Chapter #

NON-FUNCTIONAL REQUIREMENTS ELICITATION

Luiz Marcio Cysneiros[1] and Eric Yu[2]
[1]Department of Mathematics and Statistics
York University
cysneiro@mathstat.yorku.ca
[2]Faculty of Information Studies
University of Toronto
yu@fis.utoronto.ca

1. INTRODUCTION

Software developers are constantly under pressure to deliver code on time and on budget. As a result, many projects focus on delivering functionalities at the expense of meeting non-functional requirements such as reliability, security, maintainability, portability, accuracy, among others. As software complexity grows and clients demand higher and higher quality software, non-functional properties can no longer be considered to be of secondary importance. Many systems fail or fall into disuse precisely because of inadequacies in these properties. These non-functional aspects have been treated as properties or attributes after the fact. While these properties have always been a concern among software engineering researchers, early work have tended to view them as properties or attributes of the finished software product to be evaluated and measured. Recent work offer the complementary view that they should be treated as requirements to be dealt with from the earliest stages of the software development process [6][7], and then throughout the entire life cycle.

This chapter will start by defining NFRs and showing its importance within the software development process. It continues by showing an approach to elicit NFRs and pointing out future trends in the treatment of NFRs.

2. NON-FUNCTIONAL REQUIREMENTS

2.1 What are Non-Functional Requirements?

NFRs are also known as Quality Requirements [4] [2] and unlike Functional Requirements, NFRs state constraints to the system as well as particular notions of qualities a system might have, for example, accuracy, usability, safety, performance, reliability, security. Hence, we can say that while functional requirements state "what" the system must do, NFRs constrain "how" the system must accomplish the "what". As a consequence, NFRs are always linked to a Functional Requirement [11][6].

Functional requirements address specific problems and are therefore typically implemented through particular localized modules or components. Although they are often stated informally, they can be formalized whenever necessary.

On the other hand, NFRs define global constraints on a software system or subsystem, on a functional requirement, on the development process or on the deployment processes. They are global in the sense that they arise from all parts of the system and from their interactions. There are well developed notations for specifying functional requirements, e.g., Structured English, Use Cases/UML, and various formal methods approaches. In comparison, NFRs are much harder to specify or to characterize formally. As a result, they are generally stated informally in requirements documents, making them difficult to enforce during development and difficult to be evaluated by the customer prior to delivery. NFRs are hard to be evaluated by stakeholders because they may be interpreted differently under different contexts.

Functional and non-functional requirements frequently appear together, as the non-functional needs to refer to the functional. Suppose we are dealing with a system to control an Automatic Teller Machine (ATM). There would be a functional requirement "The system must allow the customer to withdraw money". Associated to this functional requirements we could have one or more NFRs such as "For security reasons, transactions should be completed within 5 minutes requiring a response time of less than 3sec in at least 90% of the cases. But the system must also be secure when transmitting

data, thus encryption should be used which may compromise the 3 secs goal"

A common approach to understand how an NFR will constrain the functional requirement is to decompose the NFR into sub-goals represented by a graph structure inspired by the And/Or trees used in problem solving methods. This process continues until the requirements engineer considers the NFR sufficiently satisfied.

Another important characteristic of NFRs is that meeting different NFRs may lead to conflicting solutions to be dealt with. For example, to address security concerns one might have several choices, among them, setting a two level password or to use biometrics. Using a two level password would conflict with usability concerns while the use of biometric devices might conflict with cost concerns. Another example can be seen within the clinical analysis laboratory domain. We could have the following functional requirement:" The system must have a file containing all the clients to be used by the Marketing Division". Together with this functional requirement we have the following NFR: "This file must be complete enough to allow the Marketing Division to analyze prospective new clients". But an NFR associated with client's attendance states: "Patient's admission must take less than 4 minutes". In this case, having a comprehensive file with all the information needed by the marketing division would possibly be conflicting with the goal of admitting a patient in less than 4 minutes.

2.2 Why NFRs

There has been work showing that complex systems must deal with non-functional aspects [10] [23] [5]. These Non-Functional aspects should be dealt within the process of Non-Functional Requirements (NFR) definition.

Errors due to omission of NFRs or not properly dealing with them are among the most expensive type and most difficult to correct [23] [12] [7]. Recent work points out that early-phase Requirements Engineering should address organizational and Non-Functional Requirements, while later-phase focus on completeness, consistency and automated verification of requirements [26].

There have been reports showing that not properly dealing with NFRs have led to considerable delays in the project and consequently to significant increases in the final cost. The development of a real time system by Paramax System Corp. experienced major delays in its deadlines and significant increasing costs which put the deployment in risk. There were many reasons for that, but one of the most important reasons relies on the fact that performance was neglected during the development of the software

leading to several changes in both hardware and software architecture, as well as in either the design and code of the software [17].

A more serious problem related to NFRs can be seen in the London Ambulance Service Report [14] [3]. The London Ambulance System was deactivated just after its deployment because, among other reasons, many non-functional requirements were neglected during the system development such as: reliability (vehicles location), cost (emphasis on the best price), usability (poor control of information on the screen), and performance (the system did what was supposed to do but he performance was unacceptable).

2.3 Approaches for dealing with NFRs

Most of the early work on NFRs focused on measuring how much a software system is in accordance with the set of NFRs that it should satisfy, using some form of quantitative analysis [1] [13] [15] [22], offering predefined metrics to assess the degree to which a given software object meets a particular NFR.

Recently, a number of works proposed to use approaches which explicitly deal with NFRs before metrics are applicable [6][2] [8][16]. These works propose the use of techniques to justify design decisions on the inclusion or exclusion of requirements which will impact on the software design. Unlike the metrics approaches, these latter approaches are concerned about making NFRs a relevant and important part of the software development process.

Boehm and In [2] propose a knowledge base where NFRs are prioritized through stakeholders' perspectives, dealing with NFRs at a high level of abstraction. Kirner [16] describe properties for six NFRs from the real-time system domain: performance, reliability, safety, security, maintainability and usability. This work provides heuristics on how to apply the identified properties to meet the NFRs and later measure these NFRs. However, it lacks a broader approach that can be applied to other NFRs, in the real-time domain or in other domains.

A significant advance was introduced when NFRs where treated as competing goals that are extensively refined and traded off among each other in an attempt to arrive at acceptable solutions. The NFR Framework is one of the few works to deal with NFR starting from the early stages of software development through a broader perspective. The NFR Framework [6]views NFRs as goals that might conflict among each other and must be represented as softgoals to be satisficed. The softgoal concept was introduced to cope with the abstract and informal nature of NFRs. Each softgoal will be decomposed into sub-goals represented by a graph structure inspired by the And/Or trees used in problem solving. This process continues until the

requirements engineer considers the softgoal satisficed[1] (operationalized). Initially vague, NFRs are eventually operationalized in terms of techniques that can be implemented. Operationalizations can be viewed as functional requirements that have arisen from the need to meet NFRs. However, as important as getting a well-formed and as-complete-as-possible set of requirements, we need to understand and systematize how requirements will drive the rest of software development, especially during the design phase. None of the above work tackle this problem.

Approaches to NFR could be classified into product-oriented and process-oriented. Product-oriented approaches are those concerned with measuring how much software complies with non-functional requirements. They do not help to prevent problems but are helpful to evaluate the degree of compliance with non-functional needs.

Process-oriented approaches focus on the software development process. It aims to help software engineers searching for alternatives to sufficiently meet NFRs while developing the software. It also helps to justify design decisions. Under the process-oriented approach we may follow guidelines such as the ISO 9126 or using a goal-oriented approach such as the KAOS framework [28] [6]. One of the advantages of the goal-oriented approach is that it can be used to model and reason about both functional and non-functional requirements.

2.4 Dealing with NFRs

Dealing with NFRs involves many different activities such as eliciting, modelling, and analyzing. Each of these activities has its own challenges.

Eliciting NFRs calls for understanding the domain and gathering organizational knowledge. Because NFRs are generally stated and because they are not as clear in stakeholders' minds as functional requirements, eliciting them poses a great challenge. Existing knowledge about NFRs should be used whenever possible to guide on NFRs elicitation.

Once NFRs are elicited, they have to be modelled. Modelling NFRs allows them to be organized for better visualization and understanding. It will help software engineers to analyze NFRs.

Analyzing NFRs calls for reasoning about how well each NFR is being satisfied and to reason about possible conflicts. This might calls for further refining NFRs and it might also bring new conflicts to light. Alternatives must be found to deal with conflicts allowing tradeoffs to be made among different stakeholders. Each alternative must be evaluated to express to

[1] We use, here, the same notion used in [Chung 00] that an NFR can rarely be said to be satisfied. Goal satisficing suggest that the solution used is expected to satisfy within acceptable limits. The term satisfice was coined by Hebert Simon to express "good enough" alternatives.

which degree it may introduce positive or negative influence to one or more NFRs. For example, satisficing security might call for the use of some encrypting mechanism, but the use of this mechanism might conflict with performance needs.

In this work, we will describe a strategy to deal with NFRs from the early phases of requirements engineering to design. We will first describe how to elicit NFRs, where we will be describing an approach for gathering, modelling and analyzing NFRs. Then, we will show how these NFRs can be incorporated into the design process.

3. ELICITING NFRS

We propose to elicit NFRs using a strategy anchored in the Language Extended Lexicon (LEL) [18]. LEL is used to capture the vocabulary used in practice in the domain. Its objective is to register the vocabulary of a given UofD[2]. It is based upon the following simple idea: understand the problem's language without worrying about deeply understanding the problem. LEL is mainly used to register terms (words or phrases) peculiar to a specific field of application.

LEL is useful not only for understanding the domain but also as a starting point for creating different models throughout the process. The strategy uses LEL as a natural language-oriented front-end to support the NFR elicitation process. In addition, capturing the vocabulary in an organized form benefits the reuse of the domain knowledge.

To elicit NFRs you may use an existing LEL or, in case it does not yet exist, you must build a new one. You must add to the existing, or recently created LEL, the NFRs desired by the stakeholders. To do that, you run through all the LEL symbols using a knowledge base on NFRs, expressed in the form of catalogues, to ask ourselves and the stakeholders (whenever possible) whether each of the NFRs presented in this knowledge base applies to each of the LEL symbols. Once you have the LEL showing all the desired NFRs and some of their operationalizations, we represent these NFRs in a set of NFR graphs using the NFR Framework extended with a few new features. The NFR framework allows a deeper level of modelling and reasoning about NFRs than within the LEL. Finally you examine the set of NFR graphs looking for possible interdependencies. Figure 1 illustrates the approach.

[2] *"Universe of Discourse is the general context where the software should be developed and operated. The UofD includes all the sources of information and all known people related to the software. These people are also known as the actors in this UofD."*

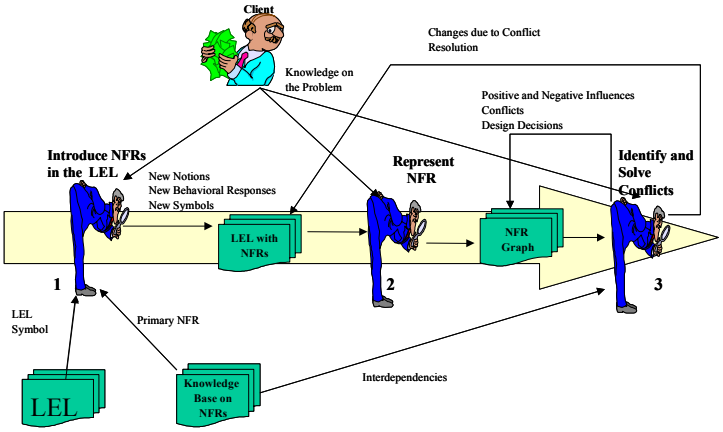

Figure #-1. Eliciting Non-Functional Requirements

3.1 Using the LEL to support NFRs elicitation on earlier phases of requirements engineering

LEL is based on a controlled vocabulary system composed of symbols where each symbol is an entry expressed in terms of notions and behavioural responses. The notions record the meaning of the symbol and its fundamental relations with other entries. The behavioural responses specify the connotation of the symbol in the UofD. Each symbol may also be represented by one or more *alias*es and will be classified as a *subject*, a *verb* or an *object*.

The construction of the Lexicon is guided by the principles of minimal vocabulary and circularity. The circularity principle prescribes the maximization of the usage of Lexicon symbols when describing Lexicon entries, while the minimal vocabulary principle prescribes the minimization of the usage of symbols exterior to the Lexicon when describing Lexicon entries. Figure 2 shows an example of an entry in the LEL. The underlined words/expressions are other symbols of the LEL.

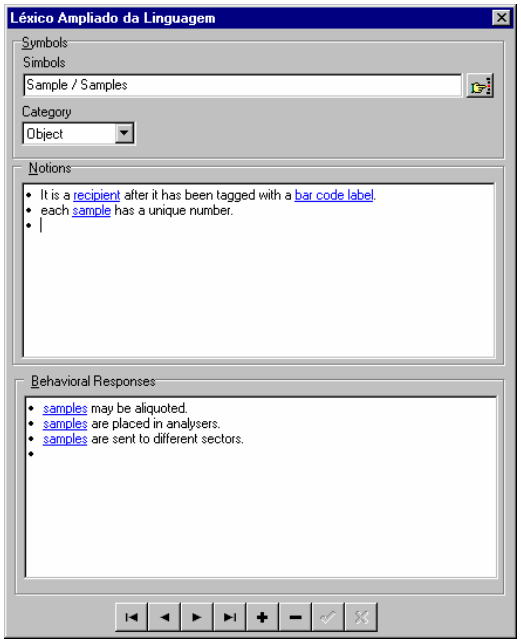

Figure #-2. One Entry of a Symbol Before Analyzing it for NFR

Since the LEL is not a function-oriented description, it has entries which refer both to the functional and to the non-functional perspectives.

Although the LEL can handle non-functional aspects of the domain, at least the very first version of the LEL is usually mainly composed of symbols related to functional requirements. This is due to the very abstract nature of non-functional requirements and because quality aspects, in spite of their importance, are usually hidden in people's minds. However, it does not mean you cannot register information about non-functional requirements. If you happen to find out that one symbol requires an NFR, you should represent it in the symbol notions. A well-defined set of symbols representing the vocabulary of the UofD is an important step to be taken.

We have extended the LEL to help NFRs elicitation. The LEL is now structured to express that one or more NFRs are needed by a symbol. It is also structured to handle dependency links between one NFR and all the notions and behavioral responses which are necessary to satisfice this NFR. Figure 6 shows these new features of the LEL. This extension is implemented in an extended version of the OORNF tool [24]

Building LEL consists of identifying all the meaningful terms (words or sentences) used in the UofD. Each term will be a LEL symbol and must contain at least one notion and one behavioral response. One good approach to start gathering LEL symbols is to read documents used in the domain. These documents usually contain several terms which would become LEL

symbols. Notions and behavioral responses for these symbols may not be perceived at first and would have to be elicited by interviewing related stakeholders and users or sending them questionnaires. Hence, validating already elicited symbols with stakeholders and users is a constant activity carried out several times during LEL construction. When you define notions and behavioral responses for a symbol you must use, whenever possible, existing symbols to express these notions and behavioral responses. If you are using the OORNF Tool [24], the tool will automatically identify that you have used a symbol and mark it as such by underlining it. You may also check if any terms used to express notions and behavioral responses are good candidates to be LEL symbols. Frequently, when we are defining a symbol we discover new symbols that will have to be detailed later. This process goes on until no new symbols arise.

As we can see in Figure 1, the first step for building the non-functional perspective is to enrich the existing LEL with the NFRs customers desire. To do that, you run through all the LEL symbols using the NFR knowledge base to ask yourself and the stakeholders if any of the NFRs in this knowledge base may be necessary for each of the LEL symbols.

3.2 Using Catalogues to Enhance the Lexicon with NFRs

Now that we have LEL done, we must enrich it with NFRs. NFRs are usually complex, global, conflicting and numerous. Aside from that, both software engineers and stakeholders are not used to recognizing NFRs. Because of that, we will use a knowledge base, here presented in the form of catalogues, to guide the requirements engineering through possible needed NFRs and the possible operationalizations for each NFR found.

In this work, we will present two catalogues as examples, one for privacy and another for traceability. You are encouraged to update these catalogues with further operationalizations and to keep your own catalogues on NFRs. Doing so will facilitate future reuse of acquired knowledge on NFR elicitation.

3.2.1 The NFR Framework

As said before, the NFR Framework [5][6]views NFRs as goals that might conflict among each other and must be represented as softgoals to be satisficed. Each softgoal will be decomposed into sub-goals represented by a graph structure inspired by the And/Or trees used in problem solving. This decomposition is done using contribution links. Contribution links can be categorized as an *or* contribution or an *and* contribution. Contribution links allow one to decompose NFRs to the point that one can say that the

operationalizations to this NFR have been reached (i.e., the goals are no longer "soft"). Operationalizations can be viewed as functional requirements which have arisen from the need to meet NFRs. This can explain why we frequently face doubts about whether a requirement is functional or non-functional. Take for example a clinical analysis laboratory. We may have stated a requirement like: "Samples should be traceable so one can know, at different times, where this sample is". This may appear to be a functional requirement while, in fact, it is a functional requirement: "The software must handle samples" constrained by the NFR Traceability. The fact that an NFR when operationalized may result in new functional requirements points to the virtual impossibility of eliciting *all* the functional requirements before eliciting non-functional requirements. An iterative process where you elicit some of the functional requirements then look for NFRs which will, in its turn, generate new functional requirements is more likely to be adequate. The NFR framework also uses correlation links to show contributions (positive and negative) from one NFR to another. Figure 3 and 4 will show examples of NFR graphs used to represent the knowledge base on NFRs.

Contribution links are the core of design decisions. By reasoning about how different operationalizations would contribute to satisfice a softgoal, one may decide which the best alternative to pursue is. Based on the semantics of the contribution links [Chung 00], decision values are propagated from an offspring to its parents allowing one to visualize what impact would come from adopting one alternative instead of another.

3.2.2 The Catalogues

Figure 3 shows a catalogue for privacy. We can see for example, that Privacy can be refined into Limit Use and Disclosure of Data, which is further decomposed into Minimize Disclosure and Collection of Personal data and later decomposed, among other options, into Reduce Need for Personal Data. To satisfice the latter, we may find three options: Use Anonymous Payment, Use Digital Certificates and Use Anonymous Profile. These options can be used alone or together to achieve different needs for privacy. Notice that to Use Digital Certificates while contributing to Privacy will eventually hurt Maintainability since personal data change over time. The use of Public Key Cryptography can implement Digital Certificates but may also have negative impact on Performance.

Figure 4 shows the catalogue for Traceability. We can see in this figure that Traceability is first decomposed into Traceability for Processes and Things. Processes will tackle concerns about being able to reconstruct all the steps of a Process such as furnishing a piece of equipment. When we furnish a piece of equipment several steps are usually involved in the

process. If some piece of equipment (e.g. a laptop computer) shows problems when tested before shipping, the manufacturer might want to be able to trace all the steps involved in furnishing that laptop to trace the cause of the problem.

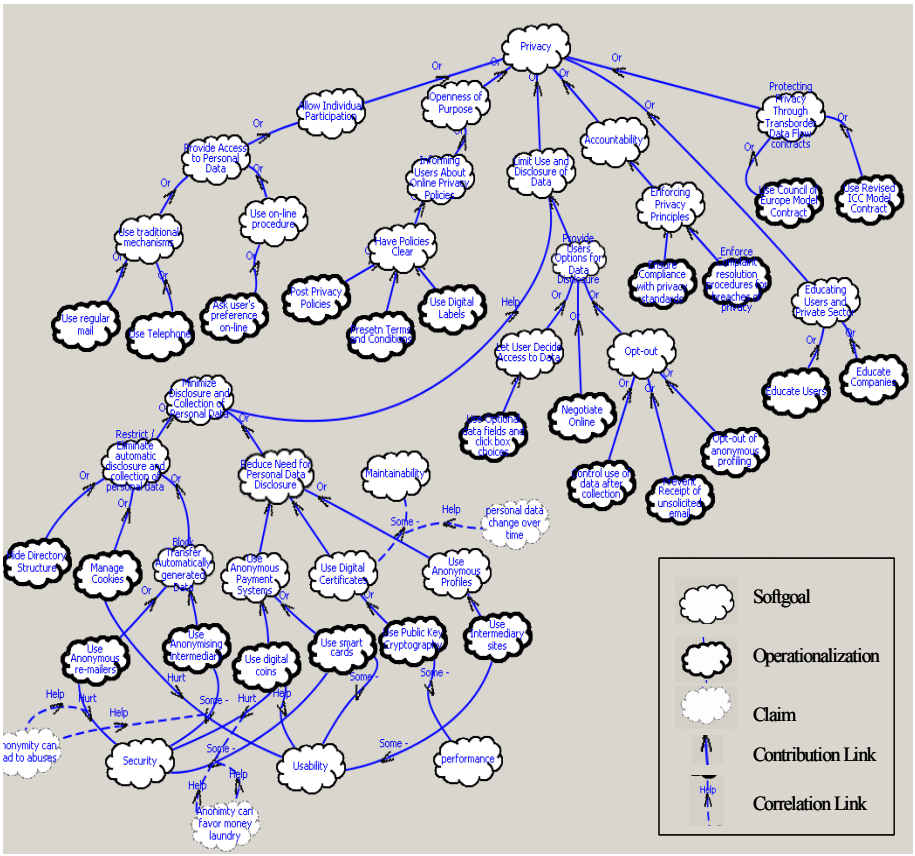

Figure #-3. Catalogue of Privacy Alternative Solutions

Tracing Things may involve many different options. Here, the term Things might be applied, but not necessarily restricted, to people, objects and information. One might want to trace the places some object is or has been located. One might also want to trace what changes were made to an object or to a piece of information in order to assure Security and Reliability. Another option is that one might want to trace times for an object or information. For example, one might want to know when an object was moved from one place to another, or simply when one object was changed. Finally, it is also possible that we may want to keep trace of whole-part relationships expressing for example that an object was aggregated to another or one was split from another. Each of these options

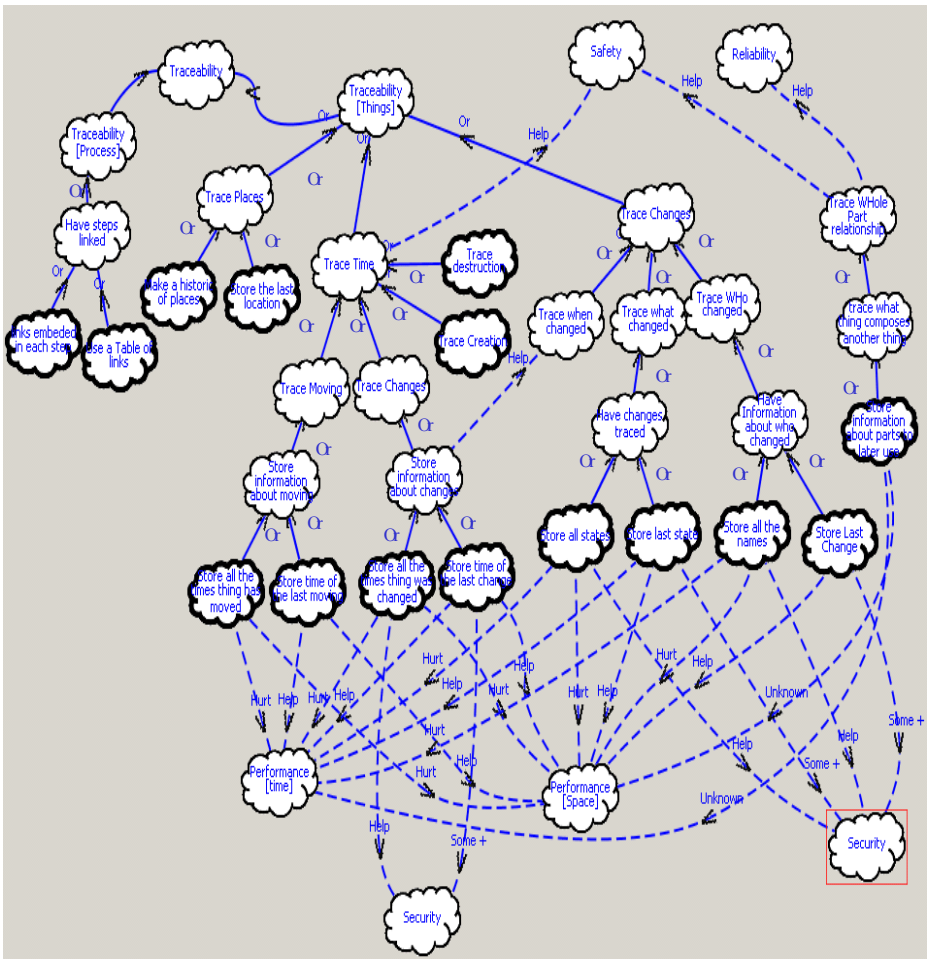

Figure #-4. Catalogue of Traceability Alternative Solutions

can be refined to different possibilities for operationalizations. For example, Trace Change might be refined into Have Changes Traced. This will call for the need to store information about when an object or information was changed. For example, in a hospital, I may want to trace every change made to a patient record so I can precisely follow the treatment prescribed to a patient and the associated pathology. On the other hand, when storing schedule for Nurses shift, I may want to keep the last used schedule just in case one nurse fails to show up and I have to ask another nurse to do a double shift. Knowing the last schedule may help on such a decision but further storage of previous schedules would not be necessary. Thus, operationalizing this goal can be done in two different ways. You can Store all states for a giving Thing or you can simply Store Last State. Notice that to Store all states hurt (contribute negatively) to both time and space

performance, but will also help (contribute positively) to security and reliability. Tradeoffs should be made to evaluate which alternative would prevail for the case being studied.

3.2.3 Using the Catalogues to enrich LEL with NFRs

In Adding NFRs to LEL, catalogues will be used as guidance for knowledge reuse. Remember that although we only show two catalogues here, you may create your own catalogues. Chung [6] provides catalogues on security, accuracy and performance. A comprehensive, although not exhaustive, list of possible NFRs can also be found in Chung [Chung 00]. The set of catalogues will be first used to remind you of the possible NFRs a system might need. As stated before, you may ask yourself and the stakeholders and end users (whenever possible) if any of the NFRs may be needed for each symbol of the LEL. Hence, starting from the first LEL symbol you will ask if performance, traceability, security, privacy and so on would somehow be important or impact the concept represented by the symbol. If the answer is positive, you may represent this NFR as a notion for this symbol. Then again, using the catalogue you may investigate the possible alternatives for operationalizing this NFR. You may then represent the operationalization in the notions or behavioral response of the symbol, whichever appropriate. It is also possible that operationalizing this NFR requires to add notions or behavioral responses to one or more other symbols. If none of the operationalizations satisfices stakeholders' expectations you must find new ways for satisficing this NFR and later update the catalogue.

Suppose you are developing an information system (LIS) for a clinical analysis laboratory. For each of the symbols present in LEL you will ask yourself and the stakeholders what possible NFRs would have to be achieved in order for this symbol to be considered completely represented. Suppose you were analyzing the symbol Sample and realized that traceability would be important to Sample since the laboratory cannot afford to lose samples. In Figure 5, it is possible to see in the symbol's notions the addition of the NFR Traceability. NFRs will be represented in the notions using the following pattern: "Has NFR"+NFR

Now that you know a Sample has to be traceable, you have to reason about how this might be achieved. You may use the catalogue trying to reuse existing operationalizations or you can simply ask yourself and the stakeholders how this traceability should be achieved. You may also apply both approaches, which in fact has been most effective.

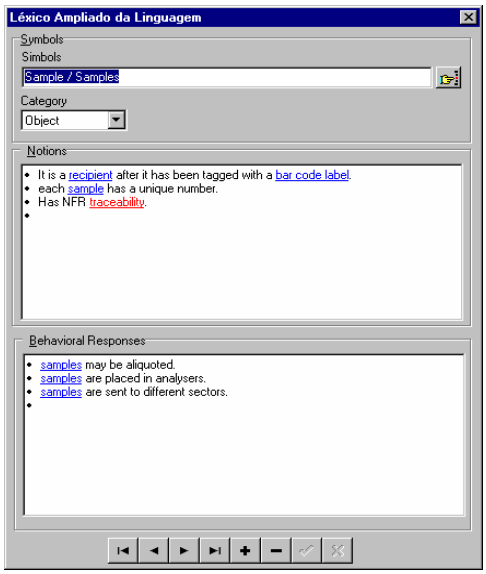

Figure #-5. Symbol After One NFR was Picked up

One response you could get is: "one should be able to know where a sample is now and where it has been before". Reflecting about the behavioral response placed in the symbol Sample, you could realize that this answer was not sufficient to achieve the traceability NFR since we were not specifying how one could actually know where a sample is at any needed time. Checking the catalogue you could see that one way to operationalized traceability for places is to Keep a History of Places. Thus you could realize that to know the exact position of a sample at any time, it is necessary to scan this sample every time it is transported from one place to another. To represent this knowledge, you could create a new symbol called Scan Sample (Figure 6).

Another answer you could get from questioning about the NFR traceability to the Sample entry is: "every time a sample is aliquoted (expression used in this domain meaning to create an aliquote, or yet to draw from one recipient to another) this procedure has to be recorded so one can know which sample was originated from another sample". Notice that this answer is expressed in the catalogue in the form of the Trace Whole-Part Relationship goal. You should then represent this answer as an entry in the behavioral responses (LIS keeps a record of what sample is originated from another) of the symbol Aliquote sample (Figure 7).

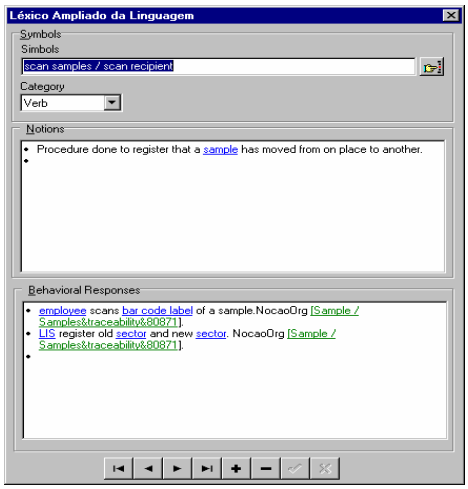

Figure #-6. A Symbol Created to Satisfice a NFR From Another Symbol

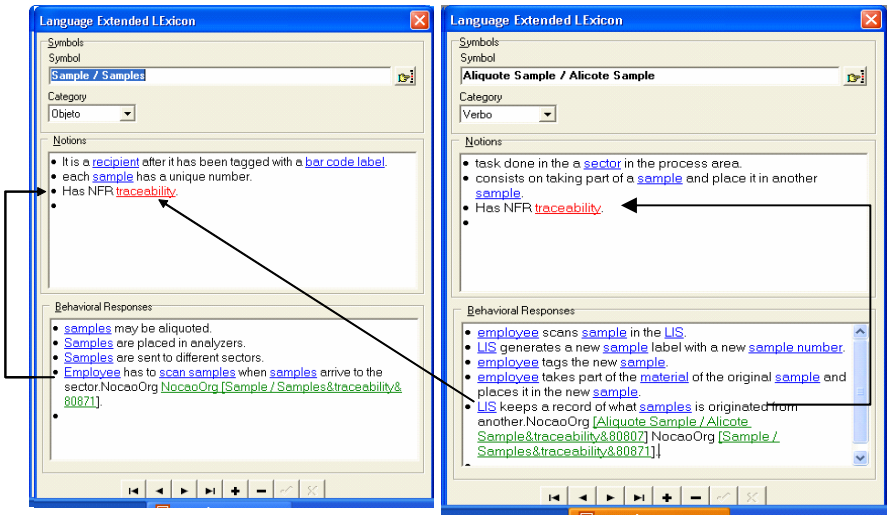

Figure #-7. Consequences of Satisficing the NFR of the Symbol Sample.

As you add a new behavioral response to the symbol Aliquote Sample, you must also create a dependency link between this behavioral response and the NFR traceability stated in the notions of the symbol Sample. This link is represented in the tool by a pattern following the rule: "NocaoOrg [" + LEL symbol + "&" + NFR + "&" + internal number]. The string "NocaoOrg" is used to differentiate this entry so one can clearly see that this notion or behavioral response exists to operationalize a NFR present in "LEL symbol". LEL symbol will contain the LEL symbol which has the NFR which originated the need for this notion or behavioral response. NFR will contain

what NFR in the referenced symbol has originated this notion or behavioral response. This is necessary since a symbol may have more than one NFR in its notions. The internal number is used by the tool for traceability purposes. In the case of the example addressed in the paragraph above we have the following pattern: NocaoOrg [Aliquote Sample/ Alicote Sample&Traceability&80807]. Notice that one behavioral response can be traced to more than one symbol. That is the case in Figure 7 where the last behavioral response in the symbol Aliquote Sample has links to the Aliquote Sample symbol itself and to the Sample symbol. This pattern is used mainly by the OORNF tool although it can be also used as a quick guide to find out what NFR in which symbol has originated the need for a particular notion or behavioral response.

We can see the behavioral responses representing the operationalizations for the above situations represented in Figure 7, respectively as a behavioral response in the Aliquote sample symbol (LIS keeps a record of what sample is originated from another) and another behavioral response in the Sample symbol itself (Employee has to scan samples every time a sample arrives in the sector).

You must proceed this way for all the symbols of the lexicon, thus at the end of the process you end up having the lexicon expressing at least the basic necessary NFRs and some of their operationalizations.

Representing NFRs in the LEL will help you to start getting acquainted with necessary NFRs and possible operationalizations to them, but LEL is not the best tool to deal with dependencies among NFRs since they frequently involve many conflicts among possible solutions to satisfice one or more NFR. Thus, once you have the lexicon enhanced with NFRs you will fully represent and reason with NFRs using the NFR Framework with some slight adaptations.

3.3 Refining NFRs using NFR graph

In accordance with [Chung 00], for us an NFR has a *type*, which refers to a particular NFR as for example security or traceability. It also has a subject matter or *topic*, for example Sample as showed in the above example. We would then represent it as Traceability [Sample].

The NFR framework was extended to represent the operationalizations in two different ways. We called them dynamic and static operationalizations. Dynamic operationalizations are those that call for some action to be carried out. Static operationalizations express the need for some data to be used in the design of the software to store information which is necessary for satisficing the NFR. Figure 8 shows an example of an NFR graph where we can see these two types of operationalizations. Categorizing

operationalizations as Dynamic and Static will later help map these operationalizations into attributes or operations belonging to a class. Cysneiros [9] shows how to integrate NFRs into Class diagrams.

On top of the Figure 8 (extracted from a case study for a Light Control System), we can see the root of this graph represented as Safety [Room], meaning that room is a place which has to be safe regarding illumination aspects, i.e. the room has to have enough light so that people do not stumble and fall.

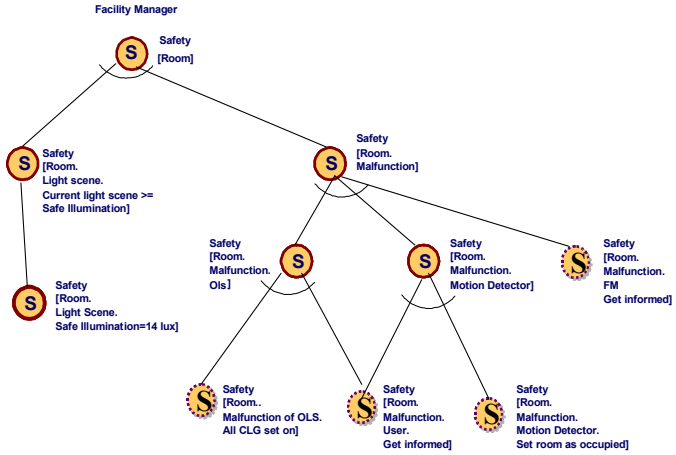

Figure #-8. An Example of a NFR Graph

One of the operationalizations that represent part of this NFR satisficing can be seen on the left side of the figure represented by a bold circle denoting a static operationalization. Here, we can see the need of some information in the system which represents the minimum illumination in lux that can be used in a room.

On the bottom of the figure we can see dotted circles representing dynamic operationalizations. One of them, Safety [Room.Malfunction.User. Get informed], represents that the user may be informed of any malfunction that occurs in the room. The letter S inside each node represents that this sub-goal is Satisficed. The letter P is used for those ones that are Partially satisficed or D for those ones that are Denied.

It is important to stress that the identifier that appears close to the NFR on the root of the graph (NFR Topic) must be a LEL symbol. In Figure 8 we see that the root node is represented by Safety [Room], so room must be an LEL symbol. If one cannot find the word or sentence intended to be used as a topic for an NFR, then either one symbol represented in the LEL has an alias not yet defined or the LEL is incomplete and therefore, must be updated.

Reasoning about different NFRs frequently leads to tradeoffs to be made. To understand and reason about the different alternatives involved in these

tradeoffs you might need to further clarify some NFRs' operationalizations and to negotiate which NFR should be denied or partially denied prejudicing another NFR. To do that, whenever possible, you might talk to the stakeholders who would be affected by such decisions. To be able to trace NFRs and their operationalizations back to these stakeholders you should represent above the NFR graph the source in the UofD for the information expressed in the graph.

To build the NFR model, you must go through every entry of the LEL looking for notions that express the need for an NFR. For each NFR found one must create an NFR graph where this NFR will be the root of the graph. This graph must be further decomposed so it expresses all the operationalizations necessary to satisfice this NFR.

Let us take for example the symbol Inspect Result belonging to a case study for a clinical analysis laboratory. One NFR we find in the notions of this symbol is Traceability, since the system must assure, in case of conflicting test results, that someone can figure out what problems happened that led to the conflict. Figure 9 shows the entry for this symbol and illustrates how an NFR graph would be originated from there, while Figure 10 shows the navigation facilities of the OORNF tool, where we can see the notions and behavioral responses that were added to satisfice the NFR Traceability [Inspect Test] (Figure 9). In this case, there were only behavioral responses.

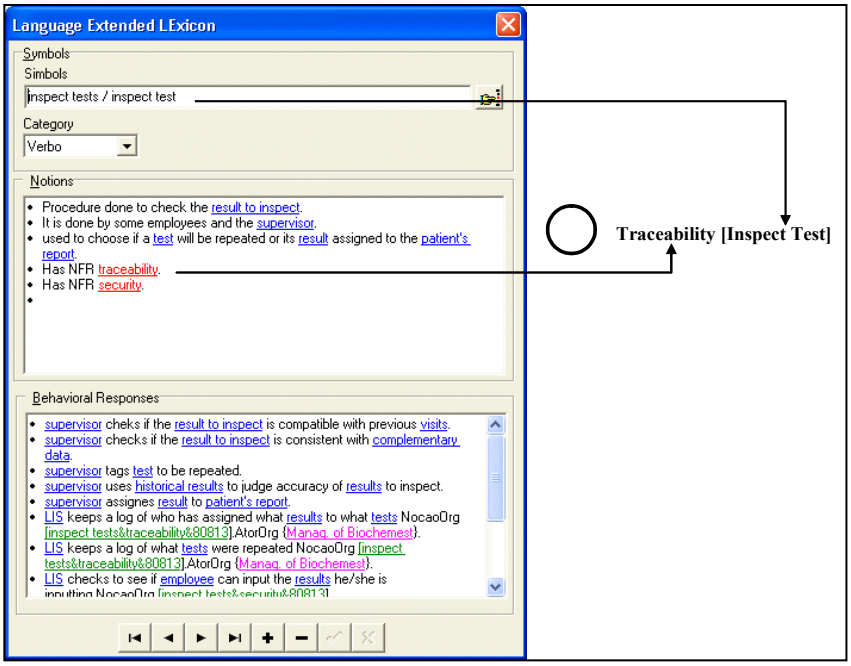

Figure #-9. Creating an NFR graph

Once we have represented the NFR graph root, we have to find out its operationalizations. We can do it by either using catalogues such as those presented in section 3.2.1 or the OORNF tool to examine what notions and behavioral responses were added to the LEL to satisfice an NFR. These notions and behavioral responses will be candidates to operationalize this NFR. These two approaches are not conflicting; in fact, they are complementary to each other.

Using the behavioral responses shown in Figure 10, we represent possible operationalizations of the Traceability [Inspect Test] NFR as it can be seen in Figure 11. Once we have done that, we try to see what possible sub-goals, if any, would represent an intermediate step between the graph root and its operationalizations. We also try to find out if additional decompositions can be made to satisfice Traceability, checking the alternatives presented in the catalogue.

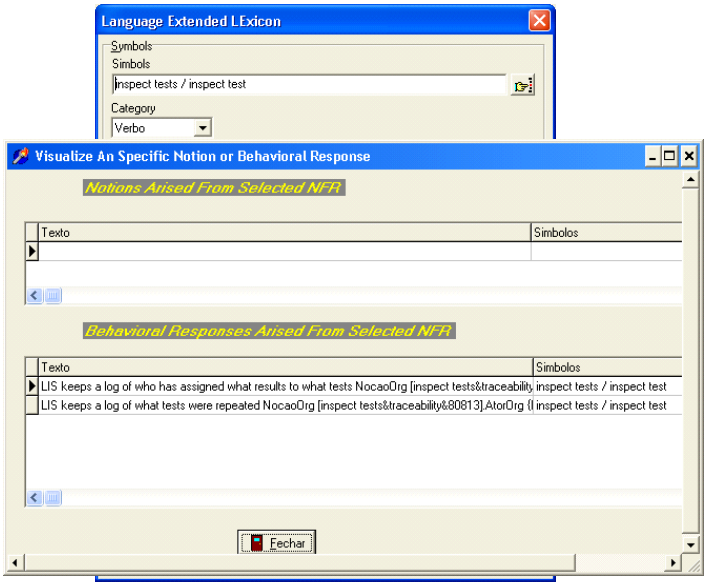

Figure #-10. Navigating an NFR to Find its Operationalizations

We may proceed in two different ways:

1) We can precede the evaluation in a bottom-up approach. For example, taking the operationalization Traceability[Which tests were repeated] may direct us to understand that between Traceability [Inspect Test] and these operationalizations we might have an intermediate decomposition Traceability [Tests].
2) Decomposing the root using a top-down approach. For example, if we use the catalogue for traceability presented before we can imagine that we could need to decompose Traceability [Inspect Test] into

Traceability [Trace Changes]. By doing so we could realize that aside from registering who entered results, we could also have to register the new value and the time the change happened. Since we are dealing with the delicate subject of finding problems in test results that could lead to serious harm to patients, we decided to do it storing all values instead of only storing the last change.

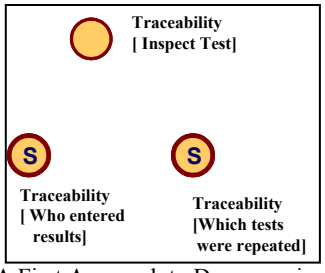

Figure #-11. A First Approach to Decomposing an NFR

Figure 12 shows the resultant graph after reasoning about Traceability for inspect test.

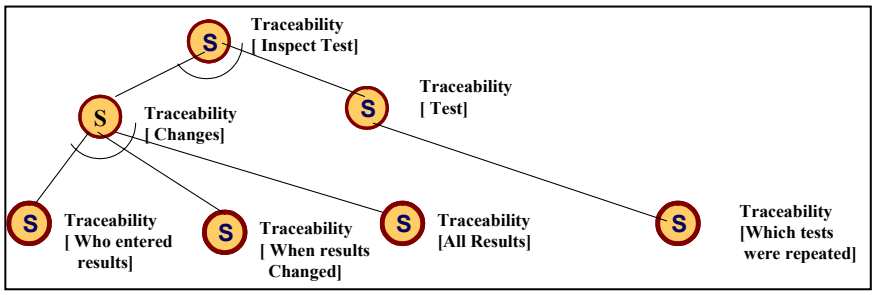

Figure #-12. Resultant NFR Graph

Note that since we are representing the actor of the UofD who is directly interested in the NFR, we can represent different viewpoints for the same NFR. You just have to build two different graphs with the same root. Each one will operationalize the NFR according to individual viewpoints, which can be either similar or conflicting.

Figure 13 shows an example for dealing with different viewpoints for the same NFR. Here we show the NFR Operational Restriction for a clinical analysis laboratory information system. We represent two different viewpoints, one from the manager of the medical bureau (area responsible for reviewing and signing patient's reports) and the manager of the processing area (area responsible for processing all the tests and to enter results). Through the manager of the medical bureau viewpoint to satisfice operational restrictions regarding patient's record, the system would have to be able to electronically sign the patient's report, meaning the system should

be able to identify when all the results are ready and print them with no further delays using some authorized signature previously digitalized. However, when we consulted the manager of the processing area about this same NFR, she said that although she recognized the need for shortening the patient's report delivery time, not all the tests could be electronically signed because of reliability concerns. Some results could fall into a range of results that would demand a physician to review them before they can be printed and signed. Once we identified this conflict, shown as a dotted line with a minus sign, we started some negotiations between both viewpoints. An agreement was achieved to allow patient's reports to be electronically signed, but only for those with results falling under pre-defined limits.

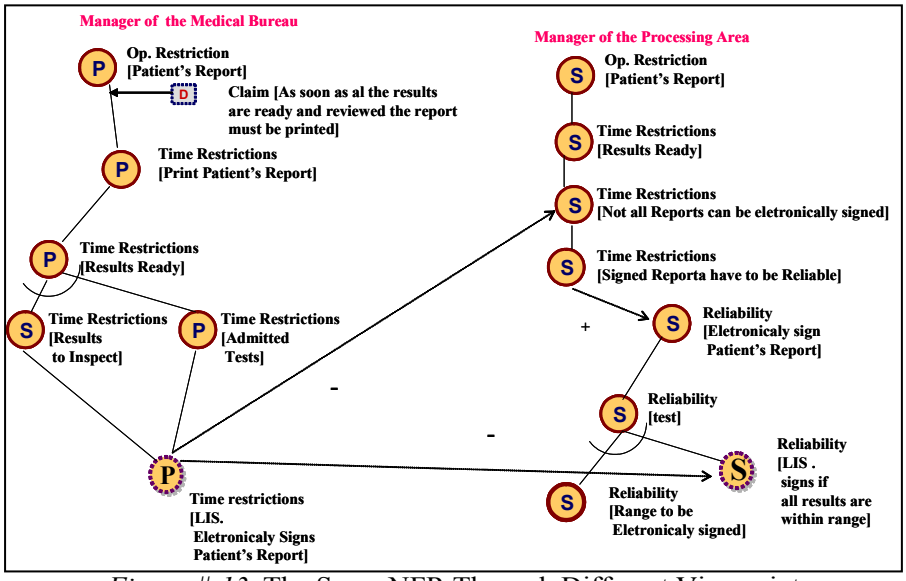

Figure #-13. The Same NFR Through Different Viewpoints

After we have carried out this process for each of the LEL symbols, we will have a set of NFR graphs that will model the non-functional perspective. As such, we can now analyze all the graphs to check for possible conflicts and different design solutions which might be then negotiated with the stakeholders.

It is important to emphasize that all the effort on NFR tradeoffs due to positive and negative interdependencies will take place in the non-functional perspective, i.e., using the NFR framework. What will be integrated into the functional perspective will be the result one gets after all the necessary reasoning on NFRs interdependencies and its consequences, i.e. the operationalizations.

We propose three heuristics to help find these interdependencies.

Compare all NFR graphs of the same type searching for possible interdependencies. For example, we may put all the NFR graphs which have the type Safety together to see if there are any interdependencies among them.

Compare all the graphs classified in the knowledge base [29] as possibly conflicting NFRs. For example, compare graphs of Security with graphs of Usability.

Pair-wise compare all the graphs which were not compared while applying the above heuristics.

The conflict shown in Figure 13 was found by applying Heuristic 1.

4. RELATED WORK

GRL (Goal-Oriented Language) is a language created to support requirements elicitation centred in the idea of goals [20]. The main focus of GRL lies on NFRs and the constructs of the NFR framework are incorporated in GRL. There are three categories of elements in GRL, intentional elements, links and actors.

Intentional elements are used to express the intentions behind the process. It helps to understand the "whys" involved in the process. There are four intentional elements in GRL:Goal, Softgoal, Task and Resources. Using these elements we can model alternatives to the existing process, criteria used to reason among the different alternatives, and the reasons that led to choose on specific alternative.

It is also possible to find detailed work on two important NFRs, security and privacy. Yu [27] uses the ***i**** Framework [Yu97] to propose an approach to model and reason about privacy in the presence of other NFRs. In a complementary work, Liu [21] explores the concept of actors from the ***i**** Framework to elicit, represent and reason about security with a special attention to internet concerns. Since both privacy and security are strongly related to relationships among actors such as stakeholders, customers and malicious users, the focus on actor supported by ***i**** plays an important role.

5. PRATICAL IMPLICATIONS

We have shown here an approach to elicit NFRs. It is based on the use of a lexicon, LEL, to support the modelling of both functional and non-functional models. LEL has been used in several real life case studies with good results. Validating LEL with layman stakeholders such as physicians and nurses has been easy and productive. After building LEL we enrich it with NFRs using catalogues to guide on the search for NFRs. This part of the

strategy aims to facilitate the process of gathering knowledge about NFRs and support some initial refinement of NFRs.

After NFRs are represented in LEL they are represented using NFR graphs which will allow for further refinements and reasoning about possible conflicts and the necessary tradeoffs. Here, the objective is to facilitate the software engineering on finding the set of possible solutions to each NFR and to support negotiation with different stakeholders to achieve a compromise when conflicts are detected.

The strategy proposed here has been applied to different real life case studies. The results point out that using the strategy one might expect to get a more complete software through the perspective of stakeholders and end users with a more efficient use of manpower involved in software development.

We envision that future work on NFRs should address several different aspects. First, it should concentrate on building a knowledge base covering various NFRs and their operationalizations to the greatest extent possible. Second, it should investigate methods to at least partially automate graph generation and conflict detection. Finally it should investigate how NFRs should be handled in emerging software engineering paradigms such as agent orientation where, among other challenges, an agent has the autonomy to decide not to provide a service for another agent at run-time.

7. REFERENCES

[1] Boehm, B. "*Characteristics of Software Quality"* North Holland Press, 1978.

[2] Boehm, Barry and In, Hoh. "*Identifying Quality-Requirement Conflicts"*. IEEE Software, March 1996, pp. 25-35

[3] Breitman, K. K, Leite J.C.S.P. and Finkelstein Anthony. *The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study*. Journal of the Brazilian Computer Society No 1 Vol. 6 Jul. 1999 pp:13:37.

[4] Chung L., "*Representing and Using Non-Functional Requirements: A Process Oriented Approach*" Ph.D. Thesis, Dept. of Comp.. Science. University of Toronto, June 1993. Also tech. Rep. DKBS-TR-91-1.

[5] Chung, L., Nixon, B. *"Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approac*h*"* Proc. 17th Int. Con. on Software Eng. Seatle, Washington, April pp: 24-28, 1995.

[6] Chung, L., Nixon, B., Yu, E. and Mylopoulos,J. "*Non-Functional Requirements in Software Engineering"* Kluwer Academic Publishers 2000.

[7] Cysneiros, L.M. and Leite, J.C.S.P. *"Integrating Non-Functional Requirements into data model"* 4th International Symposium on Requirements Engineering – Ireland June 1999.

[8] Cysneiros,L.M., Leite, J.C.S.P. and Neto, J.S.M. *"A Framework for Integrating Non-Functional Requirements into Conceptual Models*" Requirements Engineering Journal — Vol 6 , Issue 2 Apr. 2001, pp:97-115.

[9] Cysneiros,L.M. and Leite, J.C.S.P. *"Using UML to Reflect Non-Functional Requirements"* Proceedigns of the 11th CASCON, IBM Canada, Toronto Nov 2001 pp:202-216

[10] Dardenne, A.. van Lamsweerde A, Fickas, S.. "Goal Directed Requirements Acquisition". *Science of Computer Programming, Vol. 20 pp: 3-50*, Apr. 1993.

[11] Evaluation of Natural Language Processing Systems, http://www.issco.unige.ch/ewg95 1995.

[12] Ebert, C. "*Dealing with Nonfunctional in Large Software System"s.* Annals of Software Engineering, 3, 1997, pp. 367-395.

[13] Fenton, N.E. and Pfleeger, S.L. *"Software Metrics: A Rigorous and Practical Approach"* 2nd ed., International Thomson Computer Press, 1997.

[14] Finkelstein, A. and Dowell J. "*A comedy of Errors: The London Ambulance Service Case Study*" Proceedings of the Eighth International Workshop on Software Specification and Design*,* IEEE Computer Society Press pp 2-5 1996.

[15] Keller, S.E. et al *"Specifying Software Quality Requirements with Metrics"* in Tutorial System and Software Requirements Engineering IEEE Computer Society Press 1990 pp:145-163

[16] Kirner T.G. , Davis A .M. , *"Nonfunctional Requirements of Real-Time Systems"*, Advances in Computers, Vol 42 pp 1-37 1996.

[17] Lindstrom, D.R. "*Five Ways to Destroy a Development Project"* IEEE Software, September 1993, pp. 55-58.

[18] Leite J.C.S.P. and Franco, A.P.M. *"A Strategy for Conceptual Model Acquisition "* in Proceedings of the First IEEE International Symposium on Requirements Engineering, SanDiego, Ca, IEEE Computer Society Press, pp 243-246 1993.

[19] Leite, J.C.S.P. et.al. " *Enhancing a Requirements Baseline with Scenarios*." Requirements Engineering Journal, 2(4):184-198, 1997.

[20] Liu, L. and Yu E. "Designing Web-Based Systems in Social Context: A Goal and Scenario Based Approach" 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), Toronto, May 27-31, 2002. LNCS 2348 Springer Verlag. pp. 37-51.

[21] Liu, L., Yu, Eric. and Mylopoulos, J. "Analyzing Security Requirements As Relationships among Strategic Actors**"** Proc. of the 2nd Symposium on Requirements Engineering for Information Security, North Carolina – October 2002.

[22] Lyu, M.R. (ed.) *"Handbook of Software Reliability Engineering"* McGraw-Hill, 1996.

[23] Mylopoulos,J. Chung, L., and Nixon, B.*, "Representing and Using Non-functional Requirements: A Process-Oriented Approach",* IEEE Trans. on Software Eng, 18(6), pp:483-497, June 1992.

[24] Neto, J.S.M. "I*ntegrando Requisitos Não Funcionais ao Modelo de Objetos"* M.Sc. Dissertation at PUC-Rio, Mar/2000.

[25] Rational et al, *"Object Constraint Language Specification"* 1997. Http://www.rational.com.

[26] Yu, Eric *"Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering"* Proc. of the 3rd Interna. Symp. on Requirements Eng. Jan 1997 pp:226-235

[27] Yu, E. and Cysneiros, L.M. "Privacy in the Presence of Other Competing Requirements" Proc. of the 2nd Symposium on Requirements Engineering for Information Security, North Carolina – October 2002.

[28] VanLamsweerde, A. "Goal-Oriented Requirements Engineering: A Guided Tour" *Proc of the 5th IEEE Int. Symp. on Requirements Engineering*, pp:249-262, 2001.