 15 modules and extracts the NFR from the content from all of the requirement specification. Then, the NFR conflict and potential conflict detection found the 'bad smell' requirements occurred in two quantitative inference rule cases. The output of this experiment shows the 'bad smell' requirement occurred in the NFR XML output file named "10XML.XML" in paragraph number 2 at frame id number 2, and paragraph number 4 at frame id number 4, in which the output points out the requirement sentence below. The bill system requirement in file 10XML.XML, paragraph number 2 at frame id number 2 said.

"If CRM sends the request to change subscriber status to suspend un-identified, also send one special promotion which leads to 100% discount to all of the charge."

Meanwhile, the bill system requirement in file 10XML.XML, paragraph number 4 at frame id number 4 said:

"When SFF sends the request to change customer status to suspend un-identified, BSS broker will subscribe a special promotion of an automatic 100% discount of billing time."

Figure 11 and Figure 12 show the NFR potential conflicts captured in this case are proved by the existential instantiation inference rule. It is because the source of the stimulus concept keyword "CRM", as defined in the usability dictionary, and its synset, in case frame id number 2, and the stimulus concept name "SFF", in frame number 4, are both considered to be within the specific keyword scope. This pair of stimulus concepts cannot be instantiated instead by each other. Therefore, the NFR conflict and potential conflict detection, can be reported within the existential instantiation conflict case.

Figure 13 and Figure 14 show the NFR potential conflict captured in this case as proved by the existential generalization inference rule, since the response measure concept keyword "discount" as defined in usability dictionary and its synset in case frame id number 2 to be considered as a specific scope meanwhile response measure concept also name "discount" in frame number 4 are contradict considered to general keyword scope because it is declared ambiguously in the semantic entity hierarchy of the billing system. Therefore, NFR conflicts and potential conflicts were detected in this case, and reported according to the rules of existential generalization.

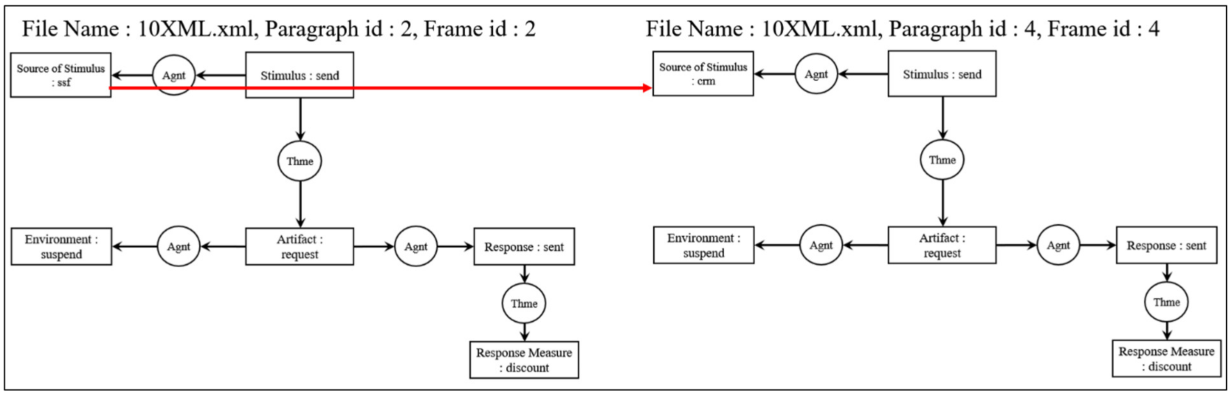


Fig.11: *NFR potential conflict (bad smell) in case of existential instantiation.*

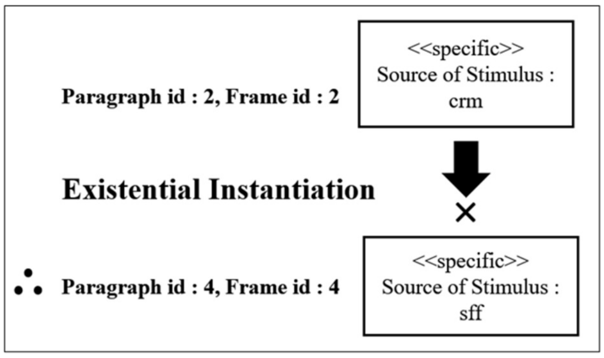

<<specific>> Source of Stimulus : crm
Paragraph id : 2, Frame id : 2
Existential Instantiation
Paragraph id : 4, Frame id : 4
<<specific>> Source of Stimulus : sff

Fig.12: *A prove of NFR potential conflict (bad smell) in case of existential instantiation.*

3.1.2 Evaluation of results

The experiment results in this work are the output of the implemented system for the proposed method of this paper.

The results reported in this paper come from the experiments we have implemented using our proposed method and system.

The reported output represents in the form of the number of NFRs that the system extracted from the software requirement specification document. It is revealed in various cases of NFRs and NFR types, such as in the complete NFR XML frame. The NFR extracting process output shows in Table 2.

The results of the extracting process reveal NFR conflicts and potential conflict detection. The system could extract 353 NFR XML frames from the input document. These frames separate into complete NFR XML frames, of which there were 110 frames, and incomplete NFR XML frames, of which there were 243 frames. As we proposed when we defined the scope of this research, these output frames categorize as one of seven NFR types (availability, modifiability, security, testability, performance, usability and interoperability). Each of these types can separate into complete NFR XML frames and incomplete NFR XML frames, with corresponding percentages of occurrence as shown in Table 2. The output of NFR identification presented in Table 3 indicated the corrected identified NFRs from the proposed method compared to the correct requirement getting from the expert of development team in our case study.

Table 2: *The output of NFR extracting.*

NFR type	Total frame	Complete frame	Incomplete frame	%Complete frame
All NFR Type	**323**	**110**	**243**	**31.16**
Availability	27	12	15	44.44
Modifiability	11	3	8	27.27
Security	85	30	55	35.29
Testability	2	2	0	100
Performance	50	13	37	26
Usability	99	15	84	15.15
Interoperability	79	35	44	44.30

Table 3: *The output of NFR identification.*

NFR type	Requirement from the expert	Requirement from the prototype	Correct identified requirement
All NFR Type	**120**	**353**	**110**
Availability	18	27	12
Modifiability	4	11	3
Security	31	85	30
Testability	2	2	2
Performance	15	50	13
Usability	15	99	15
Interoperability	35	79	35

In addition, we can determine the efficiency and the effectiveness of the NFR conflicts. Potential conflict detection is reflected in the accuracy of the number of NFR complete frames extracted from the input software requirement specification document. Table 4 represents the accuracy of the NFR potential extraction by deriving the values of precision, recall and F measures from the total frames extracted from the input document, as well as the details concerning each NFR type. Table 4 indicates that the total number of NFR XML frames extracted from the software specification document was 353 frames, and we define valid frames as including: complete NFRs, incomplete NFR frames, and invalid NFR frames. Table 4 also shows that the total NFR XML frames had a precision rate of 0.36, a recall rate of 0.92 and an

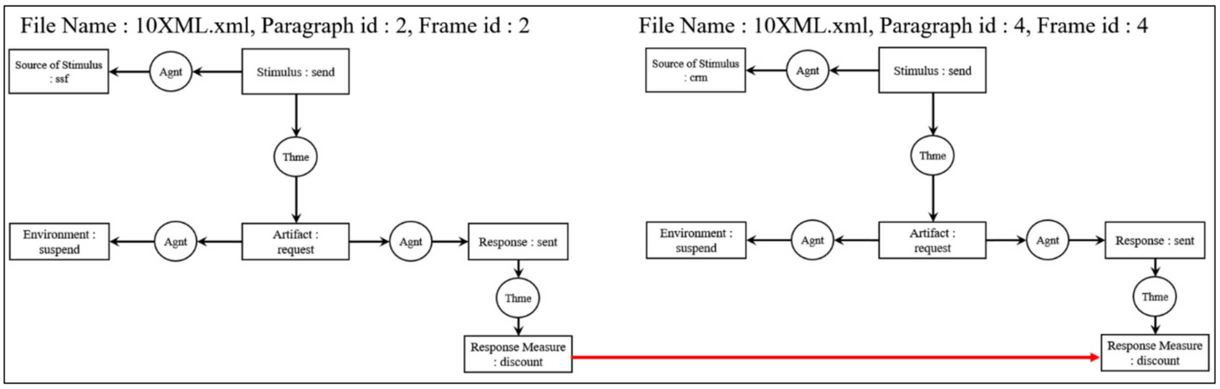


Fig.13: *NFR potential conflict (bad smell) in case of existential generalization.*

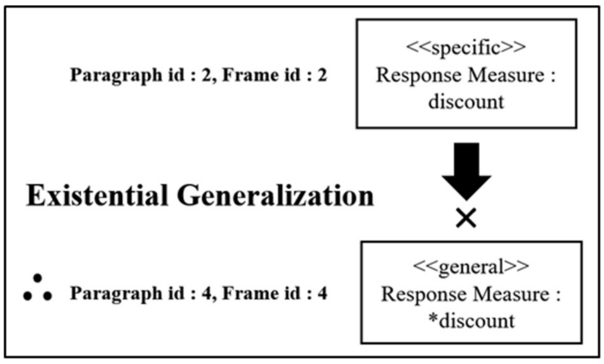

Paragraph id : 2, Frame id : 2
<<specific>> Response Measure : discount
Existential Generalization
∴ Paragraph id : 4, Frame id : 4
<<general>> Response Measure : *discount

Fig.14: *A prove of NFR potential conflict (bad smell) in case of existential generalization.*

F measure of 0.52. Moreover, from the billing case study, we found that the intersection of availability and interoperability provided the best opportunities to extract valid NFR XML frames and potential NFR XML frames from the input requirement specification document.

Table 4: *The accuracy of NFR extracting.*

NFR type	Precision	Recall	F-Measure
All NFR Type	**0.36**	**0.92**	**0.52**
Availability	0.44	0.57	0.49
Modifiability	0.27	0.75	0.39
Security	0.35	0.97	0.51
Testability	1	1	1
Performance	0.26	0.87	0.40
Usability	0.15	1	0.26
Interoperability	0.44	1	0.61

Finally, the experimental results of this work are the outcome of the system we implemented in accordance with the method proposed in this paper. The main output of the proposed method represents in the form of the number of conflicts detected from the software requirement specification document. The results of the NFR conflict and potential conflict detection shows in Table 5.

The experiment results in this research are the output of the implemented system for the proposed method of this work. The output shows that the NFR conflict and potential conflict detection analysed that from the input requirement specification document, there are 20 NFR XML frames that can be a candidate of the NFR conflict using the quality attribute technique for constructing the extracted NFR into NFR conceptual graph to detect the conflict that may occur between them. The detail of these technique had described in section 2.3. When the system assessed case by case the quantitative inference rules in existential logic, the system found that there were 18 cases that NFR considers the same situation as the billing system requirement but cannot prove to be a conflict. However, the results show that there is a conflict in the case of interoperability types for two NFRs, proved by universal generalization which need to be reported to the software development team and let them be the overheads for the software development life cycle. Moreover, the implemented system also shows that it can capture the potential conflict or the ‘bad smell’ NFR in case of Existential generalization and Existential instantiation respectively.

Table 5: *The result of conflict detection.*

Case of conflict	Number of conflicted NFRs	Inference rules
NFR conflict	2	Universal generalization
NFR potential conflict	2	Existential generalization
NFR potential conflict	2	Existential instantiation

4. DISCUSSION

This paper proposed a method for extracting Non-functional requirements (NFRs) from the software requirement specification document, and to detect conflicts that may occur between the NFRs.

The experimental study entailed in the billing system case demonstrated that amongst the NFR XML frames extracted from the software specification document, there were 353 valid frames including complete NFRs, incomplete NFR frames, and Invalid NFR frames. We also found that the total NFR XML frames demonstrated precision of 0.36, recall of 0.92, and an F measure of 0.52.

Our experimental results revealed that it is possible to increase accuracy in precision and recall. According to the NFR extracting process, for all cases of NFR types, the most important resources for the extracting mechanism are the NFR dictionaries, this work, we used seven dictionaries for the 7 NFR types. It is because this research requires the NFR dictionaries from the target organization. They ,in turn, are in the development phase process of collecting all the billing system keywords which can subsequently give more coverage of the billing keywords after further revisions of the NFR dictionary. Therefore, it should be possible to improve the accuracy rate of our extracting process as we had presented with ever newer versions of NFR dictionaries.

For the NFR conflict and potential conflict detection, the system extracts the NFR for all seven types of NFR cases. We found that within the requirement specification document, there were 20 NFRs could be in conflict with each other. Nevertheless, the system also proved that these 20 NFRs could be potentially designated as conflicting NFRs. We have shown that implementing our approach captures the NFR conflict for the 20 NFRs and detected two NFRs as definitely in conflict, and reported two potentially conflicting NFRs.

From the result of the NFR conflict and potential conflict detection, we found that the main factors required to improve the accuracy of the implemented system are: the keyword defined in the entity hierarchy of the billing system, the ancestral keywords, and the synonyms from the synset which the system queried from the wordnet dataset. It is possible that the keyword defined in the billing hierarchy may be derived from proprietary domains and thus may not be consistent with the semantics that we found in the other dataset. In that case, a process to check the consistency between these datasets should be developed to improve conflict-detection accuracy rates. The specific template and particular dictionary in the other domains could be left for future work.

5. CONCLUSIONS

This paper has proposed a technique using a conceptual graph for NFR representation and has further proposed a method for NFR conflict detection in software requirement specification documents. Since NFRs are so complex, no consensus has been reached on defining them. We introduce the NFR XML frame as a formal structure that, during the extraction, can represent the NFRs drawing from quality attributes and the use of an NFR dictionary to provide the keywords. For NFR conflict detection, our conceptual graph can be used as a model that can represent the NFRs in terms of semantic modeling and the rules of quantitative logic. This work also proposes an NFR conceptual graph template which can give rise to NFR conceptual graphs, based on their semantic quality attribute portions. As well they can represent the instance of the concept to reflect the quantitative logic concern. To evaluate the feasibility and efficiency of the proposed method, this work developed a prototype system which uses Java and Python on a billing system requirement specification document as the case study. The case study results indicated that there was a total of 353 NFR XML frames extracted from the software specification document. Of which 353 were valid frames consisting of complete NFRs, incomplete NFR frames, and invalid NFR frames. The accuracy of the extraction process was measured to be: the precision was 0.36, the recall was 0.92 and the F measure was 0.52. Moreover, these NFR XML frames can be detected as candidate NFR conflicts for 20 NFRs, and two NFRs were detected as an NFR conflict, and two NFRs reported as potential conflicts or ‘bad smell’ NFRs.

In addition, this work found that the top three NFR portions that had missing values are environment, response measure, and response. Therefore, these NFR portions should be more focused, thus better completing the value of the NFR portion when developing the software requirement specification. Finally, this work presented the possibility of detecting the conflict in the requirement engineering domain. It is also possible that this approach could be applied to further works in the future by developing more specific templates and dictionaries to be used to new target domains.

References

[1] Z. Jin, “Requirements engineering methodologies,” *Environment modeling-based requirements engineering for software intensive systems*, Morgan Kaufmann, 2018, ch. 2.

[2] M. Glinz., “On Non-functional requirements,” *15th IEEE International Requirements Engineering Conference*, pp. 21-26, 2007.

[3] “IEEE Recommended Practice for Software Requirements Specifications,” in *IEEE Std 830-1998*, pp.1-40, 1998.

[4] Chung L., do Prado Leite J.C.S., “On Non-functional requirements in Software Engineering,” in *Borgida A.T., Chaudhri V.K., Giorgini P., Yu E.S. (eds) Conceptual Modeling: Foundations and Applications*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg. Germany, Vol. 5600, 2009.

[5] J. Mylopoulos, L. Chung and B. Nixon, “Representing and using Non-functional requirements: a process-oriented approach,” *IEEE Transactions on Software Engineering*, vol. 18, no.6, pp. 483-497, 1992.

[6] X. Franch, L. López, C. Cares, and D. Colomer, “The i* Framework for Goal-Oriented Model-

ing," in *Karagiannis D., Mayr H., Mylopoulos J. (eds) Domain-Specific Conceptual Modeling*, Springer, Cham. Switzerland, 2016.

[7] T. H. Al Balushi, P. R. F. Sampaio, D. Dabhi, and P. Loucopoulos, "ElicitO: A Quality Ontology-Guided NFR Elicitation Tool," in *P. Sawyer, B. Paech, P. Heymans, (eds) Requirements Engineering: Foundation for Software Quality. REFSQ 2007.* Lecture Notes in Computer Science, Springer, Berlin, Heidelberg. Germany, Vol. 4542, 2007.

[8] D. V. Dzun and A. Ohnishi, "Improvement of Quality of Software Requirements with Requirements Ontology," *Proceedings of the Ninth International Conference on Quality Software*, pp. 284-289, 2009.

[9] I. Castillo, F. Losavio, A. Matteo, and J. Boegh, "REquirements, aspects and software quality: The REASQ model," *Journal of Object Technology*, vol. 9, pp. 69-91, 2010.

[10] Y. Matsumoto, S. Shirai, and A. Ohnishi, "A Method for Verifying Non-functional requirements," *Procedia Computer Science*, vol. 112, pp. 157-166, 2017.

[11] A. Kayed, N. Hirzalla, A. A. Samhan and M. Alfayoumi, "Towards an Ontology for Software Product Quality Attributes," *2009 Fourth International Conference on Internet and Web Applications and Services*, pp. 200-204, 2009.

[12] R. Vlas and W. N. Robinson, "A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects," *2011 44th Hawaii International Conference on System Sciences*, pp. 1-10, 2011.

[13] V. S. Sharma, R. R. Ramnani, and S. Sengupta, "A framework for identifying and analyzing Non-functional requirements from text," *The 4th International Workshop on Twin Peaks of Requirements and Architecture (TwinPeaks 2014)*, Association for Computing Machinery, New York, NY, USA, pp. 1–8, 2014.

[14] D. Mairiza, D. Zowghi, & N. Nurmuliani, "Managing conflicts among Non-functional requirements," *12th Australian Workshop on Requirements Engineering*, pp. 11–19, 2009.

[15] J. F. Sowa, *Conceptual Structures Information Processing in Mind and Machine*, Addison-Wesley Publishing company, Inc. Boston, United States, 1984.

[16] F. van Harmelen, V. Lifschitz, and B. Porter, *Handbook of Knowledge Representation*, Elsevier Science. San Diego, United States, 2007.

[17] J. F. Sowa, "Conceptual Graphs for a Data Base Interface," *IBM Journal of Research and Development*, vol. 20, no.4, pp. 336-357, 1976.

[18] J. Zhong, H. Zhu, J. Li, and Y. Yu, "Conceptual graph matching for semantic search," *Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces*, pp. 92–196, 2002.

[19] L.W. Harper and H.S. Delugach, "Using conceptual graphs to capture semantics of agent communication," *Conceptual Structures for Knowledge Creation and Communication, 11th International Conference on Conceptual Structures*, pp. 393-404, 2003.

[20] R. Dieng-Kuntz and O. Corby, "Conceptual graphs for semantic web applications," *Conceptual Structures: Common Semantics for Sharing Knowledge, 13th International Conference on Conceptual Structures*, pp. 19–50, 2005.

[21] L. Bass, M. Klein, & F. Bachmann, *Quality attribute design primitives and the attribute driven design method*, vol. 2290, p. 323-328. 2002.

[22] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice (3th. ed.)*, Addison-Wesley Publishing company, Inc., Boston, United States, 2012.

[23] S. Blackburn, *The oxford dictionary of philosophy (2nd ed.)*.Oxford University PressPrint Publication. 2008.

Taweewat Luangwiriya is a Lecturer at Data Science and Innovation Program, College of Interdisciplinary Studies, Thammasat University. His bachelor's degree was B.Sc. (Computer Science) and M.Sc. (Computer Science) from Thammasat University, Thailand. He is also studying Ph.D. in Computer Science from Thammasat University, Thailand. His researches are about Artificial Intelligence, Requirement Engineering, Software Engineering, and Data Science. He can be contacted at email: taweewatlu@gmail.com, taweewat.l@tu.ac.th.

Rachada Kongkachandra is an Assistant Professor at Data Science and Innovation Program, College of Interdisciplinary Studies, Thammasat University. Her bachelor's degree was B.Sc. (Computer Science), Thammasat University, Thailand, M.Sc. (Computer Technology) from Asian Institute of Technology, Thailand. She obtained Ph.D. in Electrical and Computer Engineering from King Mongkut's University of Technology Thonburi, Thailand. Her researches are in the fields of Artificial Intelligence, Natural Language Understanding, Text Network Analysis, Data Science in Education. She can be contacted at email: krachada@tu.ac.th.