International Journal of Advanced Computer Science, Vol. 3, No. 11, Pp. 575-588, Nov., **2013.**

Model-Based Trade-off Analysis of Non-Functional Requirements: An Automated UML-Based Approach

Mehrdad Saadatmand [1, ✉], Antonio Cicchetti [2], & Mikael Sjödin [3]

Manuscript
Received:
27, Aug., 2013
Revised:
10, Sep., 2013
Accepted:
18, Sep., 2013
Published:
15, Nov., 2013

Keywords
Non-Functional Requirements, Trade-off analysis, UML, Model-Based Development, Quality attributes, Non-Functional Properties, Extra-Functional Properties.

***Abstract*— One common goal followed by software engineers is to deliver a product which satisfies the requirements of different stakeholders. Software requirements are generally categorized into functional and Non-Functional Requirements (NFRs). While NFRs may not be the main focus in developing some applications, there are systems and domains where the satisfaction of NFRs is even critical and one of the main factors which can determine the success or failure of the delivered product, notably in embedded systems. While the satisfaction of functional requirements can be decomposed and determined locally, NFRs are interconnected and have impacts on each other. For this reason, they cannot be considered in isolation and a careful balance and trade-off among them needs to be established. We provide a generic model-based approach to evaluate the satisfaction of NFRs taking into account their mutual impacts and dependencies. By providing indicators regarding the satisfaction level of NFRs in the system, the approach enables to compare different system design models and also identify parts of the system which can be good candidates for modification in order to achieve better satisfaction levels.**

1. Introduction

In software engineering, there are different types of programming languages and development methods that have been introduced to develop software systems in different domains. There is one common goal that is inherent in all these different development tools and methodologies, and that is to help build a software system which satisfies the set of requirements that are defined for it. While the focus has usually been mainly on functional requirements [1, 2] inadequate attention and improper handling of Non-Functional Requirements (NFRs) has been identified as one of the important factors for failure of many project [3, 4]. In spite of this fact, NFRs are still rarely taken into account so seriously as functional requirements and not considered as first-class entities in software architecture [5]. Part of this is due to the fact that NFRs are usually defined at a high abstraction level and specified in an informal way [6, 5]. Therefore, there need to be appropriate tools and methods to incorporate them at earlier phases of development and in design models along with functional requirements. Integration of NFRs and FRs is especially important considering that having a different set of NFRs for the same FRs can result in different architectural decisions and implementations [6, 7].

While NFRs might receive less attention and degree of importance in certain systems such as desktop applications, however, they can be critical in certain domains such as in real-time and embedded systems. In these systems, there are different set of constraints and limitations on available resources and therefore, a successful design and implementation depends heavily on how it can satisfy the non-functional requirements of the system [8]. Examples of such limitations that get formulated in the form of NFRs can be limited amount of available memory, limited energy resources, and so on. Therefore, it is important to be able to evaluate different design models and alternatives with respect to the satisfaction of NFRs. For example, in one design, to fulfill security requirements, a stronger encryption algorithm might be used than another design alternative. However, using a stronger encryption algorithm may lead to consuming more memory or processing capacity and CPU time, and this way, it impacts memory and performance requirements (if there are any defined). This brings us to the next challenge with respect to NFRs and it is that NFRs are interconnected and have dependencies and for this reason, cannot be considered in isolation. Therefore, designers should be able to carefully identify how satisfying and fulfilling one requirements can impair the satisfaction of other NFRs in the system. Establishing and maintaining such interdependencies during the development process and the lifecycle of the product is also an important point taking into account the evolution of software architecture and introduction of new requirements or modifying existing ones. Moreover, not only NFRs can have impacts on each other, but also an NFR usually crosscuts different parts of a system. For example, achieving security in a system requires design decisions for different parts of a system spanning from user interfaces (e.g., what a user can enter as input), database backends, communication protocols, network topology and so on.

Model-based development (MBD) is a promising approach to cope with the design complexity of systems such as those in real-time embedded domain. It helps to

Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin; Mälardalen Real-Time Research Centre (MRTC); Mälardalen University; Västerås, Sweden; ✉{mehrdad.sadatmand, Antonio.cicchetti, mikael.sjodin}@ mdh.se.

raise the abstraction level (and hiding unnecessary details and complexities in each viewpoint), perform analysis at earlier phases of the development and also enables automatic generation of code from the models [9]. By providing views of the system at a high abstraction level, MBD concepts can also be used to model NFRs, which as stated are usually defined at a high abstraction level, and incorporate them with other parts of the system. Analysis of NFRs can then be performed on the model and also the model of NFRs can be maintained as a development artifact.

In this paper we introduce a UML profile [10, 11] for modeling NFRs in a generic way and regardless of their type (i.e., performance, availability, and so on), to enable performing trade-off analysis on them. By including important information about each requirement in the model, such as its priority and also its relationships to other requirements and functional parts of the system, the dependencies and impacts of NFRs are analyzed to provide system designers with information about how good a system design is in terms of the satisfaction of its NFRs. It also helps to identify parts of the system in which violations and deviations have occurred that deserve more attention. Based on this information, system designers can also compare different design models and alternatives. Another approach for modeling NFRs could be to define a Domain-Specific Language (DSL) from scratch (i.e., non-UML based approaches), however, using UML and its profiling mechanism to extend it and define new modeling semantics has some advantages. Introducing a DSL requires extra efforts on training the developers, while most developers may already be familiar or even using UML. For this reason, it can also serve as a unifying factor between different development teams (e.g., to communicate design decisions). Moreover, there is a big variety of different UML tools which are already available and can be used off-the-shelf. Also, integrating NFRs with functional parts of the systems will be more straightforward, such as when there already exist UML models of the system and the model of NFRs based on our introduced profile can be constructed and integrated with them (e.g., legacy systems). A comparison of advantages and disadvantages of using DSLs and UML profiles for defining new modeling semantics are discussed in detail in [10, 12].

Using the suggested profile for modeling NFRs, not only NFRs can be modeled and integrated with already existing functional models of the system, but it is also intended to be used for constructing the NFR model at the beginning of the development process and to perform analysis of their trade-offs, especially when enough information about their impacts and dependencies are available. The model may then gradually grow, be integrated with functional parts as they get designed, and automatic analysis of NFRs can be done iteratively when any changes that can affect NFRs are made.

The remainder of the paper is structured as follows. In Section 2, NFRs and their characteristics are introduced and the challenges related to NFRs during the development process are identified and discussed. In Section 3, we formulate and summarize the characteristics that different solutions for managing the trade-offs of NFRs should be able to provide to cope with the identified challenges. Section 4 describes in the detail the suggested UML profile and its modeling semantics including the rules and formulas that are defined for performing trade-off analysis on the models of NFRs. An application of the profile and how analysis is performed on NFRs is provided in Section 5 using a selected part of the NFR model of a mobile phone. Discussion of different aspects of the proposed approach is offered in Section 6. In Section 7, related works are investigated and finally in Section 8, a summary of the work is provided and conclusions are drawn.

2. Non-Functional Requirements

A. Definitions

A requirement is basically an expression of a need [13] and in developing software systems, there can be different stakeholders with their own specific requirements [14]. Some requirements, such as those related to the user interface, may originate from the customer or end-user side, while some other requirements may be due to the selection of a particular development process (e.g., agile or model-based development). Also, there are different standards and regulations that may need to be followed in the development of a software system which bring along additional sets of requirements. Examples of such standards could be different safety standards that a safety-critical system should conform to, for instance, in avionics, automotive, and medical systems.

In systems engineering, requirements are usually categorized as *functional* and *non-functional* [13]. Simply stated, functional requirements state what a system should do and are sometimes identified as capabilities of a software product [14], whereas non-functional requirements define how the system should perform or as mentioned in [15] a non-functional requirement is "an attribute or a constraint on a system". A list of different definitions for non-functional requirements are collected in [16]. An example for functional requirements could be that a system should be able to read input from a text file. A non-functional requirement could be that the process of reading the input file should not take more than 10 milliseconds; this requirement is basically an expression of a performance need in the system.

The IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E) [17, 18] provide the following definitions for requirement, and functional and non-functional requirements (quoted):

- Requirement:
    1. a condition or capability needed by a user to solve a problem or achieve an objective.
    2. a condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard,

specification, or other formally imposed documents.
3. a documented representation of a condition or capability as in (1) or (2).
4. a condition or capability that must be met or possessed by a system, product, service, result, or component to satisfy a contract, standard, specification, or other formally imposed document.

- Functional Requirement:
  1. a statement that identifies what a product or process must accomplish to produce required behavior and/or results.
  2. a requirement that specifies a function that a system or system component must be able to perform.

- Non-Functional Requirement: a software requirement that describes not what the software will do but how the software will do it (i.e., design constraints). Examples: software performance requirements, software external interface requirements, software design constraints, and software quality attributes. Non-functional requirements are sometimes difficult to test, so they are usually evaluated subjectively.

Moreover, a requirement can be refined (into smaller, more detailed and fine-grained ones) and this way a hierarchy of requirements can be created. The term *derived requirement* is also offered by the IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E), which is defined as:

- Derived Requirement:
  1. a lower-level requirement that is determined to be necessary for a top-level requirement to be met.
  2. a requirement that is not explicitly stated in customer requirements, but is inferred from contextual requirements (such as applicable standards, laws, policies, common practices, and management decisions) or from requirements needed to specify a product or service component.

In this context, the term extra-functional is also used at times as an equivalent of non-functional to change the focus and take away and replace the negative aspect that is inherent in 'non'. On the other hand, there is the concept of non-functional/extra-functional property (NFP/EFP), which is often confused with NFRs. As a type of requirements, NFRs are also expression of a need which are generally stated in an informal way, while a property is a statement that can be asserted formally, and therefore, it can be analyzed and proven. An example of extra-functional properties could be the worst-case execution time of a component in a system which may be calculated statically or measured. Therefore, saying that "the worst-case execution of component A is 5ms" or that "the execution time of component A never exceed 10ms" are actually expression of properties. On the other hand, "the execution time of component A should never exceed 10ms" is a non-functional requirement and an expression of a need. The key point here is that a property per se does not tell us much about its validity, and it is only when it is considered along with its related requirement(s) that we can determine whether it is acceptable and good for a specific design or not. In other words, if we know that the worst-case execution time of a component is 5ms, we cannot determine whether it can be considered a good value or not, unless we check it against the requirements. While for one system this value of 5ms could be acceptable, for other systems this may be considered as problematic and lead to the violation of requirements. Considering such a relationship between an NFR and an extra-functional property, to satisfy an NFR, its related extra-functional properties should have valid values. For example, to satisfy performance and schedulability requirements in a real-time system, execution and response time values (among others) should remain within a valid range. Understanding the differences between these two terms is important in some works (such as this paper), while in other contexts, their differences can be ignored and using these two terms as equivalents can be safe. In [19], NFP is used instead of NFR when talking about the final product implying that the requirement has been concretized and become an actual property of it.

B. Characteristics and Challenges

Addressing NFRs in the development of a software product is a challenging task. Aside from the fact that often times NFRs are expressed in a natural language and informally, they have some characteristics that makes their consideration in the development process complicated. In contrast to FRs which are typically realized locally and implemented one by one and step by step in an incremental manner while the software product is being built, NFRs do not follow such a pattern. In this respect, NFRs can be considered as specification of global constraints on the software product, such as security, performance, availability and so on [5] which can crosscut different parts of a system. Also in satisfying NFRs, the dependencies among them should not be neglected, as satisfying one NFR can affect and impair the satisfaction of other NFRs in the system. Therefore, performing trade-off analysis to establish balance among NFRs and identify such mutual impacts is necessary.

There are also other issues that contribute to the complexity of managing NFRs in the development process. For example, organizational structures of companies and the way they are divided into different development departments and sub-departments usually fit functional requirements; as these requirements can be (more easily)

implemented in separation from each other and then integrated to satisfy a parent requirement (considering a hierarchy of requirements consisting of refinements of each) [20, 21]. On the other hand, a non-functional requirement such as security, availability, or user-friendliness crosscuts different parts of the system and requires a more holistic view and a top-down approach [21]. Another problem which is mostly observed in large organizations is that different teams may have different interpretations of an NFR, or vice versa, refer to one NFR using different terms [20]. Therefore, a coherent way of representing and defining NFRs, and also establishing and maintaining traceability links among them can be helpful to mitigate such problems. Issues related to traceability between NFRs can also occur easily during the development process [22]. For example, code tweaks that one development team may do to improve performance, which may affect security or memory consumption, can become hidden and lost to other teams.

Considering that NFRs are usually specified in an informal and abstract way [5, 9], providing a more formal approach using model-based development which enables to raise the abstraction level can help with the treatment of NFRs during the development process. Dealing explicitly with NFRs and incorporating them in different phases of development becomes more important especially considering the increasing number of systems in which NFRs are critical such as real-time embedded systems. Moreover, an explicit treatment of NFRs facilitates the predictability of the system in terms of the quality properties of the final product in a more reliable and reasonable way [19].

Sometimes the approaches for the explicit treatment of NFRs are categorized into two groups: product-oriented and process-oriented [23]. The former approaches try to formalize NFRs in the final product in order to perform evaluation on the degree to which requirements are met. In the latter approaches, NFRs are considered along with functional requirements to justify design decisions and guide and rationalize the development process and construction of the software in terms of its NFRs [23, 19].

3. Addressing the Challenges of NFRs

Considering the nature of NFRs and to cope with the challenges that have been discussed so far in managing and treatment of them in the development process, we formulate here the key features that are required in order to model NFRs and enable performing trade-off analysis among them to evaluate a system design with respect to the satisfaction of its NFRs.

Traceability of design decisions related to an NFR: An NFR can crosscut different parts of a system and there needs to be a mechanism to identify the parts that contribute to its satisfaction. Establishing such a relationship is especially important after performing trade-off analysis in order to identify which parts of the system should be replaced or modified in order to improve the satisfaction of an NFR. On the other hand, in maintaining a system, it is important to find out which requirement(s) a specific part of a system is related to and as a result of which requirement(s) that part has been implemented. Such information can easily become lost in complex systems and also as the system ages.

Traceability between an NFR and its refinements: as mentioned before, during the whole development process, high level NFRs get refined into more fine-grained ones which leads to the formation of a hierarchy and tree-structure of NFRs and parent-child relationships among them. Therefore, in order to evaluate the satisfaction of one NFR in the system, it is necessary to keep track of its refinements and the children requirements originated from it at lower levels of requirements hierarchy. The evaluation of an NFR, is thus, performed recursively by evaluating to what degree its refinements have been satisfied. As an example of such refinements, we can name security as an NFR which can then be refined into lower level and more concrete requirements such as encryption of data and access control mechanisms.

Impact of an NFR on other NFRs: Due to the impacts that NFRs have on each other and the interdependencies among them, an NFR cannot be considered in isolation in a system in order to satisfy and achieve it. System designers should be able to identify the impacts that a system feature and design decision that is made to satisfy one NFR can have on other NFRs. Examples of such impacts can be more tangible in embedded systems. For instance, performing heavy computations by an encryption component in an embedded system can lead to consuming more battery. Therefore, it is important to be able to identify and include such impacts and side effects as part of the system design models.

Priority of an NFR: In a system, different NFRs can have different levels of importance. It is necessary to know the importance of each NFR to be able to compare them and resolve conflicts among them (reduce the impact of one NFR in favor of another) to improve the overall satisfaction of NFRs. Considering priorities for NFRs is also important to capture the preferences of customers. Similarly, priorities can also be considered for different features implemented to satisfy an NFR.

Satisfaction level of an NFR: To enable comparison of a system design against the specifications of the system and customer requirements and also to compare different design alternatives, it is needed to evaluate, specify and

represent the satisfaction degree/level of an NFR in the system. The end goal is that system designers should be able to get an idea to what extent each NFR is satisfied and how good a system design is in terms of the satisfaction of its NFRs. After analyzing the dependencies and impacts of NFRs and determining their satisfaction levels, as the next step, it can be judged whether the satisfaction level of an NFR is acceptable or not. This phase can probably be done by checking and consulting with the stakeholders, if needed.

Coherent terms for NFRs: It was discussed that especially in large organizations, it can happen that different departments and development teams may have their own interpretations for each NFR or use different terms to refer to an NFR. By providing a coherent and consistent representation and notation for NFRs and also establishing traceability links for them (to other NFRs as well as to design elements implementing each), it becomes possible to mitigate such inconsistency problems. This problem can be very subtle and easily remain unnoticed [20].

Coherent measurements of NFRs: To enable the comparison of different NFRs and performing trade-off analysis among them, specification of the satisfaction level and impact values of NFRs should follow a coherent representation. This means that the criteria or metrics that are used should be such that to allow pair-wise comparison of NFRs (e.g. using the same types, scales and units, or a convertible format).

4. Suggested Approach

This section is devoted to the illustration of the proposed UML profile enabling the modeling of NFRs and hence their trade-off analysis. Therefore, in the following we first introduce some basic concepts about UML profiles that underpin the technicalities of our proposal.

A. UML Profiles

As mentioned before in this article, thanks to MBD the early evaluation of quality attributes can dramatically save development time and verification and validation costs. The underlying assumption is that the adopted modeling means are capable of carrying by enough details to perform reliable evaluations.

Historically there have been two different ways of addressing language expressiveness limitations, either UML profiling or designing a new DSL from scratch. The former exploits a possibility given by the UML to extend itself, while the latter prescribes building a new modeling language specifically tailored to the domain taken into account. Both approaches have their own advantages and drawbacks [10, 12], the discussion of which goes beyond the scope of this article. However, it is worth noting that, especially in industrial settings, UML profiles are typically preferred due to multiple (practical) reasons: UML is a *de facto* standard for modeling industrial software systems, therefore it is expectable the existence of a 'legacy' including models, tools, skilled personnel, and so forth; UML profiles, as will be discussed below, are still UML models, thus compatible with other models, and even more important, with existing UML tool formats. We opted for a UML profile as the means for supporting the modeling of NFRs details to enable their trade-off analysis. Nonetheless, there are no limitations from the expressiveness perspective preventing the realization of the same kind of modeling support by adopting the DSL solution.

UML has been conceived from the beginning as a general purpose language, therefore it does not contain any domain-specific concept. On the contrary, it allows to model any kind of reality abstraction thanks to its expressiveness. Preservation of generality comes at the cost of lack of precise semantics and ambiguities that can be fixed by exploiting UML profiles. It is worth mentioning that the UML language can be refined by adding, removing, and changing the available concepts, thus creating a new DSL [10]. However, models created by means of such a new language would be not compatible with other UML models and tools. Consequently, UML has been equipped with modeling concepts able to specialize the language itself, i.e. profiles [11].

A *Profile* is a specialization of an existing UML modeling concept; for instance, profiles can be created not only for classes and relationships, but also for states in Activity Diagrams, actors in Use Cases, messages in Sequence Diagrams, and so forth. *Interface* is a famous example of profile for *Class*. When exploited, the profile allows users to recognize that what they have in their hands is not a regular UML Class but an Interface, and act appropriately (that is, give a precise semantic to the kind of object taken into account). Profiles can be also enriched by adding new attributes and properties, called *Tagged Values* (simply referred to as properties in this work). In this way, information can be provided as specifically pertaining to the introduced profile. In the next section, we show how this powerful concept can be used to store NFRs information in order to enable trade-off analysis at the design level of abstraction.

B. NFR Profile

Based on the challenges identified in Section 3, we have created a UML profile to define NFRs as model elements and include necessary information (in the form of properties of model elements and different relationships among them) to enable performing trade-off analysis and evaluating the design with the respect to the satisfaction of NFRs. The structure of the defined profile is depicted in Fig. 1. The profile consists of several key stereotypes and properties that are described as follows:

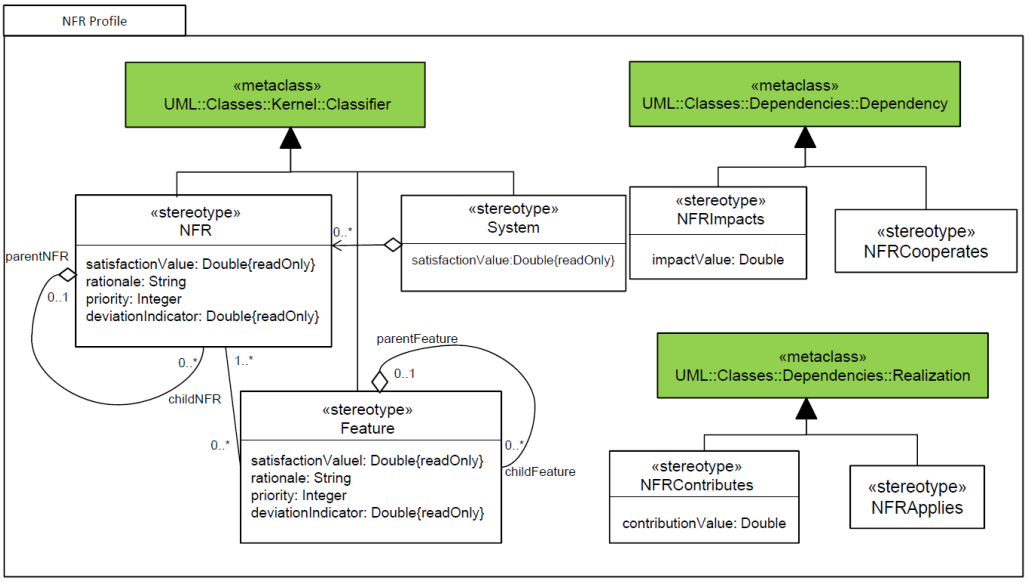

Fig. 1. NFR Profile

System: In the hierarchy of NFRs, the root node will represent the system itself which can have several different NFRs represented in the model at the lower levels of the hierarchy as children model elements. The System stereotype is used to annotate this root model element as the system. The system is also considered as the context of the analysis.

SatisfactionValue: This property is used to represent the satisfaction degree of the model element it belongs to and to what extent it has been fulfilled. As can be seen in Fig. 1, several stereotypes have this property. In case of the System stereotype, the value of this property shows the total calculated satisfaction value for the system (described later). This value is calculated and set by the analysis engine and the users cannot set it.

NFR: NFRs in the system are stereotyped and annotated with this defined stereotype. Since NFRs can have other NFRs as refinements and thus as children nodes, an association relationship to itself (reflexive aggregation) has been defined for it.

Feature: A feature in the system that is defined to satisfy an NFR is identified by using this stereotype. It is basically the equivalent of *Operationalization* concept in NFR framework and Softgoal Interdependency Graph (SIG) [24] or *tactics* as used in [1] (described later in the work).

NFRContributes: This stereotype is used to indicate that an NFR or Feature contributes directly to the satisfaction of another one. It has a property called contributionValue that specifies the degree of this contribution.

NFRImpacts: this is similar to NFRContributes stereotype but is used to include the impact of a model element on other NFRs in the system in a quantitative manner. In other words, this stereotype is defined to capture the side effects of features and NFRs. ImpactValue property of this stereotype shows the degree of the impact. A positive value for the ImpactValue implies a positive side effect, and a negative one implies a negative side effect accordingly.

NFRCooperates: When there are more than one element that are defined to work together in satisfying an NFR, this stereotype is used to annotate and show such a cooperation relationship between them. This concept is similar to the AND relation in the NFR framework and SIG (another reason to provide this stereotype to explicitly specify such cooperation relationships is to help with the extensibility of the suggested approach in future to include different design alternatives in the form of OR relationships in the same design model, when needed).

NFRApplies: This stereotype is defined to enable the possibility to relate the NFR model to functional model elements (e.g. an NFR that applies to a component). For instance, if there is already a UML model of the system available (e.g., a class diagram), with this stereotype it can be specified to which part of that model an NFR or Feature applies and is related to.

Rationale: The rationale behind having an NFR or Feature and any other description about it can be captured and specified in this property. Both NFR and Feature stereotypes have this property.

Priority: This property which exists in both NFR and Feature stereotypes captures the preferences of customers (and also developers priorities when relevant and applicable) and their priorities in terms of the relative importance of NFRs and Features.

DeviationIndicator: By taking into account the priority and the satisfaction value of an NFR or Feature, a value for this property is calculated (as will be described soon) and provided which indicates to the designer the importance and

magnitude of how much the satisfaction of an NFR or Feature has deviated or been violated. The deviation indicator value basically shows and helps to identify which parts of the system have deviated more from the specification (i.e., from being fully satisfied) and may need to be modified to achieve a better satisfaction level. This value is also calculated and set by the analysis engine and the users cannot set it. While the satisfaction value does not reflect user preferences and priorities, the deviation indicator value identifies to the designers which parts need to be considered first with respect to the preferences and priorities of the customers. This is especially helpful and beneficial for identifying such parts in complex systems.

To use the profile and perform calculations, there are several rules that are defined on model elements and their relationships and how to set and calculate values for different properties:

- The priority for an element can be set to one of the following values: 1 (very low), 2 (low), 3 (medium), 4 (high), 5 (very high).
- The satisfaction value for each leaf node is always considered to be 1.
- The contribution value of the NFRContributes link connecting a child node to its parent can be set as a positive value between 0 and 1, but the sum of the contribution values of the links connecting children nodes (refinement/lower level elements) to their parent should always be less or equal to 1.
- The contribution of a child node to its parent is calculated by multiplying the satisfactionValue of the child node by the contributionValue of the NFRContributes link that connects it to the parent.
- For NFRImpacts links, the allowed range of values is between -1 and 1. A negative value on the NFRImpacts relationship shows the negative impact of the source element on the target.
- The total impact value of other nodes on a node (denoted as *I*) is calculated as follows: if the sum of all impact values is positive and not greater than 1, then the total impact value will be this sum, however, if the sum is greater than 1, then the total impact value on the node will be 1. On the other hand, if the sum of all impact values is negative and not less than -1, then the total impact value will be this sum, however, if the sum is less than -1 (e.g., -1.5 or -2), then the total impact value on the node will be set as -1. Note that the value of *I* in this calculation will always be between -1 and 1. This is summarized by the following formula, considering that i_j is the impact value of another node on the node for which we want to calculate the total impact value:

$$I = \begin{cases} Min(\sum i_j, 1) & if \sum i_j >= 0 \\ Max(\sum i_j, -1) & if \sum i_j < 0 \end{cases} \quad (1)$$

- To calculate the satisfactionValue of a node, first the total contributions from all of its children nodes are calculated, and then the total impact value is also taken into account. If s_k is the satisfaction value for each child node of a node, l_k is the value on the link that connects the child node k to its parent node (NFRContributes relationship), and I is the total impact value, the satisfaction value of the parent node is calculated as:

$$S = \begin{cases} Min((\sum s_k * l_k) + I, 1) & if (\sum s_k * l_k) + I >= 0 \\ 0 & if (\sum s_k * l_k) + I < 0 \end{cases} \quad (2)$$

Considering the above rules and formulas, the satisfaction value of a node will be in the range of 0 and 1. To perform these calculations, nodes are navigated and traversed starting from leaf nodes (considering that the satisfaction of leaf nodes is 1) and values are calculated using the above formulas upwards toward the top element which is the system.

- The DeviationIndicator is calculated after the calculation of satisfaction value using the following formula:

$$DeviationIndicator = Priority - Priority * SatisfactionValue \quad (3)$$

- Based on this calculation and considering that the SatisfactionValue is always between 0 and 1 and priority is an integer value between 1 and 5, the value of DeviationIndicator will be in the range of [0,5]. The perfect situation is when the DeviationIndicator value is 0, and the more this value increases the more is the deviation from the desired design, and thus, it indicates a bigger and more severe problem.

C. Implementation

The profile and its concepts that were described are implemented using MDT Papyrus [25] in Eclipse [26]. To navigate and transform a model that is annotated with our suggested UML profile, a model-to-model (M2M) transformation is also developed using QVT Operational language (QVT-O) [27]. The transformation incorporates all the rules for performing calculations and reads as input a UML model annotated with our profile, traverses the nodes

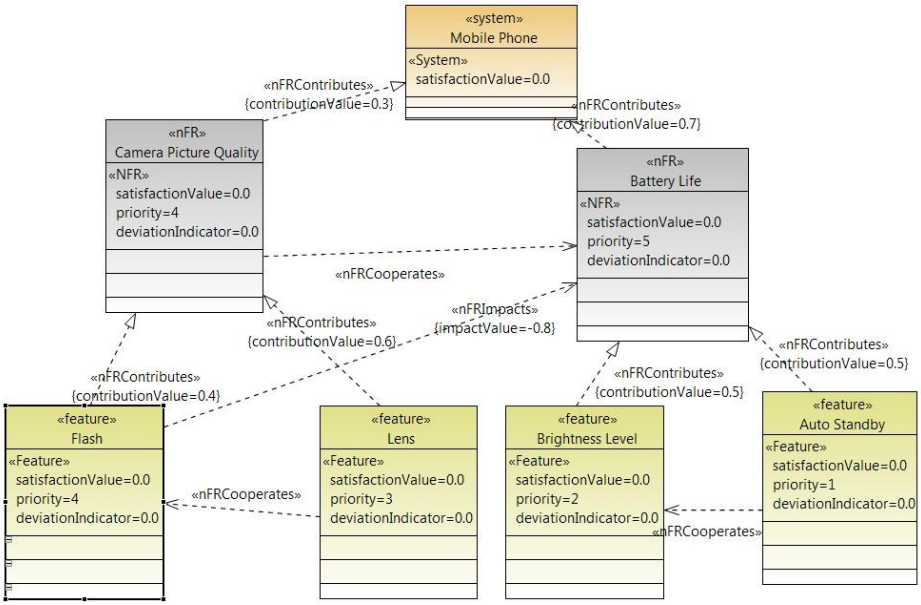

Fig. 2. NFRs for the mobile phone system (before analysis)

and calculates satisfaction and deviation values and writes the results back in the same model. This means that we use an *in-place transformation* (i.e. input and output models are the same) to perform the analysis on the model. A recursive algorithm is executed as part of the transformation which starts from the System node. To calculate the total satisfaction value of the system, it first retrieves the children NFRs of the system node and recursively performs calculations on each of them based on the defined formulas and rules; meaning that all the children of that node are again retrieved and this continues until it reaches a leaf node whose satisfaction value will be considered 1. In other words, for each node, first all the links that are stereotyped with NFRContributes or NFRImpacts are retrieved. A node which does not have such a link is then considered a leaf node, while for other nodes, the source node of the link is retrieved (which will be another node); hence the recursion.

5. Usage Example

In this section we show the applicability of the approach and how it is used for modeling NFRs and performing analysis on them to evaluate the *satisfiability* (by this term we mean the ability to satisfy the NFRs) of a model and also compare it with other design alternatives. Fig. 2 shows NFRs that are defined for part of a mobile phone system using our profile in Papyrus. One NFR is defined for the quality of the pictures that are taken by the mobile phone. This NFR which can for example state that the quality of the picture should not be below a certain level is represented in the model simply as `Camera Picture Quality`. Similarly, another NFR is defined to represent the requirement on efficient use of battery and energy consumption in the mobile phone, denoted as `Battery Life` NFR in the model. To satisfy the `Camera Picture Quality` NFR, the possibility to use flash for taking pictures, and also a specific type of lens have been considered (modeled as `Flash` and `Lens` features). To satisfy and achieve the requirement related to the battery life of the mobile phone, automatic adjustment of brightness level and also automatic standby mode (e.g., when the phone is in idle state) have been designed.

NFRContributes stereotype is used to annotate the relationship between each feature and the NFR to which it contributes. Moreover, the dependencies and impacts of NFRs and features on each are modeled using the NFRImpacts stereotype, which as mentioned before can have positive or negative values. Since the use of the flash has a negative impact on the battery level and consumes energy, the value of the NFRImpacts relationship between the `Flash` feature and `Battery Life` NFR, which shows the magnitude of this impact is specified as a negative number. Importance of different NFRs and features for the customer and his/her preferences are captured by the priority property. The initial values of `satisfactionValue` and `deviationIndicator` properties are zero indicating that no calculation has been done on the model yet.

To analyze the model and perform calculations based on the formulas defined for the profile (which are implemented as part of the transformation code), the model is fed as input to the transformation. The calculations are done using the recursive algorithm that was described before. In case of the mobile phone example here, the `Flash` and `Lens` features will be identified as leaf nodes and thus their satisfaction values are set to 1. The satisfaction value of `Camera Picture Quality` is calculated as the satisfaction value of `Flash` multiplied by the contribution value of the NFRContributes links that connects it to the `Camera`

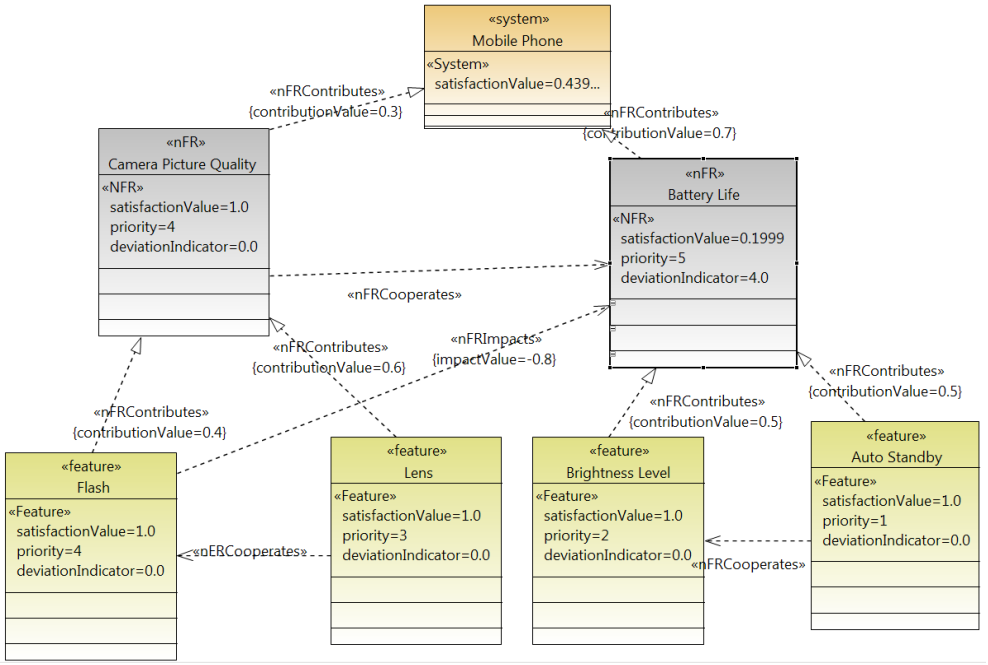

Fig. 4. Analyzed model of the system

`Picture Quality` plus the same multiplication done on the `Lens` and its NFRContributes link: *1*0.4+1*0.6=1*.

The same calculations are done to obtain the satisfaction value for `Battery Life`, however, in this case there is an impact from the use of the `Flash` feature. Therefore its satisfaction value is calculated as: *1*0.5+1*0.5-0.8=0.20*. Fig. 4 shows the analyzed model of the system. The discrepancy that is observed in the calculated satisfaction value for *Battery Life*, that is 0.1999… instead of being 0.20, is due to the OCL implementation of real numbers that are used in QVT.

The total satisfaction value which is calculated for the `System` node is therefore: *1*0.3+0.2*0.7=0.44*. Having the satisfaction values of NFRs and features in the model, the deviation indicator values can now be calculated using Formula 2. The deviation indicator value for the leaf nodes will always result in 0 as their satisfaction values are set to 1. For the `Camera Picture Quality` whose satisfaction value is also 1 the deviation indicator value will be *4-4*1.0=0* as well. However for `Battery Life`, this value will be *5-5*0.2=4*. This high deviation indicator value (compared to other parts) in the model shows the designers that this part of the model requires a more careful attention. Such parts could be good candidates for modification and refactoring in order to improve the satisfiability of the system. Considering the deviation indicator value of the

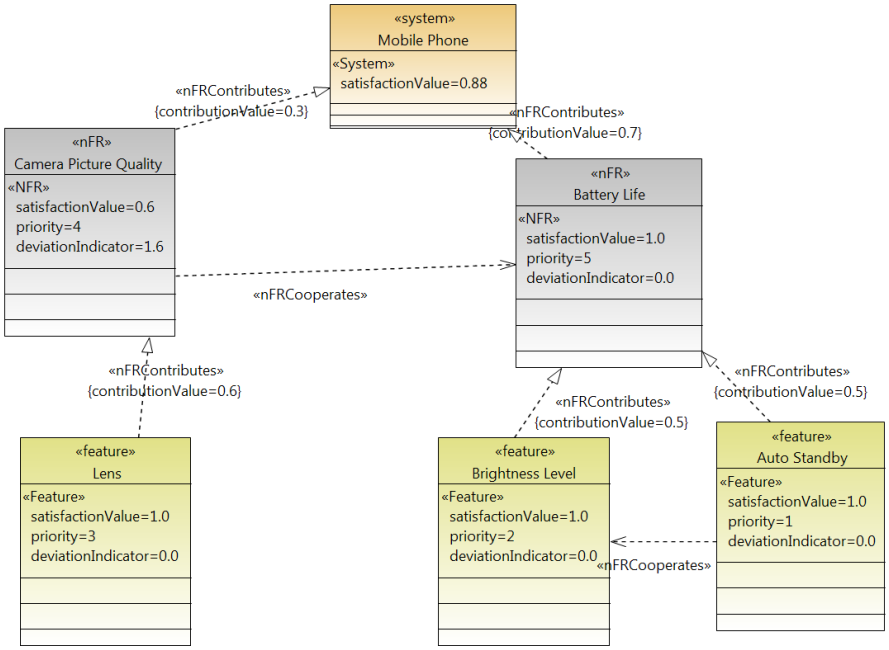

Fig. 3. Analyzed model of the system without the Flash feature

Battery Life, and by investigating the elements that have impacts on it (here only the Flash feature), it can imply that the type of the flash that is selected to be used in this system and model of mobile phone is not good enough in terms of energy consumption and a more energy efficient flash can be used to improve the satisfiability of the system. In this rather simple example, we could have also guessed the issue with the type of flash that is used, based on the magnitude of the impact that it has on the Battery Life in the system; especially that it is the only impact on Battery Life (there could, for example, exist other NFRs and features with positive or negative impacts on it as well). However, in more complex systems with lots of dependencies and mutual impacts and taking into account the priorities of the customers, identifying the parts that have quite major (negative) effects on the satisfiability of the system and thus are of utmost importance to be re-considered could be a real staggering challenge.

Fig. 3 shows the model of the system but without the Flash feature, which could represent a different model and family of mobile phones. By performing analysis on this model, the total satisfaction value of 0.88 is calculated for this design of the mobile phone; versus 0.44 in the model which included the flash. On the other hand, removing the flash, as can be seen from the analyzed model, has led to some deviaiton (1.6) in the Camera Picture Quality NFR.

6. Discussion

As was demonstrated in the previous section, our suggested approach enables designers to compare different design alternatives with respect to the satisfaction of NFRs by taking into account interdependencies and impacts of NFRs as well as the features that are designed to satisfy and fulfill each. This can help the designers in making decisions when building a system. Moreover, the approach provides for several other interesting features which we discuss here. Considering that we can now evaluate the satisfiability of a system design and compare different design alternatives, it becomes also possible to use the suggested approach in optimization of design models with respect to their NFRs. For example, in the mobile phone system, if there is a kind of repository of NFRs and features to choose from, it becomes possible to perform a series of analysis in order to find a set of NFRs and features which lead to the highest possible satisfaction value for the Battery Life NFR, for instance (or even the whole system). However, this may not be as simple as it sounds due to the famous state-space explosion problem [28] that can happen in bigger and more complex systems.

Another use of the suggested approach could be to support runtime adaptation and building re-configurable systems. For instance, in case of power consumption in the mobile phone system example, if at runtime it is detected that the battery level has fallen beyond a certain level, an analysis can be performed using the introduced approach to find alternatives and identify a set of features that incur minimum impact on the battery consumption and then replace active components in the system accordingly to make the system go into a power-saving mode. To reach such an adaptive behavior, the analysis part may or may not be done at runtime. In other words, different design alternatives may have been considered and analyzed offline, and then based on desired Quality-of-Service (QoS) levels at runtime, a different architecture may be adopted to re-configure the system (similar to design diversity techniques [29]).

To enable performing a quantitative type of analysis which in turn gives designers the possibility to more carefully evaluate a model as well as different parts of it and also compare it with other alternatives, it was assumed that the designers can specify the necessary values (in this case, contribution and impact values). There are some methods that help with providing such quantitative information (as will be discussed in the related work section), however, as also mentioned in [18, 1], deciding on these values is usually a subjective task, whose precisions can be improved and increased through the use of the different methods. On the other hand, our suggested approach is deemed more suitable in Component-Based Design (CBD) of systems [30], where a system is built by composing and as an assembly of already existing components, and thus, more information and knowledge about the characteristics and behaviors of the different constituting features of the system are available. Such information could be memory usage, execution time, energy consumption and similar properties which help designers to specify more accurate quantified values in the NFR model. For example, if there is an NFR which specifies that the actual throughput should not be lower than a certain level, however, a protocol is used to satisfy security requirements which is known to double the amount of transmitted packets due to the transmission of security related information, then the impact of this feature on the bandwidth NFR can be specified as -0.5 indicating that it consumes half of the bandwidth to pass the additional information. Also, in this work we assumed that the satisfaction values of leaf nodes are always 1, meaning that they are/will be fully implemented. If, for any specific reasons, the system needs to be analyzed using not-fully implemented features, then this assumption and rule can be relaxed to also enable specifying values between 0 and 1 for leaf nodes.

7. Related Work

One of the fundamental works in the field of non-functional requirements is the NFR Framework which is proposed in [24]. It is a process- and goal-oriented approach which makes use of Softgoal Interdependency Graphs (SIG) to represent NFRs. In this approach NFRs are refined into other fine-grained NFRs and also entities that function to satisfy NFRs which are termed as *Operationalization*. The dependencies and contributions of

NFRs are specified using *make*, *hurt*, help, *break* and *undetermined* relationship types. Besides NFR softgoals, and operationalizating softgoals, NFR framework also introduces *claim* softgoals which convey the rationale and argument for or against a design decision. In addition, it provides notations to mark critical NFRs in the graph as a way to specify priorities on NFRs, and also an evaluation procedure to determine the satisfaction and conflicts of NFRs. NFR Framework is basically a qualitative approach for evaluation of NFRs and their impacts and dependencies, which although is quite useful for capturing NFRs and their relationships, but evaluating the satisfaction of NFRs is not easy [1] and hard to automate. Moreover, the criticality concept in NFR framework may be more suitable for developers and does not convey enough information for prioritization of NFRs particularly from the customer's perspective and also for performing trade-off analysis. In [1], QSIG is introduced which is basically a quantified version of SIG. It enables to perform quantitative evaluation of impacts and trade-offs among NFRs. Our work is inspired by the QSIG approach in the sense that the structure of the UML model that is built is similar to that of QSIG, and as in QSIG, we also defined a set of rules for calculations of different values, although our rules are different to be more suitable for complex systems where, for example, an NFR may be impacted by several different NFRs. We also introduced the concept of deviation indicator which is especially useful in such situations in complex systems to identify problematic parts of them. Also in QSIG, there is no explicit concept of priority for capturing customers' preferences and the impact of one NFR on another is assumed to also convey priorities. This is also another fundamental difference as we believe the concept of impact and priority should be separate, considering that the impact of an NFR on another one should be evaluated per se, while the customer priority for that the latter NFR can show the designers the meaning and importance of such impact especially when the deviation indicator is also taken into account. Moreover, in [1], no automation mechanism for the calculations is discussed, and while the QSIG graph is used to make decisions as a separate document with no connection to the functional parts, the integration of NFRs with functional parts are actually done at the code level through the notation of *classpects* [31] and irrespective of the constructed graph. In contrast, we enable the integration of NFRs with functional parts at the model level and the analysis of NFRs is also done automatically. In the case that the code is to be generated from the models later on, the concept of classpects could be considered as an interesting method for the integration of NFRs in the implementation code, if the code is based on an aspect-oriented and object-oriented language (as classpects is basically a concept unifying classes and aspects for such languages). The work in [2] introduces FQQSIG which is a fuzzy quantitative and qualitative softgoal interdependency graph representation for analysis of NFRs in trustworthy software, however it offers no solution for the integration of NFRs with other parts of the system. On the other hand, although both QSIG and FQQSIG approaches provide solutions for evaluating different design alternatives, one subtle but important difference that our suggested modeling solution has is that the main idea in our work is to maintain the NFR model throughout the development process and perform analysis whenever and as many times as needed, such as when a new requirement is added or an existing one is modified, as well as when a new design model is created which should be evaluated and compared with the old one in terms of the satisfiability of its NFRs. Such an approach and vision on NFRs is important in managing NFRs throughout the development lifecycle, particularly, considering all the related challenges of NFRs which we discussed in this paper.

Another important work in the area of evaluation of different systems designs and architectures, and identifying the trade-offs of competing quality attributes is the Architecture Trade-off Analysis Model (ATAM) [7]. It is a spiral model of design and risk mitigation process that helps to find the dependencies among quality attributes which are referred in ATAM as trade-off points. These trade-off points are considered to be caused and derived from architectural elements that are important for and affected by multiple attributes. This method is helpful at the beginning of development process to evaluate different designs and architectures and select one, however, it does not help that much to address the challenges of NFRs that we discussed in this paper such as integration with functional requirements, and its usefulness also decreases when a more fine-grained analysis is needed [1]. Automation of this analysis approach and thus its applicability for large and complex systems is another weakness of this method, particularly, in cases where trade-off analysis might need to be done several times during the development process and lifecycle of a product.

While deciding on the satisfaction of NFRs is mainly considered to be subjective, there are several works that try to provide quantifications for NFRs to ease their evaluation and analysis. Kassab et al. in [3, 32] offer a method to quantify NFR size in a software project based on the functional size measurement method to help with the estimation of effects of NFRs on the effort of building the software in a quantitative manner. In a more recent work in [33], Kassab also proposes to incorporate Analytical Hierarchy Process (AHP) with the NFR framework. AHP is a mathematical based trade-off technique whose combination with the NFR framework enables to quantitatively deal with ambiguities, trade-offs, priorities and interdependencies among NFRs and operationalizations. An approach is introduced in [4] which makes use of Requirements Hierarchy Approach (RHA) as a quantifiable method to measure and manipulate the effects that NFRs have on a system. It does so by capturing the effects of functional requirements. In [34], an approach for quantifying NFRs based on the characteristics of and information from execution domain, application domain and component architectures is suggested. Moreover, an interesting quantitative approach for discovering the dependencies of quality metrics and identifying their impacts in the architecture of a system is provided in [35].

While models used to be thought mainly just as another form of documentation during the development process, with the introduction of model-based development and further maturation of this field, models have got a more important role as in the automatic generation of code and performing different types of analysis at earlier phases of development, and thus saving time and effort by identifying problems earlier. Aligned with this direction, there are several works that provide different forms of solutions for modeling requirements. For modeling SIG and concepts of NFR framework to represent NFRs as UML elements, a UML profile is provided in [36] to help with integration of the graph of NFRs with functional parts of the system (that are modeled in UML). Considering that NFRs and design decisions are usually specified in an informal way and as a separate document with poor or no traceability to architectural elements, [37] offers two UML profiles for modeling design decisions and NFRs as first-class entities in software architecture and to maintain traceability between them and architectural elements in the system. The profile for modeling NFRs in this work, offers six stereotypes for modeling reliability, security, performance, modifiability, and scalability each with their own specific and different set of fixed properties, such as a property called 'effort' for modifiability requirement, and 'response_time' for performance. In contrast, in our work, we have tried to provide a generic way for modeling for all NFRs regarding of their specificities (i.e., performance or security, etc.), and more importantly, with the goal of enabling designers to perform trade-off analysis on them.

In the telecommunication domain, the Telecommunication Standardization Sector (ITU-T) [38] has suggested User Requirements Notation (URN) for modeling requirements which consists of Goal-Oriented Requirement Language (GRL) and Use Case Maps (UCM). GRL is basically defined to models goals and non-functional requirements in the form goals and sub-goals, while UCM is used to describe functional scenarios. There are also some works done to define these languages as UML profiles such as [39] for GRL. As another example, for modeling security requirements, UMLsec [40] is suggested that comes with an analysis suite which enables performing analysis on the model to identify violations of security requirements. SysML [41] which is both an extension and subset of UML 2 was offered by Object Management Group (OMG) for system engineering. SysML enables to represent requirements as first-class model elements by providing a package for generic modeling of requirements (both NFRs and FRs) and the relationships among them. Different types of associations which are provided in SysML to model the relationships between the requirements include: *copy*, *deriveReqt*, *satisfy*, *verify*, *refine* and *trace*. While SysML does not specifically focus on NFRs and analysis of them, our approach and SysML can be used together to complement each other. EAST-ADL [42] which is developed for modeling software architecture and electronic parts of automotive systems, makes use of SysML requirements semantics for modeling requirements and specializes them to match the needs of automotive domain (e.g., definition of timing, delay and safety requirements). In relation to our discussion on non-functional requirements and the difference between a requirement and a property, it is worth here to also mention the UML profile for Modeling and Analysis of Real-time Embedded Systems (MARTE) [43] which offers a rich set of semantics for modeling non-functional properties and supporting analysis of them, such as performance and schedulability analysis.

8. Summary and Conclusion

In this paper, we introduced a UML-based approach for generic modeling of NFRs and automatically performing trade-off analysis on them. By identifying and discussing different challenges related to the treatment of NFRs during the development process, we formulated what information is required to be incorporated in the models of NFRs to include them as first-class entities as part of a system's architecture and enable their trade-off analysis. Through an example, it was demonstrated how the approach can be applied and how it helps to evaluate a system design with respect to the satisfaction of its NFRs. It was also shown that using the suggested approach designers can evaluate different design alternatives and get a better idea of the satisfiability of each. Moreover, the analysis highlights problematic parts of the system through the deviation indicator value which hints to the designers which parts of the system need to be reconsidered and are good candidates for improvement, taking into account the preferences of the customers. As another contribution of this work, we applied a model transformation technique to provide support for automatic analysis of the model. The possibility to analyze models of NFRs in an automatic way is particularly essential for large and complex systems and also to ease performing the analysis as many times as needed. The latter is also useful in the evolution of software architecture [44] as requirements and features are modified or new ones are added during the lifecycle of a software product and thus analysis of NFRs (including different design alternatives) may need to be performed again and again.

It was also discussed how the introduced approach can be extended and used in other contexts and as part of other solutions such as in optimizing a system design in terms of the satisfaction of its NFRs and also for providing runtime adaptation mechanisms and to manage different QoS levels of a system. As future directions of this work, quantification of NFRs and how to evaluate and provide more accurate values for them is an interesting research topic in order to reduce possible inaccuracies related to their subjective specifications. Extending our approach to incorporate other available methods such as FQQSIG [2] in which NFRs and their related relationships are specified in a qualitative manner and then through a fuzzification process quantitative values are determined for them could also be another possible direction of this work. Along with this goal, it would be interesting to include several algorithms and

methods in the analysis engine which the user may then select to use, and offer the approach as a complete tool suite. One point to remember though is that since the evaluation of NFRs and quality attributes is basically a subjective task, the methods and tools provided for this purpose serve actually as helpers for system designers to make better and more accurate evaluations and decisions.

Acknowledgment

This work has been partially supported by the Swedish Knowledge Foundation (KKS) through the ITS-EASY industrial research school [45] and by Xdin AB [46] in the scope of the MBAT European Project [47].

References

[1] T. Marew, J.-S. Lee, D.-H. Bae, Tactics based approach for integrating non-functional requirements in object-oriented analysis and design, The Journal of Systems and Software 82 (2009) 1642–1656.

[2] M.-X. Zhu, X.-X. Luo, X.-H. Chen, D. D. Wu, A non-functional requirements tradeoff model in trustworthy software, Elsevier Journal of Information Sciences 191 (2012) 61–75.

[3] M. Kassab, O. Ormandjieva, M. Daneva, A. Abran, "Software process and product measurement", Springer-Verlag, Berlin, Heidelberg, 2008, Ch. Non-Functional Requirements Size Measurement Method (NFSM) with COSMIC-FFP, pp. 168–182.

[4] A. J. Ryan, An approach to quantitative non-functional requirements in software development, in: Proceedings of the 34th Annual Government Electronics and Information Association Conference, 2000.

[5] N. Rosa, P. Cunha, G. Justo, Processnfl: a language for describing non- functional properties, in: Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002. HICSS. 2002, pp. 3676 –3685.

[6] Y. Liu, Z. Ma, W. Shao, Integrating non-functional requirement modeling into model driven development method, in: 17th Asia Pacific Software Engineering Conference (APSEC), 2010, 2010, pp. 98–107. doi:10.1109/APSEC.2010.21.

[7] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, in: Fourth IEEE International Conference on Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. , 1998, pp. 68 –78.

[8] T. Henzinger, J. Sifakis, The embedded systems design challenge, in: J. Misra, T. Nipkow, E. Sekerinski (Eds.), FM 2006: Formal Methods, Vol. 4085 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 1–15.

[9] B. Selic, The pragmatics of model-driven development, IEEE Software Journal 20 (2003) 19–25.

[10] B. Selic, A systematic approach to domain-specific language design using uml, in: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07., 2007, pp. 2 –9.

[11] L. Fuentes-Fernandez, A. Vallecillo-Moreno, An Introduction to UML Profiles, in: R. F. Calvo (Ed.), The European Journal for the Informatics Professional - UML and Model Engineering, Vol. V, 2004.

[12] I. Weisemoller, A. Schurr, A comparison of standard compliant ways to define domain specific languages, Springer-Verlag, Berlin, Heidelberg,2008, pp. 47–58.

[13] G. Kotonya, I. Sommerville, Requirements engineering with viewpoints, Software Engineering Journal 11 (1) (1996) 5 – 18.

[14] P. Sawyer, G. Kotonya, Chapter 2-Software Requirements, in: Guide to the Software Engineering Body of Knowledge, IEEE Computer Society, 2001.

[15] M. Glinz, On non-functional requirements, in: 15th IEEE International Requirements Engineering Conference, New Delhi, India, 2007, pp. 21–26.

[16] L. Chung, J. C. Prado Leite, Conceptual modeling: Foundations and applications, Springer-Verlag, Berlin, Heidelberg, 2009, Ch. On Non- Functional Requirements in Software Engineering, pp. 363–379.

[17] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.

[18] Systems and software engineering – Vocabulary (IEEE Standard), ISO/IEC/IEEE 24765:2010(E).

[19] N. S. Rosa, G. R. R. Justo, P. R. F. Cunha, A framework for building non-functional software architectures, in: Proceedings of the 2001 ACM symposium on Applied computing, SAC '01, ACM, New York, NY, USA, 2001, pp. 141–147.

[20] M. Saadatmand, A. Cicchetti, M. Sj¨odin, Uml-based modeling of non- functional requirements in telecommunication systems, in: The Sixth International Conference on Software Engineering Advances (ICSEA), 2011.

[21] A. Borg, A. Yong, P. Carlshamre, K. Sandahl, The bad conscience of requirements engineering : An investigation in real-world treatment of non-functional requirements, in: Third Conference on Software Engineering Research and Practice in Sweden (SERPS'03), Lund :, 2003.

[22] A. Borg, M. Patel, K. Sandahl, Good practice and improvement model of handling capacity requirements of large telecommunication systems, in: RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference, Washington, DC, USA, 2006.

[23] J. Mylopoulos, L. Chung, B. Nixon, Representing and using nonfunc- tional requirements: a process-oriented approach, Software Engineering, IEEE Transactions on 18 (6) (1992) 483 –497. doi:10.1109/32.142871.

[24] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Vol. 5 of International Series in Software Engineering, Springer, 1999.

[25] MDT Papyrus, http://www.eclipse.org/modeling/mdt/papyrus/ , Last Accessed: August 2013.

[26] Eclipse Modeling Framework Project (EMF), http://www.eclipse.org/modeling/emf/, Last Accessed: August 2013.

[27] QVT Operational Language, http://www.eclipse.org/m2m/ , Last Accessed: August 2013.

[28] A. Valmari, The state explosion problem, in: Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, Springer-Verlag, London, UK, UK, 1998, pp. 429–528.

[29] J. P. J. Kelly, T. I. McVittie, W. I. Yamamoto, Implementing design diversity to achieve fault tolerance, IEEE Software Journal 8 (1991) 61–71.

[30] I. Crnkovic, M. Chaudron, S. Larsson, Component-based development process and component lifecycle, in: Software

Engineering Advances, International Conference on, 2006, p. 44. doi:10.1109/ICSEA.2006.261300.
[31] H. Rajan, K. J. Sullivan, Classpects: unifying aspect- and object- oriented language design, in: Proceedings of the 27th international con- ference on Software engineering, ICSE '05, ACM, New York, NY, USA, 2005, pp. 59–68.
[32] M. Kassab, M. Daneva, O. Ormandjieva, Early quantitative assessment of non-functional requirements (June 2007). URL http://doc.utwente.nl/64134/
[33] Mohamad Kassab, An integrated approach of AHP and NFRs framework, IEEE Seventh International Conference on Research Challenges in Information Science (RCIS), vol., no., pp.1,8, 29-31 May 2013, doi: 10.1109/RCIS.2013.6577705
[34] R. Hill, J. Wang, K. Nahrstedt, Quantifying non-functional require-ments: A process oriented approach, in: Proceedings of the Requirements Engineering Conference, 12th IEEE International, IEEE Computer Society, Washington, DC, USA, 2004, pp. 352–353.
[35] A. Mentis, P. Katsaros, L. Angelis, G. Kakarontzas, Quantification of interacting runtime qualities in software architectures: Insights from transaction processing in client-server architectures, Information and Software Technology Journal 52 (12) (2010) 1331–1345.
[36] S. Supakkul, A uml profile for goal-oriented and use casedriven representation of nfrs and frs, in: In Proceedings of the 3rd International Conference on Software Engineering Research, Management and Applications, 2005, pp. 112–121.
[37] L. Zhu, I. Gorton, Uml profiles for design decisions and non-functional requirements, in: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 8–.
[38] Telecommunication Standardization Sector (ITU-T), http://www.itu.int/en/pages/default.aspx, Last Accessed: August 2013.
[39] M. R. Abid, D. Amyot, S. S. Som´e, G. Mussbacher, A uml profile for goal-oriented modeling, in: Procs. of SDL'09, 2009.
[40] J. Jurjens, Umlsec: Extending uml for secure systems development, in: UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, London, UK, 2002, pp. 412–425.
[41] OMG SysML Specifcation, http://www.sysml.org/specs.htm, Last Accessed: August 2013.
[42] EAST-ADL Specification V2.1, http://www.atesst.org Last Accessed: August 2013.
[43] OMG, MARTE specification, http://www.omgmarte.org, Last Accessed: August 2013.
[44] H. Pei-Breivold, I. Crnkovic, M. Larsson, A systematic review of software architecture evolution research, in: Journal of Information and Software Technology, Elsevier, doi:10.1016/j.infsof.2011.06.002, 2011.
[45] ITS-EASY post graduate industrial research school for embedded software and systems, http://www.mrtc.mdh.se/projects/ itseasy/ , Last Accessed: August 2013.
[46] Xdin AB, http://xdin.com/ , Last Accessed: August 2013.
[47] MBAT Project: Combined Model-based Analysis and Testing of Embedded Systems, http://www.mbat-artemis.eu/home/, Last Accessed: August 2013.