Towards a Tool to Help Exploring Existing Non-Functional Requirements Solution Patterns

Rodrigo Veleda  
School of Information Technology  
York University  
Toronto, Canada  
rveleda@yorku.ca

Luiz Marcio Cysneiros  
School of Information Technology  
York University  
Toronto, Canada  
cysneiro@yorku.ca

Abstract—Requirements Engineering play a crucial role during the software development process. Many works have pointed out that Non-Functional Requirements (NFRs) are currently more important than Functional Requirements. NFRs can be very complicated to understand due to its diversity and subjective nature. The NDR Framework has been proposed to fill some of the existing gaps to facilitate NFR elicitation and modeling. In this paper, we introduce a tool to help to explore stored knowledge for delivering solutions to implement quality requirements. Preliminary search mechanisms are provided in this tool to facilitate the identification of possible solutions to an NFR and its related consequences to other solutions and/or other NFRs.

Index Terms— Non-Functional Requirements, Reuse, Knowledge, Ontology.

I. INTRODUCTION

Non-Functional Requirements (NFR), also known as quality requirements, express constraints to be imposed on functional requirements as well as general quality properties that will apply to software. Examples of NFRs are Privacy, Usability, Reliability, Security, and Trust.

Despite many new works going from elicitation to verification and validation [1], there is still no convergence towards an amalgamated approach to deal with NFRs. NFRs are frequently fuzzy in nature, hard to identify, and it is even more difficult to elicit how to satisfice1 them. More important, adopting solutions to cope with one particular NFR frequently introduce synergies and most likely conflicts with other NFRs. Identifying these interdependencies is the most challenging task for effectively eliciting NFRs.

One way of representing knowledge on satisficing NFRs is to build catalogues using Softgoal Interdependency Graphs (SIG) [2]–[4], where softgoals represent NFRs and their refinements. Softgoals can be refined using an And/Or decomposition approach.

In a previous work [5], we detail an empirical study that suggests the use of NFR catalogues help identifying a larger and more appropriate set of NFRs that should be utilized in a project as well as a larger set of possible solutions to operationalize these NFRs.

Other works such as [6], [7] also conducted empirical experiments pointing out to catalogues helping to elicit NFRs in different situations.

However, all the above works recognize that the use of catalogues faces some challenges, among them, the scalability problem. Catalogues can grow quite fast and become difficult to visualize. Another problem relies on the ability to discover conflicts among solutions to different NFRs. More importantly, when dealing with catalogs for several NFRs at the same time, which is a common situation, even when these conflicts are mentioned in one catalogue, realizing the connection among these catalogs for such interdependencies is a visual task quite prone to mistakes and omissions.

To mitigate these problems, we started to investigate a framework that can at least semi-automate the acquisition of knowledge linking possible interdependencies (positive and negative ones) portrait in several different catalogues while at the same time providing an environment that can facilitate searches to guide requirements engineers to identify all possible NFRs and operationalizations that could be needed in a project. We understand that NFRs solutions will vary from one project to another and hence, the best approach is to provide a comprehensive set of options to guide a better cost vs. benefit analysis for each project. As such, our approach does not suggest one solution. Rather, it suggests a whole set of solutions to be studied as possible solutions to guide the cost vs. benefit analysis.

We have investigated the use of ontologies and semantic web techniques to represent SIGs in a machine readable format [8]. It aims to facilitate the reuse of the knowledge captured in SIGs. Veleda [9] introduced the requirements for the NFR Design and Rational (NDR) Framework and its supporting tool. The NDR Framework aims at capturing knowledge on how to develop software that will deliver appropriated solutions for needed NFRs.

We use SIGs to facilitate the visualization of this knowledge instead of using tables or other natural language models where interdependencies among different solutions would be hard to be modeled and understood. We view this knowledge expressed in the form of SIGs as being solution

1 We use the term satisfice in the same way as it is used by Mylopoulos denoting that an NFR is expected to be satisfied within acceptable limits.
patterns in a similar way that design patterns introduced by Gamma [10] played a significant role in developing better design solutions. Mangalaraj [11] presents an empirical study that strongly supports their claim that using visible artifacts such as Gamma design patterns can significantly positively impact the cognitive design and performance of a system. It also points out that patterns can adequately compensate for the absence of a collaborating partner. While Gamma patterns expressed common approaches to be used in coding, SIGs collect knowledge on satisfying NFRs portrait common solutions to satisfy NFRs.

The work presented in this paper presents the initial approach to provide a tool to help software engineers to retrieve knowledge stored in an ontology-based structure modeled following the NDR ontology. This Tool provides mechanisms to query the knowledge base using different levels of granularities to explore the search.

The paper is structured as follow: Section 2 tackles some of the related work while Section 3 briefly introduces the NDR tool as a whole and stresses its search capabilities for retrieving existing patterns of NFR solutions. Section 4 concludes this work.

II. RELATED WORK

SIGs were introduced by Chung [2] and have been used to describe quality attributes. Many works have used SIGs to represent knowledge related to NFRs such as [2], [3], [12]. Empirical works have suggested that the use of catalogues have a positive impact on the quality of the developed software [5], [7]. However, SIGs do not scale efficiently hence, the larger amount of knowledge we gather the more difficult it is to reuse it.

On a different path, Sancho et al. [13] suggested to use an ontological database and exemplified its use with the Software Performance Engineering Body of Knowledge. Their proposal consists of two ontologies both written in OWL [14]: the NFR ontology and the SIG ontology. The NFR Ontology describes the NFRs concept and relationships among them. The SIG Ontology depicts SIG constructs and their relationships. We have identified two shortcomings of this ontology. First, the SIG ontology does not define any class to describe the correlation interdependency between softgoals. Correlations model the impact of one NFR solution into other NFRs, and it is one of the most challenging goals of eliciting and negotiating NFRs satisfying. Second, it does not enforce the use of the proper kind of softgoals as parent and offspring of each refinement. A few other semantic-based approaches have been proposed in the NFR context. ElicitO [15] is an ontology-based tool that supports NFRs elicitation; it provides a knowledge base about NFR requirements and associated metrics. Najera proposes an approach also using OWL and RDF but targets only the representation of different $i^*$ variants [16]. NFRs reuse is not tackled in this work nor is cited as future work. Hu [17] proposed an ontology model as a baseline for modeling NFRs.

However, these works do not facilitate the required reasoning for a requirements engineer to analyze the tradeoffs involving different NFRs. Neither have they tackled the important aspect of creating methods to retrieve knowledge using searches with varying levels of granularity and filters to help visualize the retrieved knowledge. The NDR Framework aims at mitigating some of these gaps. The following sections will illustrate how the NDR Framework proposes to tackle these issues. More specifically we will focus on the first part of the framework targeting the elicitation and reasoning of NFRs solutions for a given project. The Tool presented in this work aims at introducing search and visualization mechanisms that can help software engineers to visualize innumerable alternatives (pre-stored patterns) to satisfy one NFR. Most importantly, the tool will also bring all other NFRs that might have positive or negative contributions to each solution retrieved as well as how each retrieved solution may positively or negatively affect other NFRs. We believe that identifying the interdependency properties among different solutions for different NFRs is one of the most challenging tasks to adequately eliciting NFRs.

III. THE NDR TOOL FOR SEARCHING NFRS

In this work, we present an initial approach to a tool based on the NDR Framework providing basic features to facilitate NFR elicitation. We departure from the idea that using Catalogues indeed help to elicit NFRs [5]–[7] but do present a challenge in dealing with the acquisition of knowledge to create these catalogues and moreover, how to store and represent its knowledge in such a way that retrieving all pertinent information can be done efficiently.

The NDR Framework uses the concepts for building SIGs from the NFR Framework, and it is centered on the use of ontologies and semantic web techniques to facilitate reuse. The use of techniques such as RDF schemas [18] and SPARQL [19] aims at exploring the possibility of having inference rules that can help identifying interdependencies among operationalizations that could easily go unnoticed otherwise. It may also facilitate the ability to explore different levels of granularity in queries to retrieve knowledge [9].

The Framework is based on the NDR Ontology [8]. The NDR Ontology allows describing well-formed SIGs in a machine-readable format allowing requirements engineers to explore the rationale knowledge embedded in SIGs.

However, we need to provide an environment that can hide all the details of handling ontologies concepts and querying it. The NDR Tool was initially developed to treat domain-independent catalogues of NFRs together with the interdependencies among these catalogues. Later, it will be extended to allow these catalogues to be instantiated into domain-specific alternatives. It is important to note that the NDR Framework does not intend to propose one single solution to satisfice one or more NFRs. One solution that could satisfy privacy expectations in one company may not fit into another company policy. Hence, the NDR Framework aims at recovering as many as possible alternatives so developers can choose the one that fits better to their project.

It is important to emphasize that in the present work we are only exploring the Tool for retrieving the existing knowledge stored in the NDR repository. The process of acquiring this
knowledge and storing it in the knowledge base will be tackled in future work.

### A. Architecture

Figure 1 illustrates a high-level architecture of the NDR Framework. As it is noticeable, the NDR Tool plays a significant role within the framework, acting as a knowledge cortex. The NDR Tool is responsible for concentrating the essentials regarding NFR information, incorporating knowledge importation, searching, and visualization. Thus, prospective techniques proposed within the NDR Framework such as Integration Methods and Reuse Techniques involving Doors and i*

![Fig. 1. NDR Framework's architecture overview](image)

The tool contains two repositories: Knowledge and Ontology. The knowledge repository stores the NFR information extracted from SIG catalogues, maintaining the knowledge evolution associated with a particular ontology model. On the other hand, the ontology repository holds different ontology models. Consequently, within this approach, we envision to provide support to multiple ontologies that are also capable of representing NFR and design rationale knowledge, other than limit the Tool operation solely to the NDR ontology.

To fulfill its functioning and achieve needed requirements, the NDR Tool incorporates multiple technologies. We mainly adopted alternatives that have an open-source implementation and an active community support.

PlayFramework\(^2\) represents the baseline for the Tool's architecture. It works as the main engine for handling every request within our platform. Apache Jena\(^3\) provides certain functionalities for dealing with Semantic Web needs at a programming level. Similarly, Apache Fuseki\(^4\) offers an SPARQL server, capable of handling SPARQL queries over resources and individuals of an ontology model. Both Apache Jena and Fuseki are in charge of managing the ontology needs within the NDR Tool. Lastly, GraphViz\(^5\) represents the primary component responsible for achieving and producing the graphical output based on the results generated by the SPARQL queries.

In regards to knowledge acquisition, the NDR Tool applies inference rules to assure the constraints of the NFR Framework [2]. Since the NDR ontology follows the NFR Framework definitions, our approach will enforce its restrictions by using predefined inference rules during the importation of SIGs into the knowledge repository. The importation process is already defined and partially implemented, but it is not the focus of this work due to space limitation. The focus of this work is the process of retrieving solution patterns that were stored during the importation process. Hence, we only briefly introduce the importation process as it stands right now but will not detail it.

Figure 2 demonstrates a partial output of the knowledge acquisition process. A decomposition between Transparency and Informativeness illustrated by a SIG imported into the system is now represented in OWL format and instantly part of the NDR ontology as a new individual.

### B. Searching and Visualizing NFR Knowledge

The NDR Tool provides two main possibilities for searching specific NFR Knowledge across the repository:

- Selection of one specific NFR
- Input of a custom search term

Suppose our knowledge base has acquired the knowledge for the NFRs of Security, Privacy, Usability, Traceability, and

![Fig. 2. NDR Ontology update output](image)

---

\(^2\) [https://www.playframework.com/](https://www.playframework.com/)

\(^3\) [https://jena.apache.org/](https://jena.apache.org/)

\(^4\) [https://jena.apache.org/documentation/fuseki2/](https://jena.apache.org/documentation/fuseki2/)

\(^5\) [http://www.graphviz.org/](http://www.graphviz.org/)
Transparency. Clicking on the “Choose a NFR” menu illustrated in Figure 3 would bring a list with these five NFRs for the requirements engineer to choose. After selecting one of them, the tool will then display the SIG that represents the chosen NFR together with a list of correlated catalogues. Based on the constraints of the NFR Framework, the tool uses two distinct colors to reason about the nature of an association between two elements. Therefore, it applies the Green color when an association has a positive nature and employs the Red color when an association has a negative nature. For instance, a decomposition that may help to *satisfice* a given softgoal is highlighted in Green color. On the other hand, a correlation that may hurt *satisficing* a particular softgoal is represented using red color.

As a consequence of demonstrating the occurrences of a queried element, the NDR Tool may infer existent correlations when a parent NFR is utilized as a query term. Figure 3 illustrates this scenario. After searching for Security, the tool graphically outputs the complete existent knowledge associated with Security with subgoals such as Confidentiality, Availability, and Integrity. It can also be seen that Security presents correlations with three other NFRs Privacy, Traceability, and Usability. To find out whether the correlation produces a synergy or a conflict, one can simply click on one of the listed NFRs and visually verify the nature of the correlation. Figure 4 demonstrates a partial graphical output generated after selecting Privacy from the list of occurrences. It is possible to notice the details of the correlation between Security and Privacy. In this case, there are multiple correlations with a negative nature, characterizing a potential conflict among Security and Privacy. For instance, the use of Smart Cards may somehow help *satisfice* Privacy. However, its usage also brings a negative interaction with Security, characterizing a conflict.
Notice that as it can be seen in the upper-left part of the screen in Figure 4, the Tool allows one to zoom in or out the retrieved SIG. It also allows to move the SIG up, down, left or right hence, although one can often only visualize part of the resulting SIG at a time he/she can navigate the whole SIG. At this first version, we directed our efforts on developing a basic
user interface focused mainly on the ability to browse and explore the existent knowledge, successfully highlighting the possible solutions for *satisficing* a particular NFR and how these solutions may affect other NFRs.

Under a similar approach, after selecting Traceability from the drop down menu, the tool outputs the knowledge built on top of the NDR Ontology, demonstrating the possible solutions patterns for *satisficing* Traceability. Additionally, the system highlights the idea that Traceability also correlates with Transparency. Figure 5 illustrates this mentioned scenario.

Furthermore, after clicking on Transparency from the usage list, a requirements engineer would be able to notice that Traceability plays a major role in *satisficing* Transparency. In fact, more than just correlating with Transparency, it is possible to notice that Traceability is part of a decomposition of a positive nature. In other words, the graphical output provided by the tool demonstrates that Traceability may be considered in the context of a possible solution pattern for *satisficing* Transparency under a particular context. Figure 6 depicts this scenario. Notice that Traceability, as well as Security in Figure 4, is painted in a Blue color to call the attention for the correlation relationship.

**To explore different levels of granularity regarding how solutions are developed and modeled, the tool provides the capability to use a custom search term. Using the search box in the previous figures will trigger the tool to look for any particular element that contains the term provided. The NDR Tool performs a non-restrictive search for custom terms. Therefore, it will produce a list with all the occurrences of this term in different SIGs.**

Figure 7 illustrates this mentioned scenario. Let us suppose one wants to investigate where disclosure of data may be an issue. Filling up the search box with the term “disclosure” will trigger the tool to produce a series of SPARQL queries. These queries are then executed on the SPARQL server side in a recursive manner. Hence, the system can retrieve every interdependency associated with each element related to the search. Figure 8 depicts an example of two SPARQL queries: the former is produced and applied by the tool during the search term process, and the latter is employed just before the tool begins with the drawing process. As a result, the system outputs a partial graphic visualization starting from the chosen search possibility, in this case, five possible operationalizations as demonstrated in Figure 7. If necessary, the user can still select among the subgoals/operationalizations associated with the term “disclosure” one to be further explored. Figure 9 illustrates the partial graphic output originated from clicking on the Reduce Need For Personal Data Disclosure. The custom search term feature also informs the occurrences of a specific element across the knowledge repository. Hence, under this particular scenario, one can simply refer and click on the provided usage list in order to verify the level and nature of Reduce Need For Personal Data Disclosure among the possible solution patterns that tackle some form of Disclosure.

Another example of searching for a custom term is demonstrated in Figure 10. Even when searching for a major element (e.g. Usability, in this case), the tool is consistent and provides the output in the same manner as the one generated in the previous example since the search mechanism relies on the execution of SPARQL queries recursively as a whole. By selecting Usability from the result list, one will visualize the existent alternative solutions for *satisficing* Usability according to the knowledge repository. This behavior is illustrated in Figure 11.
It is important to mention that the search capability provided by the NDR Tool works on different levels of granularity. In other words, a requirements engineer can search for any NFR related capabilities (subgoals or operationalizations), ranging from early refinements to a very specific refinement level. This characteristic provides a versatile mechanism for scenarios where the granular level of a needed NFR related solution is unknown.

Any correlations are displayed, as it can be seen in the above figures, using a type of hyperlink between solution patterns where the requirements engineer can navigate through those patterns. Put that together with the non-restricted search capability as well as the search for specific NFRs and requirements engineers have a powerful and flexible search mechanism at hand. It helps, although it does not solve, the scalability problem allowing one to search for one specific NFR and yet be directed to look where a specific solution will hurt or help another solution either for this NFR or another. By the same token, the search by substring where the tool searches the complete knowledge base for all the occurrences of this substring helps to find information in a huge SIG. We believe that such kind of Tool facilitates the reuse of solutions patterns previous elicited helping requirements engineers to evaluate many different possible alternatives to comply with stakeholder’s demands. We understand that each project will face different challenges, eventually addressing different types of users and therefore stakeholders, in general, may need different solutions for different domains and even for different companies in the same domain. That is the main reason why our SIGs do not evaluate which alternative would be better to use nor do we weigh different solutions against each other. We aim at providing a general approach that can be used in various projects regardless the type of development process being used or the domain in which it is being used.

IV. CONCLUSIONS

This paper introduces an approach to reuse NFR knowledge stored in the form of solution patterns represented by SIGs.
We introduce the NDR tool as a support mechanism based on a previously defined ontology [8] to support the acquisition of knowledge to satisfice NFRs while allowing queries to be made on this knowledge base. In this work, we focused on the search capabilities provided by the NDR Tool. It provides a retrieval approach that allows the requirements engineer to query the existent knowledge base using different levels of granularity as well as different ways to search and visualize results.

Future work will involve conducting in-vitro experiments to evaluate the utilization of the tool. The process of acquiring knowledge introduced briefly in this work will be further explored to be efficiently semi-automated. We also intend to carry out qualitative experiments to identify possible angles to amplify the ability to provide different levels of granularities for searching solutions and Filter that might mitigate the complexity of some resulting SIGs. We also aim at investigating mechanisms to tackle the problem of expressing one NFR or operationalization using different names/expressions. Finally, we also intend to expand the functionalities regarding knowledge visualization, implementing features that can promote an enhanced browsing experience.

On a longer perspective, we will investigate how to facilitate the integration of possible solutions into models depicting the entire solution for the problem.

REFERENCES


