Driving Non-Functional Requirements to Use Cases and Scenarios

Luiz Marcio Cysneiros *
Department of Computer Science
University of Toronto
cysneiro@cs.toronto.edu

Julio César Sampaio do Prado Leite*
Departamento de Informática PUC- Rio
julio@inf.puc-rio.br

Abstract

Today, companies are continuously changing and improving their business strategies. As a consequence, stakeholders are demanding more flexible and complex software to be built . To handle this complexity, conceptual models have to describe aspects beyond entities and activities. Recent research has pointed out that dealing with goals in order to capture intentions associated with complex situations is a major aspect to handle this new reality [14]. Non-Functional Requirements are a particular class of these goals that has to be dealt with since the early stages of software development. Therefore, expressing these NFRs in use cases and scenarios models is a must. In this work we show a strategy to drive elicited NFR towards use cases and scenarios that reflect the functional requirements of the software. We tested our proposal through two case studies and the results suggest that our strategy can help developers to deal with complex conceptual models and might result in a more complete software specification and thus, to a shorter time-to-the-market.

1. Introduction

As the globalization becomes an obligation to almost all countries and companies each day, the complexity of software increases in order to deal with problems that arise from this process and from the incredible speed with which the business reality changes.

One of the major aspects of software programs developed during this new age is how well they deal with non-functional aspects, which can be seen as non-functional requirements (NFR) of the software.

Some examples of these NFRs are: reliability, performance, cost, usability and security. Not paying attention to these requirements has produced histories of failures in software development that include the deactivation of a software just one day after its deployment [5]. Errors caused by not properly dealing with NFRs are among the most expensive and difficult to correct [7] [11] [15].

In spite of this, not many work can be found proposing strategies to deal with NFRs under a process-oriented approach, i.e., proposing the development of techniques to justify design decisions on the inclusion or exclusion of requirements that will impact on the software design. Most of the work on NFR rely upon a product-oriented approach, i.e., they are mainly concerned with how much a software is in accordance with the NFRs that it should satisfy [18] [2] [16] [25] [24].

Although

it is important to measure how well a software implements NFRs, we believe, as in [10], that NFRs must be dealt with since the early stages of software development.

When examining the existent literature, we can see that most of the work on using NFRs during the software development process are still partially focused [3] [19]. An important

* This work was in part supported by CNPq

work on how to represent and deal with NFRs is presented by Chung [10] in his NFR Framework that uses NFRs to drive design and to support architectural design

Some of our previous work [11] [12] emphasize the premise presented by Chung [10] that dealing and representing NFRs is not enough. We have to get conceptual models that reflect these NFRs and what is needed to satisfice[1] them.

Handling NFRs sometimes calls for using very detailed aspects as well as ways to handle interdependencies between several NFRs. We believe that most of the artifacts commonly used to represent functional requirements like use cases, scenarios, CRC among others, are not suitable for this job.

This work evolves out of a previous one [12] and aims at helping to deal with NFRs since the early stages of the software development, by bringing to use cases and scenarios the information and actions that are necessary to satisfice the set of NFRs elicited.

We believe that we may face the software development as two distinct evolutionary cycles, one for the functional requirements (named functional view) and another one for the NFRs (named non-functional view). An integration process will take care of driving what is needed to satisfice NFRs, i.e. how to operationalize these NFRs, from the non-functional view to use cases and scenarios

In order to validate the proposed strategy, we conducted two case studies. One utilizing use cases as part of a real situation environment and the other one using scenarios in a controlled environment. We believe that the results we got suggest that the strategy proposed here may lead to more complete software specification, and thus, to a more satisfied stakeholder, to a better time-to-the market and finally to lower maintenance costs.

An overview of the entire strategy to deal with NFRs is presented in Section 2. Section 3 details the integration process and some examples of the use of the strategy. Section 4 presents the results from the case studies used to validate the strategy, and Section 5 concludes and present some future work.

2. An Overview of the Strategy to Deal with NFRs

As previously mentioned, we consider the software development process comprising of two *independent* evolutionary cycles that may be dealt with separately, since we propose to establish convergence points between both cycles. Through the use of these convergence points we can express in the functional view all the actions and data that will be necessary to satisfice the NFR tackled in the non-functional view.

The so-called functional view includes some of the artifacts commonly used in the software development process, while the non-functional view uses the NFR Framework [10] to represent the NFR found.

As portrait in Figure 1, we propose to build both the functional and the non-functional views anchored in the Language Extended Lexicon (LEL) [20]. This will lead to a smooth integration between both views. Therefore, building the Lexicon will the first thing to be carried out in this strategy.

The objective of the LEL 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

[1] We use here the same notion used by Mylopoulos [2] that an NFR can rarely be said to be satisfied. Goal satisficing suggest that the solution used is expected to satisfy within acceptable limits.

[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."*

understanding the problem [20]. The main objective of the LEL is to register signs (words or phrases) peculiar to a specific field of application.

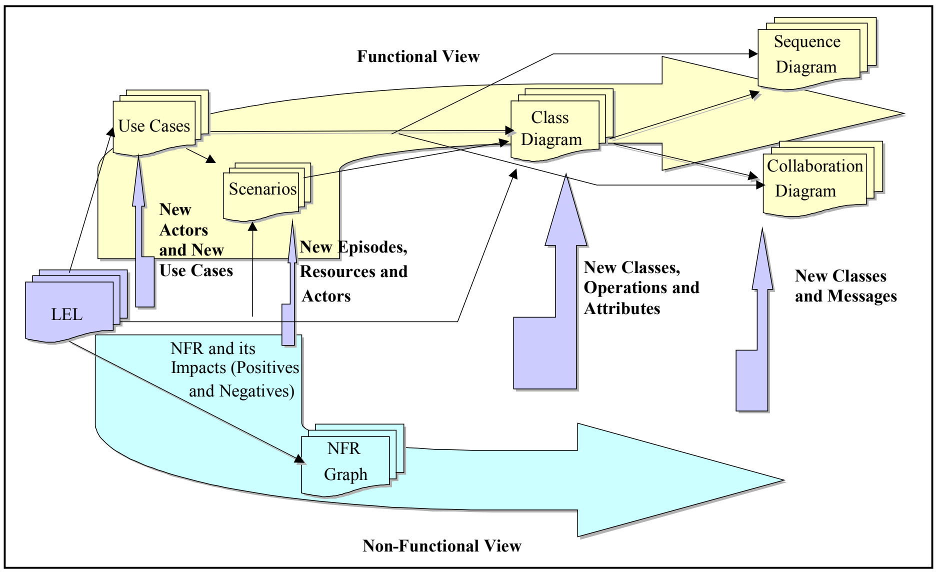

Functional View
Sequence Diagram
Use Cases
Class Diagram
Collaboration Diagram
Scenarios
New Actors and New Use Cases
New Episodes, Resources and Actors
New Classes, Operations and Attributes
New Classes and Messages
LEL
NFR and its Impacts (Positives and Negatives)
NFR Graph
Non-Functional View

Figure 1 – An Overview of the Strategy to Deal With NFRs

The LEL is based on a code system composed of symbols where each symbol is an entry expressed in terms of notions and behavioral responses. The notions must try to elicit the meaning of the symbol and its fundamental relations with other entries. The behavioral response must specify the connotation of the symbol in UofD. Each symbol may also be represented by one or more aliases.

The construction of the LEL must be oriented by the minimum vocabulary and the circularity principles. The circularity principle prescribes the maximization of the usage of LEL symbols when describing LEL entries while the minimal vocabulary principle prescribes the minimization of the usage of symbols exterior to the LEL when describing LEL entries. Because of the circularity principle, the LEL has a hypertext form. Figure 2 shows an example of an entry in the LEL.

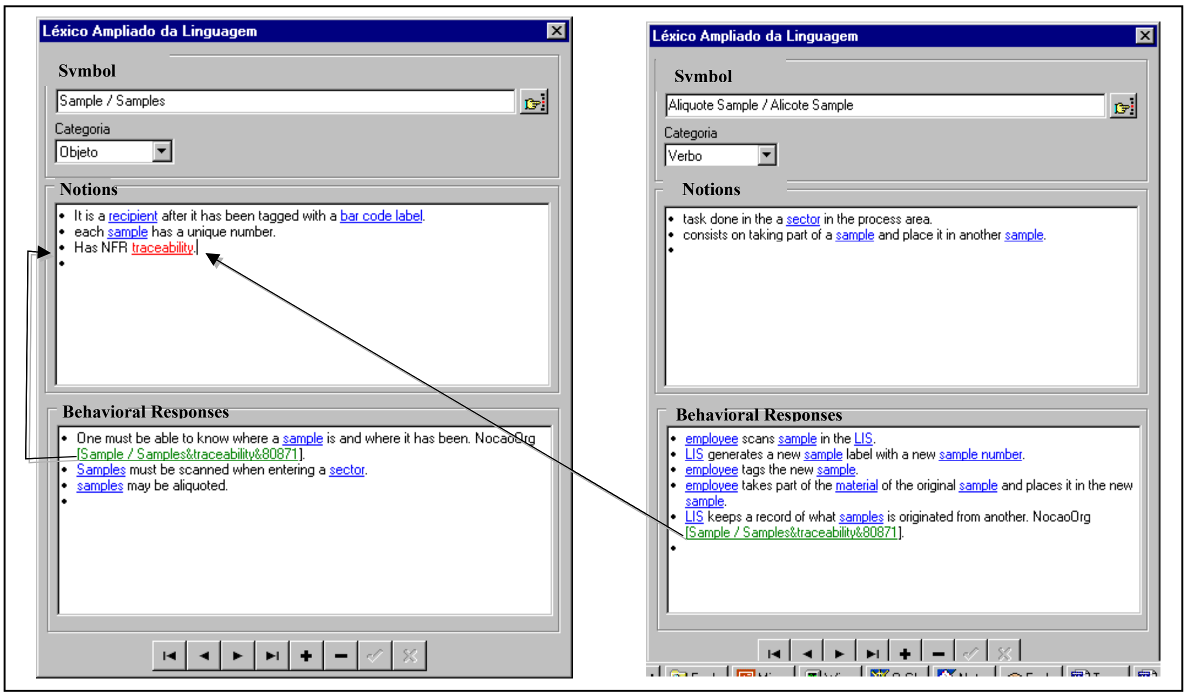
It is important to make it clear that the LEL is not restricted to holding information related to functional requirements. Its idea is to register the entire vocabulary in the UofD and therefore it might also include the non-functional aspects of the domain.

It is also important to say that LEL construction is very cost effective once before eliciting the requirements of a proposed system, one has to understand what the stakeholders are talking about.

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 NFRs and because quality aspects, in spite of its importance, are usually hidden in everyone's mind. However, it does not mean that the software engineer can not register information about NFRs. A well-defined set of symbols representing the vocabulary of the UofD is a key point to the strategy.

Building the non-functional view departs from the use of an existing LEL. In order to aid on this process we have extended the LEL to help NFR elicitation. The LEL is now structured to express that one or more NFR is needed by a symbol. It is also structured to handle dependency links among one NFR and all the notions and behavioral responses that are necessary to satisfice this NFR. Figure 2 shows these new features of the LEL.

We can see in Figure 2 that the symbol Sample belonging to a Clinical Analysis Laboratory information system, has in its notions the reference to the need of the NFR *Traceability*. It also shows that in order to satisfice this NFR we had to add one behavioral response to the symbol Sample itself (the one that states that one should be able to know where a sample is and where it has been). We also had to add another behavioral response to the symbol Aliquote Sample (the one that states that the system – LIS – should keep a record of what samples were originated from another sample).



Sample / Samples

Objeto

- It is a recipient after it has been tagged with a bar code label.
- each sample has a unique number.
- Has NFR traceability.

- One must be able to know where a sample is and where it has been. NocaoOrg [Sample / Samples&traceability&80871].
- Samples must be scanned when entering a sector.
- samples may be aliquoted.

Aliquote Sample / Alicote Sample

Verbo

- task done in the a sector in the process area.
- consists on taking part of a sample and place it in another sample.

- employee scans sample in the LIS.
- LIS generates a new sample label with a new sample number.
- employee tags the new sample.
- employee takes part of the material of the original sample and places it in the new sample.
- LIS keeps a record of what samples is originated from another. NocaoOrg [Sample / Samples&traceability&80871].

Figure 2 – Example of LEL Entries and its Extension to Deal with NFRs

We can also see besides these behavioral responses a pattern that was included to establish a dependence link between these behavioral responses and the notion that originated its need. This will help us to further represent the NFRs and its operationalizations. It also helps when, for any reason we need to update the LEL. For example, suppose that reviewing the LEL we decide now that Sample does not need any traceability any more. We should be able to easily see what notions and behavioral responses may be excluded from this symbol or from other symbols.

We have extended the OORNF tool [27] to support these extensions. This tool was originally developed to support the requirements baseline proposed by Leite [21] with some NFR support.

The tool has a knowledge base on NFRs that can and must be constantly updated. This knowledge base stores a variety of NFRs and some common way to decompose them. The tool brings along with the definition of these NFRs a set of possible conflicting NFRs as well as a set of NFRs that may be positively impacted by this NFR.

The OORNF tool also has support for entering the LEL, scenarios and CRC cards together with a support to create the scenarios from the LEL entries [17] and the CRC cards from the LEL entries and the elicited scenarios [23].

The first step on building the non-functional view will be to enhance the existing LEL with the NFRs that are desired by the stakeholders. To do that, we will run through all the LEL symbols using the knowledge base on NFRs present in the OORNF tools to ask ourselves and the stakeholder (whenever possible) if any of the NFRs present in this knowledge base may be necessary to each of the LEL symbols. Each NFR found may be represented in the symbol as shown in Figure 2.

After representing the need for an NFR the software engineer has to ask himself and the stakeholders what should be necessary to do to satisfice this NFR. Again, the knowledge base in the OORNF tool may be used. All the information that arises from this questioning may be represented in the LEL either in the same symbol or in another symbol if necessary.

Coming back to Figure 2 when we get to the symbol Sample (after examining all the previous symbol in the LEL), we asked the stakeholders and ourselves if any of the NFRs in the knowledge base would apply for samples. The answer was that traceability was essential once the laboratory could not afford to lose even one sample since drawing a new one could be sometimes almost impossible or at least very painful.

We then represented this NFR in the notion of the symbol Sample as seen in Figure 2 and started to ask how could we guarantee this traceability to work. One of the answers was that every time a sample is aliquoted (draw from one recipient to another) this procedure has to be documented in the software so one can know which sample was originated from another sample. We represented this answer as an entry in the behavioral responses of the symbol Aliquote sample and then established a dependency link between this behavioral response and the NFR traceability stated in the notions of the symbol Sample.

Although the LEL can now handle some representation of the NFRs and its impacts, it is not the best tool to deal with them in a more complete way, so one can reason about their interdependencies and conduct the necessary tradeoffs that arises during this process. To fully represent and reason with NFRs we propose to use the NFR Framework proposed by Chung [8] with some slights adaptations.

The NFR Framework [26][8][10] faces NFRs as softgoals 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 process continues until the requirements engineer considers the softgoal satisficed (operationalized) [10], so these satisficing goals can be faced as operationalizations of the NFR. Another way of understanding operationalizations is that they are, in fact, functional requirements that have arisen from the need to satisfice NFRs.

An NFR has a *type* which refers to a particular NFR as for example security or accuracy. It also has a subject matter or *topic*, for example room showed in Figure 2, which portrays an NFR graph which specifies a light control software extracted from one of our case studies.

The NFR framework was extended to represent the operationalizations in two different ways. The first one we called dynamic operationalizations. This type of operationalizations can be faced as those that ask for abstract concepts and usually calls for some action to be carried out. The second one we called static operationalizations and usually expresses the need for some data to be used in the design of the software to store information that is necessary for satisficing the NFR [12]. Figure 3 shows an example of an NFR graph where we can see these two types of operationalizations.

On the top of the Figure 3 we can see the node that represents the root of this graph represented as Safety[Room], meaning that room is a place that has to be safe regarding illumination aspects. One of the operationalizations that represents 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 that represents the minimum illumination in lux that can be used in a room. Some dotted circles appear on the bottom of the figure 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 can also be used for those ones that are Partially satisfied or D for those ones that are Denied. Further details can be seen in [12].

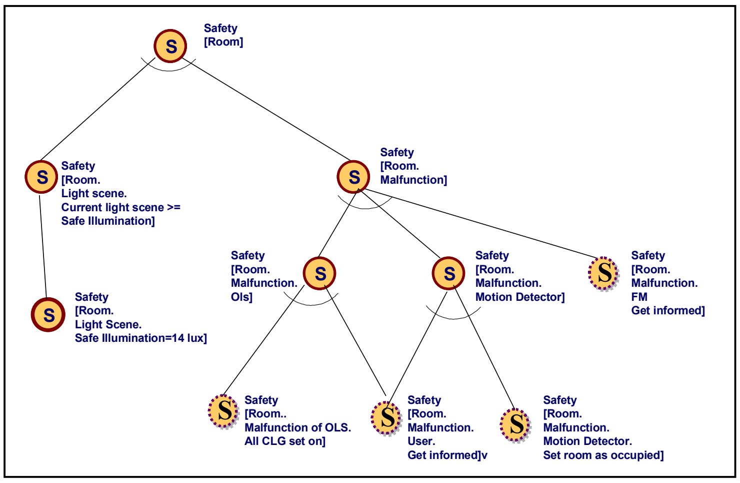

S
Safety [Room]
S
Safety [Room. Light scene. Current light scene >= Safe Illumination]
S
Safety [Room. Malfunction]
S
Safety [Room. Light Scene. Safe Illumination=14 lux]
S
Safety [Room. Malfunction. Ols]
S
Safety [Room. Malfunction. Motion Detector]
S
Safety [Room. Malfunction. FM Get informed]
S
Safety [Room.. Malfunction of OLS. All CLG set on]
S
Safety [Room. Malfunction. User. Get informed]v
S
Safety [Room. Malfunction. Motion Detector. Set room as occupied]

Figure 3 – An Example of a NFR Graph

It is important to call attention for the fact that the identifier that appears close to the NFR on the root of the graph (NFR Topic) has necessarily to be a symbol of the LEL. In Figure 3 we see that the root node is represented by Safety[Room], so room has to be a symbol of the LEL. If one can not find the word or sentence intended to be used as a topic for an NFR either one symbol represented in the LEL has an alias not defined or the LEL is incomplete and may therefore be updated.

To build the NFR model we will search every entry of the LEL looking for notions that express the need for an NFR. For each NFR found, we must create an NFR graph expressing all the operationalizations that are necessary to satisfice this NFR. At the end of this process we will therefore have a set of NFR graphs that will represent the non-functional aspects of the system.

Once we have this set of NFRs ready, we have to look for possible interdependencies, positive or negative, among NFRs. It is important to stick out that all the effort on NFRs tradeoffs due to positive and negative interdependencies will take place in the non-functional view, i.e., using the NFR framework. What will be integrated into the functional view will be the result that one gets after all the necessary reasoning on NFRs interdependencies and its consequences. We propose three heuristics to helps us on finding these interdependencies.

1- Compare all NFR graphs of the same type searching for possible interdependencies. For example, we may put all the NFR graphs that has the type Safety together to see if there is any interdependency among them
2- Compare all the graphs that are classified in the knowledge base [27] as possibly conflicting NFRs. For example compare graphs of Security with graphs of Usability. Frequently to add security to a system we have to sacrifice the usability of it.
3- Pair wise all the graphs that were not compared while applying the above heuristics.

Figure 4 shows an example of a negative interdependence found when we were applying the third heurist during one of our case studies.

This figure shows the pair wising of graph that deals with operational restrictions that applies for patient’s report with another one that deals with reliability of tests. Pair wising these graphs we could see that the sub-goal that deals with the aspects of how to assure reliability when electronically signing patient’s report has a negative influence on the operationalizations that states that in order to satisfice operational restrictions regarding time restrictions to print the patient’s report, the system (LIS) should electronically sign all the patient’s reports.

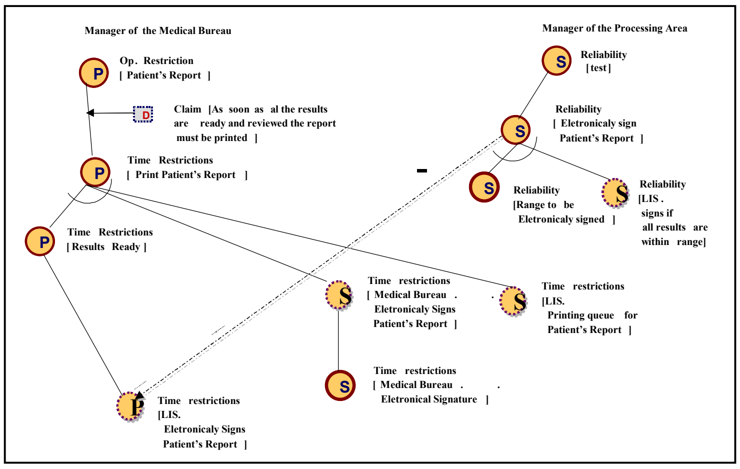

Manager of the Medical Bureau
Manager of the Processing Area
P
Op. Restriction [Patient’s Report]
D
Claim [As soon as al the results are ready and reviewed the report must be printed]
P
Time Restrictions [Print Patient’s Report]
P
Time Restrictions [Results Ready]
S
Reliability [test]
S
Reliability [Eletronicaly sign Patient’s Report]
-
S
Reliability [Range to be Eletronicaly signed]
S
Reliability [LIS . signs if all results are within range]
S
Time restrictions [Medical Bureau . Eletronicaly Signs Patient’s Report]
S
Time restrictions [LIS. Printing queue for Patient’s Report]
S
Time restrictions [Medical Bureau . Eletronical Signature]
P
Time restrictions [LIS. Eletronicaly Signs Patient’s Report]

Figure 4 – Identifying and Solving Interdependencies Among NFR graphs

It is important to clarify that the sub-goals that appear in Figure 4 as partially satisficed (P) were considered satisficed (S) before we carried out the pair wising. These sub-goals were considered partially satisficed only after we negotiated with the stakeholder to compromise with an intermediary approach. This approach is exactly what is represented in Figure 8, i.e., the patient’s reports will be electronically signed by the system *only* when all the results are within a predefined range considered safe to this task.

Although being important, heuristic three may become inappropriate to be used when the number of NFR graphs is very large. We do not have a rule-of-thumb to this number but we have used it in a case study where we dealt with more then 120 NFR graphs and it was still worth to use. During this case study the total overhead of using the proposal was 7% [13] and therefore we think it was quite worth to use it.

Further details on constructing the non-functional view can be seen in [13] and partially in [12].

To integrate the functional and non-functional views we propose a process to drive the operationalizations that are represented in the set of NFR graphs in the non-functional view to the functional view. The integration process can take place either in the early phases, integrating the NFRs into the use case or scenario models, or later, integrating NFRs into class, sequence and collaboration diagrams.

Notice that we do not propose to deal with NFRs in any of these models. NFR’s satisficing usually requires a very detailed reasoning to reach its operationalizations. Also, one NFR frequently presents many interdependencies with other NFRs. Dealing with these particular aspects of NFRs can be quite difficult to be accomplished directly in use cases, scenarios or even class diagrams. Thus, we propose to deal with NFRs in the NFR view using the approach we described to build the non-functional view, and then to integrate the operationalizations found into the models used in the functional view. We also propose a traceability mechanism to make it easier to navigate between models.

Due to the lack of space, this work will detail only the integration process that drives NFRs to the use cases and scenario models. Further details on integrating the NFRs into the class, state and collaboration diagrams can be seen in [13] and partially in [12].

3. The Integration Process

Once NFRs have been elicited and represented in the non-functional view, we have now to drive these NFRs to the functional view.

Our strategy allows the integration process to be carried out in many different ways. The requirements engineer can integrate the NFRs into the use case, scenarios and class models not necessarily in this order. The strategy also supports the integration of NFRs into class models that are already designed or even into legacy systems.

On the other hand, use cases and the scenario models are usually used to drive the construction of the class, sequence and collaboration diagrams. Therefore, if the requirements engineer drives the NFRs to the use case and scenario models in the early stages of the software development process, the conceptual models that will arise later will naturally reflect the NFRs.

Doing so, the integration of the NFRs to class diagrams will turn out to be a process that will work more as a validation of the integration made before. Still, as diagrams at this level (design) captures less abstract information then those used during earlier phases, some new designs decision may happen here despite of the previous integration of NFRs into use cases and scenario models.

This paper focus exactly on this point of the integration process, i.e., how to drive the NFRs to the use case and scenario models.

It is important to emphasize that we are much more concerned with NFRs through the viewpoint of the stakeholders then through the viewpoint of developers. The former type of NFRs tends to target specific parts of the software instead of being considered all over the software as some other do like efficiency, portability or maintainability. Notice that even these last ones may have particular instantiations in different parts of the software which will need to different use cases diagrams or scenarios depicting different conditions for these NFRs.

3.1 – Integrating NFRs into Use Cases

We will first tackle the use case diagrams. For every use case diagram in the functional view, we will identify if any symbols of the LEL appears in the name of the diagram (it usually does). We will also identify if any symbol of the LEL appears in any of the use cases of this diagram. For each symbol of the LEL that we find, we will search the set of NFR graphs for those where this symbol of the LEL appears. Once we find it, we have to see if the use case diagram realizes the operationalizations in the graph, i.e., if there is any use case that does what is stated in the operationalizations. The same has to be done for the actors of the use case diagram.

It is important to state here that, although we strongly recommend the use of the LEL, this process can be done without using the LEL as an anchor. To do that, one should see for every use case if there is any graph in the non-functional view semantically related to this use case. Of course, if the integration process is carried out this way, we will not have a process as smooth and precise as when we use the LEL.

Notice that as the LEL intends to capture every meaningful term used in the UofD, thus if a use case diagram does not have at least one LEL symbol either there are symbols in the LEL that may have aliases not yet specified or a symbol is missing.

Every use case or actor included, due to NFR satisfaction, must be followed by an expression using the pattern: ***{NFR_Type[NFR_topic]}.*** The use of this expression aims at adding traceability between the functional and non-functional views.

The traceability between the functional and non-functional view played a very important role during one of our case studies. NFRs have a very abstract nature and, in opposition to what happens in the case of functional requirements, it is hard to have them fixed in the requirements engineer's mind. This traceability link helps the requirements engineer while he is reviewing or changing the models. It helps, not only to see that some use case or episode is there to satisfy a specific need, but also to check during some change in the models if any conflict has arisen because of these changes.

Some especial conditions may be necessary to be specified together with a use case as for example **pre** and **post** conditions needed to satisfice an NFR. These conditions may be specified beside the use case name between braces, preferably using OCL [28].

Figure 5 shows a use case diagram used in one of our case studies. It portrays the diagram named "Input Results" from an information system for a clinical analysis laboratory before the integration process (first case study detailed on Section 4). This use case diagram expresses the set of use cases needed to input results to the tests that were prescribed to a patient. In this figure the actor LIS refers to Laboratory Information System.

Following the integration process, we have now to identify which are the symbols of the LEL in this diagram. The symbols are tagged as boldfaced and underlined words in the diagram. We now have to search in the set of NFR graphs for the occurrences of any of these symbols. Figure 6 portrays one of the NFR graphs we found. We picked out this graph because of the occurrence of the symbol "result" highlighted in Figure 5.

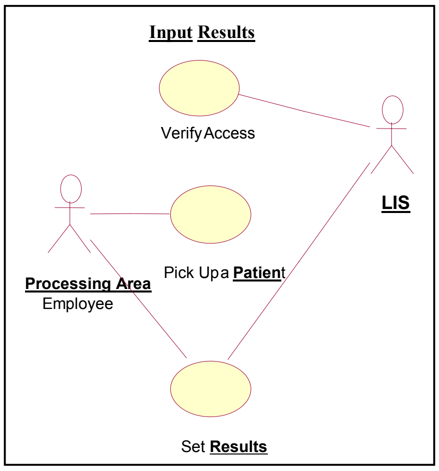


Figure 5 – An Use Case Before the Integration Process

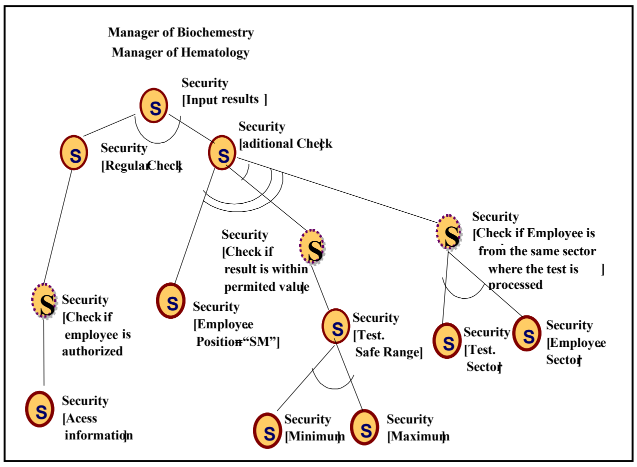

Manager of Biochemestry
Manager of Hematology
Security [Input results]
Security [Regular Check]
Security [aditional Check]
Security [Check if employee is authorized
Security [Acess information]
Security [Employee Position="SM"]
Security [Check if result is within permited value]
Security [Test. Safe Range]
Security [Minimum]
Security [Maximum]
Security [Check if Employee is from the same sector where the test is processed]
Security [Test. Sector]
Security [Employee Sector]

Figure 6 – NFR Graph to be Integrated

We can see in the graph portrait in Figure 6 that in order to satisfice the Security NFR, the system has to provide a regular access validation identifying whether the employee has access to this module of the system or not. It has also to provide additional checks as: to check if the employee is inputting a result for a test that is performed in the same sector where the employee work and to check if the result that the employee is inputting is within a range considered as safe to be inputted by regular employees. These two validations can be ignored if the employee holds a position as a sector manager (represented by "SM").

Analyzing the use case diagram in Figure 5, we can see that regular access validation was already satisfied in the use case diagram, while the other two had to be added. Figure 7 shows the same use case diagram after adding the necessary use cases and special conditions to the diagram.

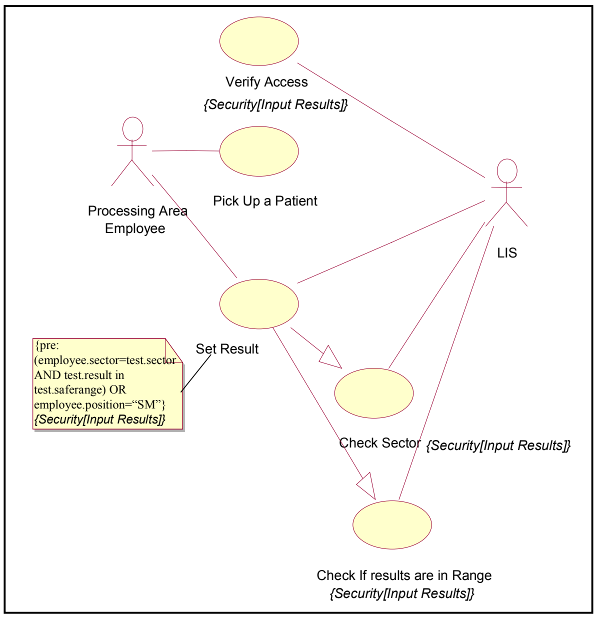

Verify Access
{Security[Input Results]}
Pick Up a Patient
Processing Area Employee
LIS
Set Result
{pre: (employee.sector=test.sector AND test.result in test.saferange) OR employee.position="SM"}
{Security[Input Results]}
Check Sector
{Security[Input Results]}
Check If results are in Range
{Security[Input Results]}

Figure 7 – Use Case Reflecting NFR

3.2 – Integrating NFRs into Scenarios

We also propose to integrate the NFRs from the non-functional view in the scenarios model. Here we face scenarios as an artifact for requirements elicitation that is not necessarily linked to any use case, although the integration process is still valid to be used when scenarios are faced as a refinement of a use case.

We use the scenario description proposed by Leite [22] to illustrate the process. Again, the integration process will be anchored in the use of LEL symbols. By taking one of the scenarios, we will identify what LEL symbols are stated in the title of the scenario, in the resources description, in the actors description or in the goal description. For each symbol found, we will search the set of NFR graphs looking for this symbol. Once we find a graph where the symbol appears, we have to check if all the operationalizations stated in the graph are already satisficed in the scenario description. If it is not, we have to update the scenario so that this condition holds.

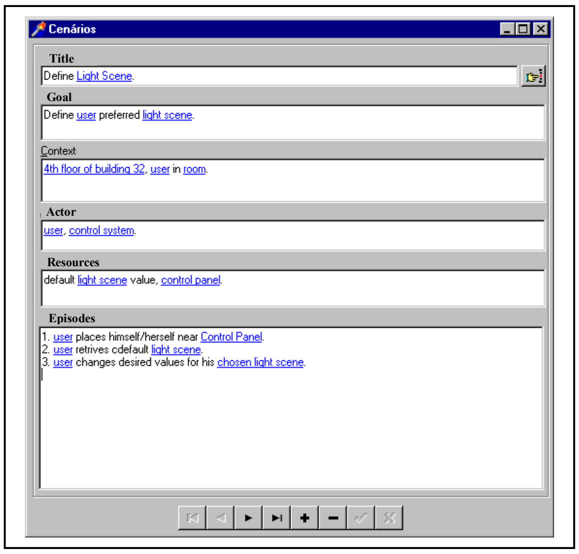
Every information included in a scenario to satisfice an NFR must be followed by the expression: *Constraint: **{NFR_Type[NFR_topic]}.***

We use this expression in the same way we use it in the use case diagrams. It aims at adding traceability between the functional and non-functional views.

This process was used in the second case study (detailed on Section 4), regarding a light control system, to integrate the NFRs into the scenarios. One of the scenarios used is shown in Figure 8.

All the expressions underlined in the scenario description are LEL symbols. Following the process, we searched the set of NFR graphs looking for the occurrence of graphs with the symbol "Light Scene". Figure 9 portrays part of the graph we found. As it can be seen here, the main graph illustrates that, in order to have usability in the software we must have a control panel that can be used to set a desired light scene.

Here again, the same as when dealing with use cases, if no LEL symbol can be found in a scenario then either there are symbols in the LEL that may have aliases not yet specified or a symbol is missing.



Title
Define Light Scene.

Goal
Define user preferred light scene.

Context
4th floor of building 32, user in room.

Actor
user, control system.

Resources
default light scene value, control panel.

Episodes
1. user places himself/herself near Control Panel.
2. user retrives cdefault light scene.
3. user changes desired values for his chosen light scene.

Figure 8 – A Scenario From a Light Control System

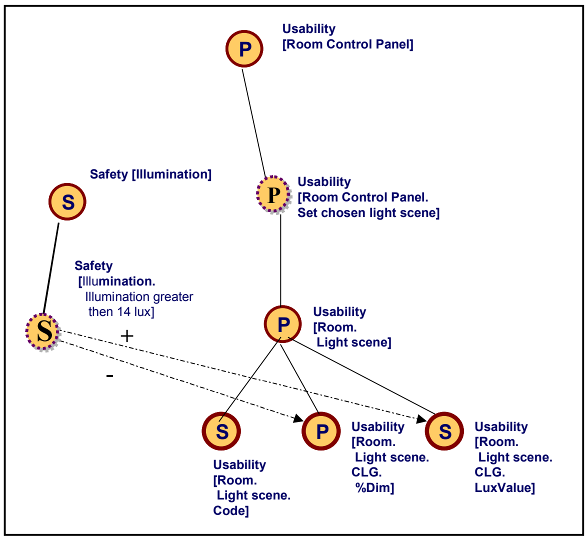
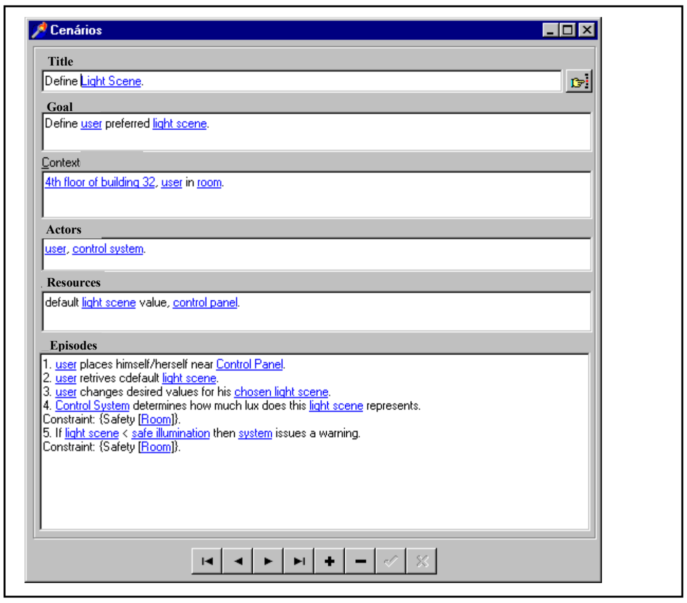

Usability [Room Control Panel]
Safety [Illumination]
Usability [Room Control Panel. Set chosen light scene]
Safety [Illumination. Illumination greater then 14 lux]
+
-
Usability [Room. Light scene]
Usability [Room. Light scene. Code]
Usability [Room. Light scene. CLG. %Dim]
Usability [Room. Light scene. CLG. LuxValue]
P
S

Figure 9 – Part of the NFR Graph Containing the Symbol "Light Scene"

To establish a light scene, the user will set how much the light must be dimmed. This information is by default passed in a percentage form.

We can see that this graph also demonstrates that in order to satisfice the NFR graph for Safe Illumination we have to store, not only the percentage value that represents how much you are dimming the lights, but also the equivalent in lux so that the system can check whether the user are inputting a safe value for the light scene or not.



Title
Define Light Scene.

Goal
Define user preferred light scene.

Context
4th floor of building 32, user in room.

Actors
user, control system.

Resources
default light scene value, control panel.

Episodes
1. user places himself/herself near Control Panel.
2. user retrives cdefault light scene.
3. user changes desired values for his chosen light scene.
4. Control System determines how much lux does this light scene represents.
Constraint: {Safety [Room]}.
5. If light scene < safe illumination then system issues a warning.
Constraint: {Safety [Room]}.

Figure 10 – The Resulting Scenario

Analyzing the scenario in Figure 8, we can see that this scenario is not in conformance with what is stated in the NFR graph. Thus, we have to add the necessary episodes to this scenario so it satisfices the NFR. Figure 10 shows the scenario after the integration process.

4 - Validating the Strategy

In order to validate the strategy presented here we carried out two case studies, both based on the use of project replication [1]. In both cases, we acted as a first team that developed the non-functional view to further integrate the NFR found into the use case or scenario models that were developed by the other team.

All the changes we made were faced as errors in the models that we were analyzing, once, if these changes had not taken place at that time, the software would had been constructed based on incomplete models, leading to incomplete softwares. Therefore, after software deployment, the stakeholder would have asked these changes to be done.

The first case study was conducted using the set of use case diagrams built during the development of an information system to a clinical analysis laboratory in a real world environment. Thirteen use case diagrams were changed in some way out of the 33 existing ones. A sum of 175 new use cases was used within these 33 diagrams before the integration process was carried out. We have added 71 new use cases within the 13 use case diagrams we changed.

The second case study was carried out using the scenarios generated by Breitman [6] as part of her Ph.D. thesis. She carried out a case study using the implementation of the Light Control System for the University of Kaiserslautern. This system was first proposed in a workshop on Requirements Engineering at Schloss Dagstuhl in 1998 [4]. There was a sum of 36 scenarios and we changed 16 out of them in one way or another. Originally, there were 95 episodes within these 36 scenarios. We have added 24 new episodes to satisfice NFR.

We were not capable of measuring the overhead for this specific process, but previous work [12] using a former version of the strategy proposed here has pointed out to a 10% overhead due to the use of the strategy. We believe that this overhead still holds. This previous work conducted a case study in a real life software development process and used three different teams during the development of the software where only one of them was using the strategy for dealing with NFR. The mentioned overhead took into account the amount of time spent since the beginning of the project to the point that the teams started coding the software. We estimated this overhead based on the fact that the amount of time used by the team using the proposed strategy to perform its tasks was 10% higher than the amount of time spent by either of the other two teams. Since each team was developing a separate part of the system, we cannot guarantee the overhead of 10%. Nonetheless, the team using the strategy was in charge of developing the most complex part of the software, having written 43% of the 570 KLoC that composes the software, as such, we believe our estimate to be fair one.

Confronting the number of changes we made both in the use case and scenario models with the expected overhead, we believe that these results suggest that the use of the strategy presented here can lead to a more complete set of requirements.

Considering that sooner or later these changes would have to be done to meet stakeholders' speciation, we think it is fare to say that finding and representing these requirements since the early phases of software development, as our strategy proposes, tends to lead to more complete software specifications. It will as well lead to a better time-to-the-market and less effort during maintenance phase once errors due to NFRs negligence will be significantly reduced and therefore we will spend less time correcting errors that are among the most difficult to fix [7] [12].

One could also argue that these benefits came just because we paid attention to NFRs and not from the use of our strategy. It may even be true, but as this is the only strategy formally defined and tested we know so far to deal with NFRs within Use Cases and Scenario, we could not measure our results against another one.

5. Conclusion

Companies are now facing an environment that calls for more complex business strategies and flexible structures in order to be adaptable to a changing world. This has to be supported by software that becomes each day more complex and hence much more difficult to specify. To deal with such complex conceptual models we have to be able, among other things, to deal with NFRs [Mylopoulos 95].

It has been shown that NFRs must be dealt with since the early stages of software development [9][10][12] and that errors due to not dealing with NFRs are among the most expensive and difficult to deal with [7][15][11][12].

We introduce in this paper a strategy to deal with NFRs and detailed the integration process that leads to get use case and scenario models that will reflect the NFRs.

We believe that if the models used in the early stages of software development already reflect the NFRs, we will get more complete and better quality conceptual models, and therefore, software that will closer meet stakeholders' expectations, better time-to-market.

Moreover, in accordance with previous studies [12] and with the results we got from our case studies, we believe that by doing so we can, not only get a more complete software specification, but also a more efficient process for software development as well as lower maintenance costs, This turns out to be true since we will avoid most of the errors due to NFR negligence that are usually very expensive and time consuming to fix [12].

Future work will focus on researching which artificial intelligence techniques can be applied to at least partially automate the integration process. We also intend to conduct more case studies, this time not using any of the authors, to measure how easily other developers can use the strategy.

6. Bibliography

[1] Basili, V.R. Selby, R.W., Hutchens, D.H. "*Experimentation in Software Engineering*" IEEE transactions on Software Engineering Vol. SE-12 No. 7 July 1986 pp733-742.

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

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

[4] Börger E., Gotzhein R. " *Requirements Engineering Case Study "Light Control""* http://rn.informatik.uni-kl.de/~recs.

[5] Breitman,Karin Koogan, Leite J.C.S.P. e 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.

[6] Breitman, K.K. *"Evolução de Cenários"* Tese de Doutorado submetida na PUC-Rio em Maio de 2000.

[7] Brooks Jr.,F.P. *"No Silver Bullet: Essences and Accidents of Software Engineering"* IEEE Computer Apr 1987, No 4 pp:10-19, 1987.

[8] 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.

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

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

[11] 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.

[12] 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 – to appear.

[13] Cysneiros, L.M. *"Requisitos Não Funcionais: Da Elicitação ao Modelo Conceitual"* Ph.D. Thesis submitted to PUC-Rio Computer Science Department in Feb. 2001.

[14] Castro, J., Kolp, M., Mylopoulos, J. *"Requirements Driven Development Methodology"* to appear in CaiSE´01.

[15] Davis, A. *"Software Requirements: Objects Functions and States"* Prentice Hall, 1993.

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

[17] Hadad, Graciela et. al. "*Construcción de Escenarios a partir del Léxico Extendido del Lenguage"* JAIIO'97, Buenos Aires, 1997, pp. 65-77.

[18] 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

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

[20] 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.

[21] Leite J.C.S.P., Oliveira, A.P.A., *"A Client Oriented Requirements Baseline",* Proc of the 2nd IEEE International Conference on Requirements Engineering, 1995.

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

[23] Leonardi, Carmen. et. al. "*Una Estrategia de Análisis Orientada a Objetos basada en Escenarios"* Actas II Jornadas de Ingeniaria de Software JIS97, Donstia, San Sebastian, España, Set. 1997.

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

[25] Musa, J., Lannino, A. and Okumoto, K. *"Software Reliability: Measurment, Prediciton, Application"* New York, McGraw-Hill, 1987

[26] Mylopoulos,J. Chung, L., Yu, E. 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.

[27] Neto, J.S.M. "I*ntegrando Requisitos Não Funcionais ao Modelo de Objetos"* M.Sc. Dissertation Submitted to the Computer Science Department of PUC-Rio, Mar/2000.

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