Towards a Knowledge-Based Representation of Non-Functional Requirements

Mohamad Kassab
The Pennsylvania State University
Malvern, Pennsylvania, U.S.A
muk36@psu.edu

Ghizlane El-Boussaidi
École de Technologie Supérieure
Montreal, Canada
ghizlane.elboussaidi@etsmtl.ca

Abstract— **Knowledge-based representation is necessary to support the description of Non-Functional Requirements within a system and to provide practitioners and researchers with a valuable alternative to current requirements engineering techniques. The aim of our research reported in this paper is to systematically develop an ontology which provides the definition of the general concepts relevant to NFRs without reference to any particular application domain. The general concepts can then act as a common foundation for describing particular non-functional attributes as well as providing a conceptual model for NFRs (including, e.g., entity definitions, relations, etc.). The ontology also contains rules which define the semantics of the defined concepts.**

Keywords- non-functional requirements; ontology; software architecture; quality.

I. INTRODUCTION

The IEEE-830: "Guide to Software Requirements Specifications" [1] defines a proper requirements specification as being: unambiguous, complete, verifiable, consistent, modifiable, traceable, and usable during operations and maintenance. To help achieving this, the requirements elicitation process should consider: (1) the functional requirements which are associated with specific functions, tasks, or behavior that the system must support and (2) the non-functional requirements (NFR).

Existing NFRs elicitation methods adopt memo of interview transcripts to collect initial NFRs and then construct systems with the NFRs integrated according to the experience and intuition of the designers [2]. However, empirical reports [3, 4, 5] indicated a number of drawbacks when using these methods. For example, a significant portion of NFRs may be neglected as it is difficult to ask users to provide their NFRs explicitly because they are always related to specific domains and affected by context. Furthermore, NFRs can often interact, in the sense that attempts to achieve one NFR can help or hinder the achievement of other NFRs at certain functionality. Such an interaction creates an extensive network of interdependencies and trade-offs between NFRs which is not easy to describe [6]. In addition, the current methods don't provide sufficient answers on how the NFRs should be accommodated at later stages of the development (e.g., software architecture).

The growing awareness of these issues among the requirements engineering (RE) community in the last few years led to a heightened interest in NFRs description and modeling and, in turn, to the emergence of several models intended to capture and structure the more relevant concepts defining the NFRs and their relations. Such models are generic ones and must be instantiated to be usable for specific domains or applications. Yet, the instantiation process is not easy to perform since the generic models usually do not contain sufficient information about NFRs interdependencies [7]. Some standards have been proposed in order to unify the definition of subsets of NFRs; e.g., software quality concepts [8]. However, till now there is no clear and coherent generic representation of the NFRs concepts.

On the other hand, the growing interest in ontology-based applications as opposed to systems based on information models have resulted in an increasing interest in the definition of conceptual models for any kind of domain. Software Engineering is one of those domains that have received high attention in that respect [9, 10, 11]. Current research studies by knowledge engineering scholars on requirement acquisition, for example, use domain ontology to support software requirements description [12, 13, 14]. These studies leverage the existing knowledge of the relationship between the software requirements and the information in the related domain. According to this relationship, the domain knowledge influences the result of requirements acquiring [2]. International Software Engineering standards such as IEEE [15] provide a foundation for the development of ontology for software engineering in terms of common vocabulary and concepts. Nonetheless, the process of analysis of the standards to come up with a logical coherent ontology is by no means a simple process [10]. Moreover, NFRs have received little or no attention from the ontology research groups due to inherent challenges imposed by the semantic imprecision of NFRs conceptual schemas [10].

Building on the above discussion, a knowledge-based representation is necessary to support the description of NFRs within a system and to provide practitioners and researchers with a valuable alternative to current requirements engineering techniques. In [16], a systematically developed ontology which provides the definitions of the general concepts relevant to NFRs was presented. The aim of our research reported in this paper is to present an updated version of the NFRs ontology with: 1) updated and more comprehensive rules which define the semantics of the defined concepts; and 2) an extension to the NFRs' refinements relating to software architecture concepts without reference to any particular application domain. The general concepts can then act as a common foundation for describing particular non-functional attributes as well as providing a conceptual model for NFRs (including e.g., entity definitions, relations, etc.).

The paper is organized as follows. Section II provides the background on ontologies and the Web Ontology Language. We describe in details the NFRs ontology in Section III. Section IV evaluates the NFRs ontology. Section V discusses related work and finally, Section VI concludes the paper.

II. ONTOLOGIES IN SOFTWARE ENGINEERING

The software engineering community has recognized ontologies as a promising way to address current software engineering problems. Researchers have so far proposed many different synergies between software engineering and Ontologies. For example, ontologies are proposed to be used in requirements engineering, software modeling, model transformations, software maintenance, software comprehension, software methodologies, and software community of practice.

Ontology can be defined as "*a specification of a* conceptualization" [17]. More precisely, ontology is an explicit formal specification of how to represent the objects, concepts, and other entities that exist in some area of interest and the relationships that hold among them. In general, for ontology to be useful, it must represent a shared, agreed upon conceptualization. The use of ontologies in computing has gained popularity in recent years for two main reasons: i) they facilitate interoperability and ii) they facilitate machine reasoning. In its simplest form, ontology is taxonomy of domain terms. However, taxonomies by themselves are of little use in machine reasoning. The term ontology also implies the modeling of domain rules. It is these rules, which provide an extra level of machine "understanding".

Holsapple [18] describes a number of approaches to ontology design: inspiration, induction, deduction, synthesis and collaboration. We chose to follow the deductive approach. Deductive approach to ontology design is concerned with adopting some general principles and adaptively applying them to construct an ontology geared toward a specific case. This involves filtering and distilling the general notions so they are customized to a particular domain subset. It can also involve filling in details, effectively yielding an ontology that is an instantiation of the general notions.

The constructs used to create ontologies vary between ontology languages. One class of ontology languages is those which are based upon description logics [19]. OWL is one such language. OWL [20] is the Web Ontology Language, an XML-based language for publishing and sharing ontologies via the web. OWL originated from DAML+OIL both of which are based on RDF (Resource Description Framework) triples. There are three 'species' of OWL – but the most useful for reasoning - OWL-DL - corresponds to a description logic. Editing OWL manually can be equally difficult for the very same reason. We used Protégé and its OWL plug-in for NFRs ontology development.

OWL ontology consists of Classes; also referred to by concepts, and their Properties; also referred to by relations. The Class definition specifies the conditions for individuals to be members of a Class. A Class can therefore be viewed as a set. The set membership conditions are usually expressed as restrictions on the Properties of a Class. For instance the *allValuesFrom* and *someValuesFrom* property restrictions commonly occur in Class definitions. These correspond to the universal quantifier ($\forall$) and existential qualifier ($\exists$) of predicate logic. More precisely, in OWL such restrictions form anonymous Classes of all individuals matching the corresponding predicate. A key feature of OWL and other description logics is that classification (and subsumption relationships) can be automatically computed by a reasoner which is a piece of software able to infer logical consequences from a set of asserted facts or axioms. For the purpose of the NFR ontology, we will use a semantic web reasoning system and information repository Renamed Abox and Concept Expression Reasoner (RACER) [21].

III. NFRS ONTOLOGY

Most of the terms and concepts in use for describing NFRs have been loosely defined, and often there is no commonly accepted term for a general concept [22]. As indicated in the Introduction, common foundation is required to enable effective communication and to enable integration of activities within the RE community. This common foundation is realized by developing an ontology, i.e. the shared meaning of terms and concepts in the domain of NFRs.

There are many resources for setting up a glossary for NFRs. In addition, there are many different perspectives from where NFR terms are defined, (e.g., NFRs in product-oriented perspective vs. process-oriented perspective [6]). In this paper, the NFRs glossary is developed based on commonality analysis and generalization from the previous publications in requirements engineering and software engineering communities.

The NFRs ontology has an important core about NFRs model, but also addresses areas such as software architectures. It contains many concepts. In order to cope with the complexity of the model we use views of the model. A view is a model which is completely derived from another model (the base model). A view cannot be modified separately from the model from which it is derived. Changes to the base model cause corresponding changes to the view [23]. Three views of the NFRs ontology are identified: The first view concerns the NFRs relation with the other entities of the software system being developed (intermodel dependency view). The second contains the classes and properties intended to structure NFRs in terms of interdependent entities (intramodel dependency). The third view represents the measurement process and contains the concepts used to produce measures to measurable NFRs. The measurement view will not be discussed in this paper as it maintains the same structure from the earlier version of the NFRs ontology [16].

A. Intermodel Dependency View

Figure 1 illustrates the structure of the NFRs intermodel dependency view by means of a simplified UML class diagram. The core of this structure relies on the fact that NFRs are not stand-alone goals, as their existence is always dependent on other concepts in the project context. If a requirement is a member of the class NonFunctionalRequirement, it is necessary for it to be a member of the class Requirement and it is necessary for it to be a member of the anonymous class of things that are linked to at least one member of the class AssociationPoint through the hasAssociationPoint property. On the other hand, isAssociatingNfrTo links the AssociationPoint to a range of: FunctionalRequirement union Element union Process union Product union Resource. The AssociationPoint can be thought of as an interface from the perspective of the association to the individuals from the above range. Thus, if an individual is a member of the AssociationPoint Class, it is necessary for it to be linked to one and only one individual from: the (FunctionalRequirement class through the isAssociatingNfrTo property) OR (Element through isAssociatingNfrTo property) OR (Process through isAssociatingNfrTo property) OR (Product through isAssociatingNfrTo property) OR (Resource though the isAssociatingNfrTo property).

An individual from AssociationPoint class can be linked to many individuals from the NonFunctionalRequirement class through hasAssociationPoint property.

1) Association to FR (or derived elements)

Functionality-related NFRs refer to individuals instantiated from the NonFunctionalRequirement class and that participate in hasAssociationPoint relation with an individual from the AssociationPoint class which in its turn participates in isAssociatingNfrTo relation with an individual from the FunctionalRequirement class (see Figure 1). In fact, a subset of NFRs, namely functionality quality requirements, is defined with an existential restriction to have at least one association point with FR as it represents a set of attributes that bear on the existence of a set of functions and their properties specified according to the ISO 9126 definition to the functionality quality [8]. Valid example of functionality-related NFRs is: “the interaction between the user and the software system while reading email messages must be secured”.

The FunctionalRequirement class is further specialized into PrimaryFunctionalRequirement and SecondaryFunctionalRequirement . A NFR can be associated to either type of FRs.

Functional Requirement is further realized through the various phases of development by many functional models (e.g., in the object-oriented field, a use-case model is used in the requirements analysis and specification phase, a design class model is used in the software design phase, etc.). Each model is an aggregation of one or more artifacts (e.g., a use-case diagram and a use-case for the use-case model, a domain model diagram and a system sequence diagram for the analysis model, a class diagram and a communication diagram for the design model). The artifact by itself is an aggregation of elements (e.g., a class, an association, an inheritance, etc. for the class diagram). Modeling artifacts and their elements in this way gives us the option of decoupling the task of tracing NFRs from a specific development practice or paradigm.

If an NFR is associated with functionality, then some or all the offspring elements that refine this functionality will inherit this association. More specifically:

$((\mathrm{NFR}_i \text{ hasAssociationPoint } \mathrm{AssociationPoint}_j) \wedge (\mathrm{AssociationPoint}_j \text{ isAssociatingNfrTo } \mathrm{FunctionalRequirement}_k)) \Longrightarrow \exists\ \mathrm{Element}_n\ ((\mathrm{NFR}_i \text{ hasAssociationPoint } \mathrm{AssociationPoint}_m) \wedge (\mathrm{AssociationPoint}_m \text{ isAssociatingNfrTo } \mathrm{Element}_n) \wedge (\mathrm{FunctionalRequirement}_k \text{ FrIsMappedInto } \mathrm{Element}_n))$

When hasAssociationPoint property links an individual NFR to an individual AssociationPoint which is further linked to an individual FunctionalRequirement or Element through isAsscoatingNfrTo property, then the AssociationPoint can be further specified through one of three subclasses. These subclasses specify the type of association between an individual from the NonFunctionalRequirement class and an individual from the FunctionalRequirement and Element classes. We adopt the concepts of overlapping, overriding and wrapping, commonly used in various separations of concerns approaches [24] to define these three subclasses:

- Overlapping: the NFR requirement modifies the FRs it transverses. In this case, the NFR may be required before the functional ones, or it may be required after them. For example, the implementation of security requirement (e.g., user’s authorization) needs to be executed before the user can access “read email messages” functionality.
- Overriding: the NFR superposes the FRs it transverses. In this case, the behavior described by the NFR substitutes the FRs behavior.
- Wrapping: NFR “encapsulates” the FRs it transverses. In this case, the behavior described by the FR is wrapped by the behavior described by the NFRs.

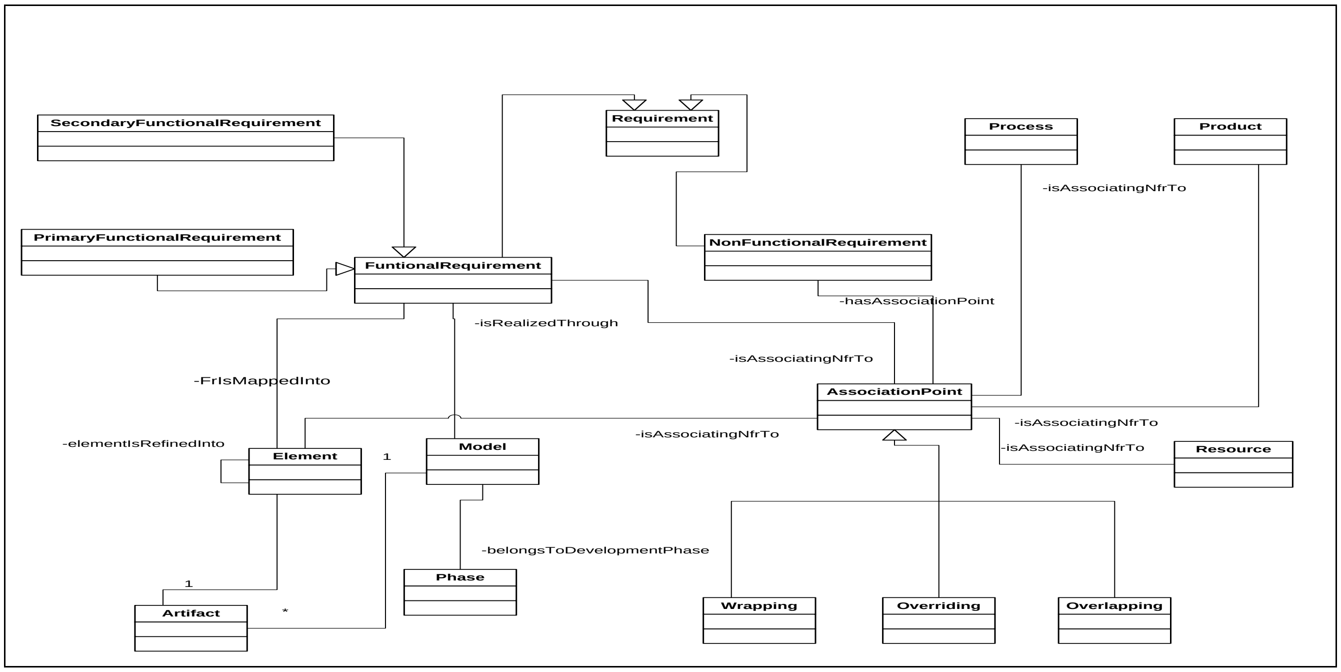

Figure 1. NFRs Intermodel Dependency View.

2) Association to Process

A software development process is a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

From the above definition to the software process, process-related NFRs specify concerns relative to the scope of the development process. Examples of such NFRs are "The project will follow the Rational Unified Process (RUP)" and "Activities X, Y, Z will be skipped for this project".

3) Association to Product

Product-related NFRs refer to those NFRs which have a global impact on the system as whole. Example of such NFRs are: "The system should be easy to maintain".

4) Association to Resource

Resources serve as input to the processes used on a project. They include people, tools, materials, methods, time, money, and skills [25]. An example of an NFR associated with a resource is illustrated through a requirement like "The software maintainers should have at least 2 years of experience in Oracle database." This is an operating constraint that is associated with candidates for the maintenance position for the system (another type of resources).

It is to be noted that the inter-relationships among the above five concepts (e.g., the relation between the product and the process) is out of the scope of this paper.

B. Intramodel Dependency View

The intramodel dependency view is concerned with the refinement of NFRs into one or more offspring; through either decomposition or operationalization, and the correlation among the concepts of the NFRs model. The

view is depicted in the UML class diagram in Figure 2 and it is discussed through the concepts and properties referring to: NFRs type, NFRs decomposition, NFRs operationalization and NFRs interactivity.

1) NFRs Type

Specifying NFR through types is a particular kind of refinement for NFRs [6]. This allows for the refinement of a parent on its type on terms of offspring, each with a subtype of the parent type. Each subtype can be viewed as representing special cases for the NFR. Five subclasses are identified as a candidate for the root node for an NFR type refinement hierarchy; namely, QualityRequirement, DevelopmentConstraint (e.g., implementation language constraint, constraints on system architecture), EconomicConstraint (e.g., allocated budget), OperatingConstraint and PoliticalCulturalConstraint (e.g., law imposing to support bilingual system user interface). These in fact are not mutual exclusive classes.

A special type of Development constraints is the architectural concern which presents an architectural requirement on the system under development. A concern is an area of interest or focus in a system. Concerns are the primary criteria for decomposing software into smaller, more manageable and comprehensible parts that have

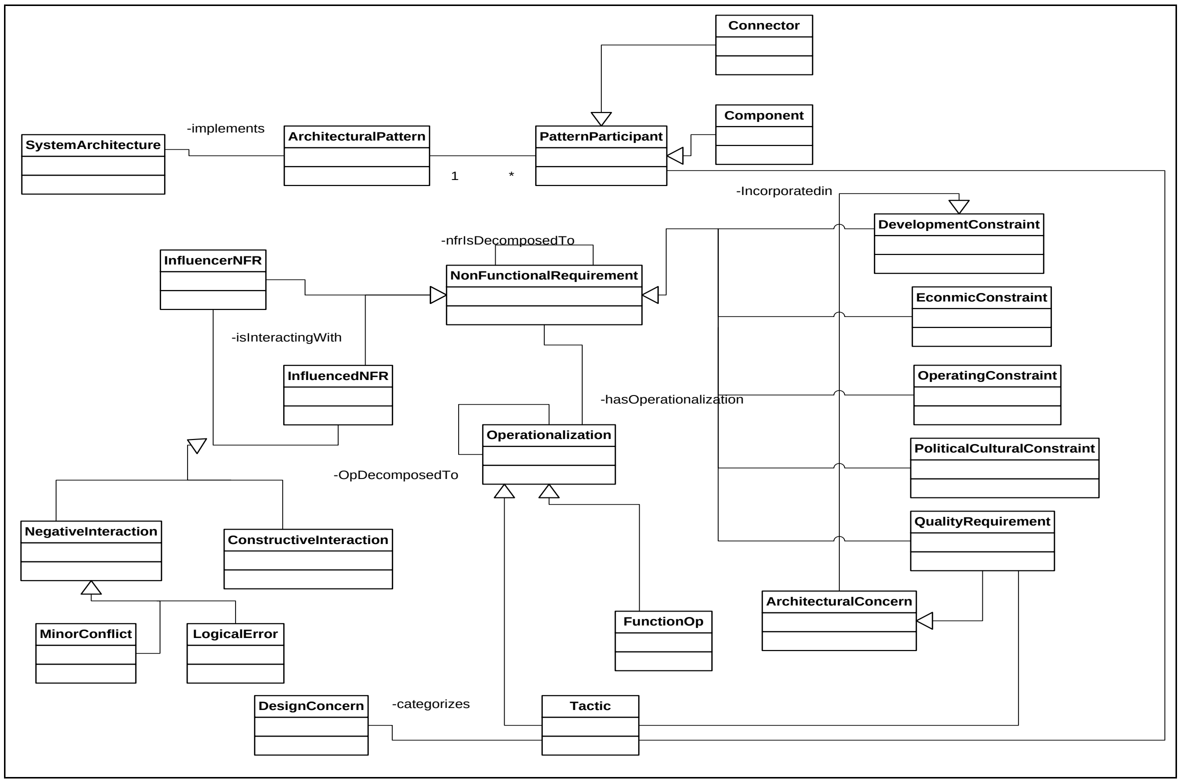

Figure 2. NFRs Intramodel Dependency View.

meaning to a software engineer. From this, architectural concerns are defined as those concerns that significantly influence the architecture [26]. An example of an architectural concern could be the need for coordination between distributed entities within the system.

On other hand; a special type of architectural concern is QualityRequirement [27] (e.g., security guarantees for the system). This implies that quality requirements are in fact development constraints themselves; as the development process should bear in mind the required qualities while taking architectural; design or implementation decisions.

2) Decomposition

This refers to the NfrIsDecomposedTo property that decomposes a high-level NFR into more specific sub-NFRs. In each decomposition, the offspring NFRs can contribute partially or fully towards satisfying the parent. NfrIsDecomposedTo is a transitive property. The decomposition can be carried either across the type dimension or the association point dimension. For example, let us consider the requirement "read an email message with high security". The security requirement constitutes quite a broad topic [6]. To deal effectively with such a requirement, the NFR may need to be broken down into smaller component using the knowledge of the NFR type; discussed in the previous subsection, so that an effective solution can be found. Thus, the requirement states as "read an email with a high security" can be decomposed into "read an email with high integrity", "read an email with high confidentiality", and "read an email with high availability". An example of decomposition across the association point is: "read inbox folder messages with high security", "read system-created folder messages with high security". The decomposition can be "ANDed" (all NFR offspring are required to achieve the parent NFR goal) or "ORed" (it is sufficient that one of the offspring be achieved instead, the choice of offspring being guided by the stakeholders).

3) Operationalization

This refers to the hasOperationalization property that refines the NFR into solutions in the target system that will satisfy the NFR [6]. One type of operationalizations is "FunctionOp" which corresponds to functionalities to be implemented. For example, "Authorization" and "Authentication" are potential instances of FunctionOp class to implement Security quality. Similar to decomposition, operationalization can be ANDed or ORed.

In the inferred taxonomy; the taxonomy after the reasoner impact, the reasoner classifies FunctionOp based on the imposed assertions as a subclass for FunctionalRequirement. This classification is consistent with many arguments in the requirements engineering community on the tight link between the FRs and NFRs [28]. The ontology brings formalism and a concrete understanding to this link.

The second type of operationalizations is "Tactic" which represents design decisions aiming at satisfying some quality requirements. Indeed, when designing software, an architect relies on a set of idiomatic patterns commonly named architectural styles or patterns. A software architectural pattern defines a family of systems in terms of a pattern of structural organization and behavior [29]. More specifically, an architectural pattern determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined [30]. Common architectural patterns include Layers, Pipes and Filters and Model View Controller (MVC). As shown in Figure 2, an instance of SystemArchitecture class is an implementation of one or more instances of ArchitecturalPattern class. While architectural patterns embody high level design decisions, an architectural tactic [31] is a design strategy that addresses a particular quality attribute. Tactics are a special type of operationalization that serves as the meeting point between the quality attributes and the software architecture. Tactics are the building blocks of patterns [31] and implementing a tactic within a pattern may affect the pattern by modifying some of its components, adding some components and connectors, or replicating components and connectors [32]. An instance of class Tactic is linked to an instance of classes Component / Connector through one of the following properties which define the semantics of impact of incorporating the tactic into the pattern (adopted from [27]):

- Implemented in: The tactic is implemented within a component of the pattern. Actions are added within the sequence of the component.
- Replicates: A component is duplicated. The component's sequence of actions is copied intact, most likely to different hardware.
- Add, in the pattern: A new instance of a component is added to the architecture while maintaining the integrity of the architecture pattern. The new component comes with its own behavior while following the constraints of the pattern.
- Add, out of the pattern: A new component is added to the architecture which does not follow the structure of the pattern. The added actions do not follow the pattern.
- Modify: A component's structure changes. This implies changes or additions within the action sequence of the component that are more significant than those found in "Implemented in".
- Delete: A component is removed.

Tactics which have relatively a similar impact can be grouped together into categories which are instances of DesignConcern class. For example, a design concern towards the architectural concern "high performance for the system" is how to "manage resources demands". This design concern is a group of four tactics that aim to improve the performance quality: increase computation efficiency, reduce computational overhead, manage event rate and control frequency of sampling. It's worth to point out that FunctionOp and Tactic are not mutual exclusive classes.

4) Interactivity

An individual NFR may participate in *isInteractingWith* property which links it to another NFR. This refers to the fact that the achievement of one NFR; *InfluencerNfr,* at a certain association point can hinder (through *isNegativelyInteractingWith* property) or help (through *isPositivelyInteractingWith* property) the achievement of other NFR; *InfluencedNfr*, at the same association point, e.g., *security* and *performance* at "*read an email message"* functionality. *isInteractingWith* is not a symmetric property.

The negative interaction is further specialized through the two sub-properties, which help classifying the negative interaction into: *hasLogicalErrorWith* and *hasMinorContradictionWith*.

Logical Error: This is a fundamental conflict which must be resolved immediately. It occurs when the achievement of NFR_1 will prevent the achievement of NFR_2. This is expressed by means of the proposition LogicalError (NFR_1, NFR_2) $\Leftrightarrow$ NFR_1 $\rightarrow$ NOT NFR_2. Logical Error demonstrates a direct contradiction between two requirements. For example, NFR_1 is stated as "Security has to be high at *read email* functionality"; while NFR_2 is stated as "There should be no security constraints at *read email* functionality"!

Minor Contradiction: This is one of the best-known cases of conflict [6]. Associating a win condition with an NFR (say NFR_1) triggers a search of the operationalization that has positive and/or negative effects on NFR_1. For example, the Portability NFR, the win condition of which is "portable to Windows", has positive effects (i) on the portability layers and separation of data generation and (ii) on the presentation, but has negative effects on the use of fast platform-dependent user interface functionalities that would be affected with the layering strategy. The operationalizations, that are found to have negative effects on other NFRs sharing the same association points with their parents NFRs, are used to identify potential conflicts.

IV. EVALUATION

We evaluated our ontology according to three criteria: 1) is it generally acceptable? 2) is it consistent? and 3) is it accurate?. 'Generally accepted' means that the knowledge and practices described are applicable to most projects most of the time, and that there is widespread consensus about their value and usefulness. 'Generally accepted' does not mean that the knowledge and practices described are or should be applied uniformly on all projects [33].

Clearly, the evaluation of the acceptance and the accuracy of the ontology as such ultimately rely upon its application by the research community. For the purpose of this evaluation, we have used our ontology within three different projects. These projects helped refining the initial NFRs ontology. Indeed we have instantiated the ontology against the set of requirements from the settings of the NOKIA Mobile Email Application System and the IEEE Montreal Website. Further, we worked closely with experts from SAP-Montreal to use the NFRs Ontology as a repository for the requirements of some of the projects which are under development. From the experiences and the participants' feedback developed from instantiating the NFRs Ontology against the three real-life projects (the Nokia project, the IEEE Montreal website project and the SAP project), the ontology has proven to be easy to instantiate and links the concepts efficiently. Each individual captured NFR was instantiated from its corresponding concept in the Ontology. We make the note here that we did not meet the case in which an individual NFR was not instantiated from a corresponding concept. Finally the consistency of this ontology has been demonstrated through the usage of a semantic web reasoning system and information repository RACER [21].

V. RELATED WORK

Even though there is no formal definition of the term 'NFR', there has been considerable work on characterizing and classifying NFRs. In a report published by the Rome Air Development Center (RADC) [34], NFRs ("software quality attributes" in their terminology) are classified into consumer-oriented (or software quality factors) and technically-oriented (or software quality criteria). The former class of software attributes refers to software qualities observable by the consumer, such as efficiency, correctness and interoperability. The latter class addresses system-oriented requirements such as anomaly management, completeness and functional scope.

Earlier work by Boehm et al. [35] structured quality characteristics of software within a quality characteristics tree of 25 nodes, noting that merely increasing designer awareness would improve the quality of the final product. A well-known and more recent approach to representing NFRs using a graphical method is the NFRs framework by Chung et al [6]. A cornerstone of the framework is the "softgoal" concept for representing the NFR. A softgoal is a goal that has no clear-cut definition or criteria to determine whether or not it has been satisfied. The operation of the framework can be visualized in terms of the incremental and interactive construction, elaboration, analysis and revision of a softgoal interdependency graph (SIG). High-level softgoals are refined into more specific subgoals or operationalizations. In each refinement, the offspring can contribute fully or partially, and positively or negatively, towards satisfying the parent. However, the particular graphical notations make it difficult to coordinate with mature UML tools and be integrated with existing models of FRs. This integration has been tackled in [24, 36, 37] by extending UML models to integrate NFRs to the functional behavior. Although the integration process must be considered at the meta-level, these approaches only model certain NFRs (e.g., response time, security) in a way that is not necessarily applicable for other requirements.

On a different track, Hauser et al. [38] provide a methodology for reflecting customer attributes in different phases of design. Dobson et al [23] describe an approach to specifying the Quality of Service (QoS) requirements of service-centric systems using an ontology for Quality of Service. The above approaches address only a subset of NFRs; namely quality requirements, and sometimes within a specific context; (e.g., service computing in [24] and automotive industry in [38]). On contrast, our work aims at providing a more generic solution to all types of NFRs with independence from any context.

Al Balushi and Dabhi [39] used an ontology-based approach to build NFR quality models with the objective to gather reusable requirements during NFR specification. We agree with these authors on the usefulness of ontology, however, the research objectives of their research efforts and ours differ, which in turn, leads to essential difference in the research outcomes. While the conceptual model in [1] is geared towards solving requirements reuse problems, our ontology covers a broader spectrum of NFR issues. This is achieved by using multiple views, which explicate requirements phenomena by complementing the strengths of multiple conceptualizations of NFRs.

VI. CONCLUSION AND FUTURE WORK

Although non-functional requirements are receiving more and more attention in the requirement and software engineering communities, little progress has been made in using ontologies for NFRs. This is mainly because NFRs are too abstract and affected by a large number of subjective factors, which makes it difficult for users to describe their own NFRs accurately and precisely. In this paper, we proposed a NFRs ontology that we developed by analyzing and generalizing concepts from the literature. We used a disciplined approach to ontology development, with explicit requirements, ontology design, and implementation. This ontology describes glossaries and taxonomies for NFRs. We used these glossaries for generalization to the common NFRs concepts. To evaluate the ontology, we have used it within the context of three projects. This initial evaluation proved that the ontology is consistent and easy to use.

Clearly, the evaluation of the acceptance and the accuracy of the NFRs ontology, as such, ultimately rely upon its application by the research community. The authors of this are hoping to soon benefit from interaction with a number of interested parties in this topic. In particular, we plan to explore the way in which NFRs ontology could be further leveraged in more complex requirements specification scenarios in real-life settings. In order to ground the concept further, we plan to develop tools to leverage the benefits of ontology for NFRs and evaluate our results against scenarios designed to test the capabilities of the ontology. One potential tool of our interest will aim at facilitating the investigation of studying the impact of incorporating the quality tactics into the software architectural patterns. In addition, we will investigate further to which degree having the NFRs ontology adopted in the requirements engineering activities guarantees the compliance of the final product with the captured NFRs.

REFERENCES

[1] IEEE Std. 830-1998. (1998), "IEEE recommended practice for software requirements specifications", IEEE Transactions on Software Engineering.

[2] T. Jingbai, H. Keqing, W. Chong, and L. Wei, "A Context Awareness Non-functional Requirements Metamodel Based on Domain Ontology", IEEE International Workshop on Semantic Computing and Systems, 2008, Huangshan, China, pp.1-7.

[3] K. K. Breitman, J. C. S. P. Leite, and A. Finkelstein, "The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study", Journal of the Brazilian Computer Society, 1(6), 1999, pp. 13-37.

[4] A. Finkelstein and J. Dowell, "A Comedy of Errors: The London Ambulance Service Case Study", proceedings of the 8th International Workshop Software Specifications and Design, 1996, pp. 2-5.

[5] L. Leveson and C. S. Turner, "An Investigation of the Therac-25 Accidents", IEEE Computer, 26(7), 1993, pp. 18-41.

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

[7] P. M. O. O. Sack, M . Bouneffa, Y. Maweed, and H. Basson, "On Building an Integrated and Generic Platform for Software Quality Evaluation", 2nd IEEE International Conference on Information and Communication technologies: From Theory to Applications. April 24 - 28, 2006, Umayyad Palace, Damascus, Syria.

[8] International Standard ISO/IEC 9126-1. Software engineering – Product quality – Part 1: Quality model. ISO/IEC 9126-1:2001, 200.

[9] O. Mendes and A. Abran, "Software Engineering Ontology: A Development Methodology", Metrics News, 9, 2004, pp. 68-76.

[10] M. A. Sicilia and J. J. Chadrado-Gallego, "Linking Software Engineering concepts to upper ontologies", Proceedings of the First Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering, 2005, Alcalá de Henares, Spain.

[11] C. Wille, A. Abran, J. M. Desharnais, and R. R. Dumke, "The quality concepts and subconcepts in SWEBOK: An ontology challenge", In Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, 2003, Taipei, Taiwan.

[12] K. Haruhiko and S. Motoshi, "Using domain ontology as domain knowledge for requirements elicitation", proceedings of the 14th IEEE International Requirements Engineering Conference, 2006, Minneapolis, USA, pp. 186 – 195.

[13] Z. Jin, "Ontology-based requirements elicitation automatically", Chinese J. Computers, Vol.23, No.5, 2000, pp. 486 – 492.

[14] H. Kaiya and M. Saeki M, "Ontology based requirements analysis: lightweight semantic processing approach", proceedings of the 5th International Conference on Quality Software (QSIC), 2005, Melbourne, Australia, pp. 223 – 230.

[15] IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

[16] M. Kassab, O. Ormandjieva, and M. Daneva, "An Ontology Based Approach to Non-Functional Requirements Conceptualization", Proceedings of the 4th International Conference on Software Engineering Advances, ICSEA 2009, September 20-25, 2009 - Porto, Portugal.

[17] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications", Knowledge Acquisition, 1993, pp. 199 – 220.

[18] C. W. Holsapple and K. D. Joshi, "A Collaborative Approach to Ontology Design", Communication of the ACM, February 2002, Vol 45, No 2, pp. 42 - 47.

[19] F. Baader, I. Horrocks, and U. Sattler, "Description logics as ontology languages for the semantic web", in Lecture Notes in Artificial Intelligence. Springer, 2003, http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/BaHS03.pdf. [retrieved: 11,2012].

[20] W3C, "Web Ontology Language (OWL)", http://www.w3.org/2004/OWL.

[21] Racer: Renamed Abox and Concept Expression Reasoner. http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[22] M. Glinz, "On Non-Functional Requirements", 15th IEEE International Requirements Engineering Conference (RE 2007), 2007, Delhi, India, pp.21-26.

[23] R. Lock, G. Dobson, and I. Sommerville, "Quality of Service Requirement Specification using an Ontology", Conference Proceedings 1st International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'05), Paris, France, 30th August 2005.

[24] J. Araujo, A. Moreira, I. Brito, and A. Rashid, "Aspect-Oriented Requirements With UML", Workshop on Aspect-Oriented Modeling with UML (held with UML 2002).

[25] S. Whitmire, "Object Oriented Design Measurement", John Wiley & Sons, 1997.

[26] N. Bouck´e and T. Holvoet, "Dealing with concerns ask for an architecture-centric approach", In European Interactive Workshop, 2005.

[27] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation", Journal of Systems and Software, vol. 83, Issue 10, pp. 1735-1758, 2010.

[28] B. Paech, A. Dutoit, D. Kerkow, and A. von Knethen, "Functional requirements, non-functional requirements and architecture specification cannot be separated – A position paper", 8th International Workshop on Requirements Engineering: Foundation for Software Quality, 2002, Essen, Germany.

[29] D. Garlan and M. Shaw, "An Introduction to Software Architecture", Technical Report, CMU, Pittsburgh, PA, USA, 1994.

[30] Microsoft Application Architecture Guide: Patterns & Practices, 2nd Edition, http://msdn.microsoft.com/en-us/library/ff650706.aspx.

[31] L. Bass, P. Clements, and R. Kazman, "Software architecture in practice", Addison-Wesley, 2003.

[32] N. B. Harrison, P. Avgeriou, and U. Zdun, "On the Impact of Fault Tolerance Tactics on Architecture Patterns", In proceedings of 2nd International Workshop on Software Engineering for Resilient Systems (SERENE 2010), London, UK, 2010.

[33] PMBOK (2000). Project Management Body of Knowledge Guide 2000. See http://www.cs.bilkent.edu.tr/~cagatay/cs413/PMBOK.pdf. [retrieved: 11, 2012].

[34] T. P. Bowen, G. B. Wigle, and J. T. Tsai, "Specification of Software Quality Attributes", Volume 2, Software Quality Specification Guidebook, 1985.

[35] B. W. Boehm, J. R. Brown, M. Lipow, "Quantitative Evaluation of Software Quality". In proceeding of the 2nd Int. Conference on Software Engineering, San Francisco, CA, Oct. 1976. Long Branch, CA: IEEE Computer Society, 1976, pp. 592-605.

[36] A. Moreira, J. Araujo, and I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", In 14th Int. Conf. on Soft. Eng. and Knowledge Engineering, Ischia, Italy, 2002, pp. 167-174.

[37] D. Park, S. Kang, and J. Lee, "Design Phase Analysis of Software Performance Using Aspect-Oriented Programming", In 5th Aspect-Oriented Modeling Workshop in Conjunction with UML 2004, Lisbon, Portugal, 2004.

[38] J. R. Hauser and D. Clausing, "The House of Quality", Harvard Business Review, May – June 1988, (pp. 63- 73).

[39] T. H. Al Balushi, P. R. Sampaio, D. Dabhi, and P. Loulopoulos, "ElicitO: A Quality Ontology-Guided NFR Elicitation Tool", Proc. Of REFSQ 2007, Requirements Engineering: Foundations for Software Quality, Trondheim, Norway, June 11-12 2007, pp. 306-319.