Modeling and Trade-off Analysis of NFRs

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

***Abstract*—In this paper we introduce a generic approach to analyze system design models with regard to the satisfaction of their Non-Functional Requirements (NFRs) to enable the evaluation of their NFRs' trade-offs. NFRs and their satisfaction become especially critical and deserve more attention in certain application domains such as real-time and embedded systems. This is mainly due to the constraints and resource limitations in these systems. A design that cannot achieve the functionality of the system under these limitations can mean a failure. However, one big issue is that NFRs are interconnected and cannot be considered in isolation as they can have direct impacts on each other like security and performance. This means that a careful balance and trade-off analysis among NFRs is necessary. In doing so, the role of functional parts that contribute and are implemented to satisfy an NFR should also be taken into account. We focus on these needs and identify what information about NFRs is required in order to perform trade-off analysis and comparison of design models. We propose and explain our approach to incorporate this information into system models using UML profiling method to annotate model elements with necessary information and then calculate satisfaction values of NFRs using model transformation technique.**

***Index Terms*—Non-Functional Requirements, Trade-off Analysis, UML, MBD, MDA.**

I. Introduction

The role of Non-Functional Requirements (NFR) throughout the software development process is gaining more and more attention, especially that the improper handling of NFRs has been identified as an important source of failure in many projects [1], [2]. However, NFRs are still rarely taken into account seriously and treated as first-class elements in software development, and are mostly considered as an after-thought in the final phases of development [3], [4]. There are several reasons that contribute to this fact. For example, NFRs are usually stated in an informal way and at a high abstraction level, therefore, appropriate methods and tools are required that can incorporate them at earlier phases of development and along with functional requirements. The importance of the integration of NFRs with functional parts becomes more apparent when we realize that different NFRs on the same functionality can result in different design decisions and implementations [5].

Proper handling of NFRs is even critical in certain domains such as embedded and real-time systems. Successful design of these systems depends heavily on the satisfaction of their non-functional requirements. This is mainly due to the constraints and limitations of these systems in terms of available resources [6]. Therefore, an embedded system needs to achieve its functionality under these limitations and NFRs. The problem is that each design decision can have impact on the system's NFRs and the system designer should be able to identify and evaluate these impacts. For example, if there is an execution time constraint on a component responsible for sorting numbers, then choosing a slower algorithm than a more time-optimized one can lead to the violation of system requirements and deviation of its behavior from the desired one. This situation becomes more complicated when we realize that NFRs not only crosscut different parts of a system, but can also have mutual impacts on each other. For example, choosing a time-optimized and faster sorting algorithm can help with the satisfaction of the timing requirements but may require more memory and thus violate memory constraints. Also, an NFR such as security crosscuts different parts of a system (e.g., user interface, database, communications and network transmissions) and affects some other NFRs like performance [7]. Therefore, the tools and methods that are suggested for handling of NFRs should not only be able cover and cope with their informal nature, high abstraction level, and integration with functional ones, but also should be able to help with identifying their dependencies and impacts on each other to enable analysis of their trade-off.

Model-Based Development (MBD) helps to raise the abstraction level and cope with the design complexity of systems. This can make it especially interesting for the design of real-time embedded systems which can have high degree of complexity. By raising the abstraction level, MBD also enables analysis at earlier phases of development which in turn enables identification of problems before reaching the implementation phase [8]. Since NFRs have a high abstraction level [3] and also MBD provides views of the system at higher abstraction levels, providing an MBD approach to incorporate NFRs into models is deemed more appropriate especially in integrating NFRs with functional aspects of the system.

In this paper, we propose a UML profile [9] for trade-off analysis of generic NFRs. The analysis in this work implies analyzing the dependencies and impacts of NFRs (irrespective of their type, e.g., performance, security…) as well as the functional features of the system to provide system designers with a better insight into how good a system design is in terms of the satisfaction level of its NFRs. The main contributions can thus be summarized in the following points:

- providing an approach to capture and model NFRs in an open and generic way and include them with functional features as well as the dependencies and relationships

among them,

- enabling designers to evaluate and compare different design models with regard to their NFRs and identify designs which can result in better overall satisfaction of NFRs,
- identifying deviations in the satisfaction of system's NFRs and highlighting potentially problematic parts that deserve more attention.

Enabling trade-off analysis of NFRs through a UML profile has several key benefits. UML is a standard modeling solution already adopted by industry and there are many design and analysis tools based on it. By offering a UML-based solution for trade-off analysis of NFRs, it becomes possible to make use of currently available tools. Also, the learning curve for developers already using UML will be less which implies both cost and time savings for companies [9]. Moreover, the approach we propose here tries to extend design models with necessary information to enable trade-off analysis of NFRs, and thus can enable using already designed system models and can save modeling efforts. We propose stereotypes in our profile by which model elements can be decorated and related with the necessary information that enable designers to consult models for specific information about NFRs and trade-off analysis of them.

The remainder of the paper is structured as follows. In Section II, we have a look at NFRs in general, including their definition, characteristics, and the issues around them. In Section III, we discuss the required characteristics and information that a solution for managing trade-offs of NFRs should be able to offer. Section IV explains the profile structure and concepts. Section V shows an application of the profile in a portion of a mobile phone design and the analysis is then performed on the annotated model. In Section VI, we will have a look at other related works and finally in Section VII, conclusions are made and future directions are explained.

II. Non-Functional Requirements

Requirements are generally divided into two main categories: functional and non-functional. In the simplest form, functional requirements are those which define what the system should do, while the term non-functional requirement is used for requirements which specify how a system should perform or as suggested in [10] "a non-functional requirement is an attribute of or a constraint on a system". There is a big number of suggested definitions for non-functional requirements which are discussed in [11]. These requirements are usually described with terms that end with 'ility' such as availability, 'ity' such as atomicity, while a few other ones such as performance and user-friendliness do not follow this pattern. According to the IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E) [12], [13], the following definitions is provided for non-functional requirements: "a software requirement that describes not what the software will do but how the software will do it (i.e., design constraints)".

One concept that is often confused with NFR is the concept of Non-Functional/Extra-Functional Property (NFP/EFP). The former can be considered as an expression of a need (possibly informal), and the latter as a statement that is usually asserted formally and can be therefore proven and analyzed (e.g., calculated response time of a component). This implies that "a requirement can require that a certain property holds (e.g., absence of deadlock, meeting deadline, not overflowing a queue, etc.) and that in order for a property to hold a number of requirements may have to be met, which we normally neither express nor assert formally" [1]. For example, "response time of a component should not exceed 2ms" is a requirement, while "response time of component A never exceeds 2ms" and "response time of component B is equal to 1ms" are expressions of properties in a system. In [14], 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.

NFRs have several characteristics that make their consideration in software development process a challenging task. While Functional Requirements (FRs) are typically realized and implemented one by one and step by step as part of the software product, NFRs do not usually follow this pattern and the design decisions taken to implement the functionality of the software affect their satisfaction. Similarly, while FRs normally have localized effects, NFRs are basically specifications of global constraints (e.g., performance, security, availability, etc.) to be satisfied by the software [3]. In this regard, NFRs can crosscut different parts of the system (e.g., security). Also, NFRs are interconnected and there are dependencies among them which implies that the satisfaction of one NFR can conflict and impair the satisfaction of other ones. Therefore, trade-off analysis is required to identify such impacts and establish balance among NFRs in a system.

On the other hand, NFRs are usually specified in an abstract and informal way [3], [14] and thus, providing a more formal approach using model-based development which tries to raise the abstraction level of systems can help with the treatment of NFRs during the software development. There are several reasons for incorporating NFRs in the development process and explicitly dealing with them. Among them are the increasing number of applications such as in real-time embedded systems where NFRs play a critical role, and also the strong interaction between functional and non-functional requirements. Moreover, an explicit treatment of NFRs facilitates prediction of quality properties of the final product in a more reliable and reasonable way [14].

The approaches that are suggested for the explicit treatment of NFRs are sometimes categorized into two groups: product-oriented and process-oriented [15]. 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

[1]These definitions have been provided and formulated with the help of Prof. Tullio Vardanega, University of Padova, Italy

the software in terms of its NFRs [14], [15].

III. Characteristics of the Solutions

Based on the identified challenges that were discussed previously, in this section we formulate the key characteristics that a model-based solution for representation and trade-off analysis of NFRs should have.

Traceability of design decisions related to an NFR: considering that NFRs usually crosscut different parts of the system, the designer should be able to understand which parts of the system contribute (both positively and negatively) to a specific NFR; for example, an encryption component that is intended to satisfy security requirements. With this information, after trade-off analysis, the designer can identify parts of the systems that should be removed, replaced, improved or kept.

Traceability among NFRs: throughout the whole design process of a system, higher level NFRs are refined and broken down into more concrete ones, particularly in a top-to-bottom approach. For example, a high level and abstract NFR such as security can be refined into access control or encryption requirements at lower levels. Therefore, in order to check the satisfaction of security in the system, it is necessary to keep track of its refinements and lower level requirements that cover different aspects of security along with information on how much each one contributes towards its parent NFR.

Satisfaction level of an NFR: it should be possible to evaluate the total satisfaction level/degree of an NFR in the system. This is necessary to compare current design against system specification and customer requirements as well as different design alternatives. In the end, the goal is that the designer gets an idea to what extent each NFR is satisfied. Judging where this level is acceptable or not is done as the next step after applying the approach we suggest in this paper and probably by checking it with the stakeholders.

Impact of an NFR on other NFRs: as mentioned before, NFRs cannot be considered in isolation in a system without taking into account their impacts on each other. Therefore, it is required to identify and evaluate the effects that a model element and design decision that is used to satisfy one NFR, has on another NFR in the system. For example, performing heavy computations by an encryption component in an embedded system can also mean consuming more battery. Therefore, the side effects of such design decision should be identifiable for the designer.

Priority of an NFR: not all NFRs have the same importance in the system. In order to increase the overall satisfaction of NFRs and also to resolve conflicts among NFRs (reduce the impact of one NFR in favor of another), it is necessary to know the importance of each NFR and compare them. Considering priority for NFRs is also important in order to capture the preferences of customers/users. Similarly, priorities can also be considered for different features implemented to satisfy an NFR.

Coherent terms for NFRs: one subtle problem with NFRs that is more often noticed in large enterprises is that different (sub)departments may have different interpretations for an NFR term or use different terms to refer to an NFR [16]. Therefore, it is important to provide a coherent and consistent notation for defining NFRs and relating them to design elements.

Coherent measurements of NFRs: to compare different NFRs and perform trade-off analysis among them, evaluation of the satisfaction degree and impacts of NFRs should follow a coherent representation. In other words, the criteria or metrics 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).

IV. Suggested Profile

Figure 1 shows the structure of our profile to include the necessary information mentioned in the previous section in models to enable trade-off analysis of NFRs.

`System`: this stereotype is used to annotate the system which is the context of the analysis and to which different NFRs belong. It includes satisfactionValue tagged value to represent quantitatively the total satisfactions of the system's NFRs.

`NFR`: each NFR is identified using this stereotype. A higher level NFR (in terms of abstraction level) can be refined into one or more other NFRs. Therefore, there is an association relationship to itself (reflexive aggregation).

`Feature`: this stereotype represents a feature in the system that contributes to the satisfaction of an NFR and is an equivalent of *Operationalization* concept in NFR framework and Softgoal Interdependency Graph (SIG) [17] (described later in the paper) or *tactics* as used in [4].

`NFRContributes`: this stereotype which is used on relationships between model elements shows that an element (NFR or Feature) contributes directly to the satisfaction of another element. The contributionValue property of this stereotype is used to specify 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`: is used to relate two or more elements that cooperate together to satisfy an NFR. This is similar to the AND relation in NFR framework and SIG.

`NFRApplies`: the relation between NFR related model elements and functional ones can be modeled using NFRApplies stereotype (e.g. an NFR that applies to a component).

`Rationale`: this property and tagged value in NFR and Feature stereotypes can be used to include the description and rationale for an NFR, its refinements into other NFRs or Features implementing it.

`Priority`: this property in NFR and Feature stereotypes is used to capture customer preferences and priorities in terms of the importance of NFRs and Features. This helps to identify

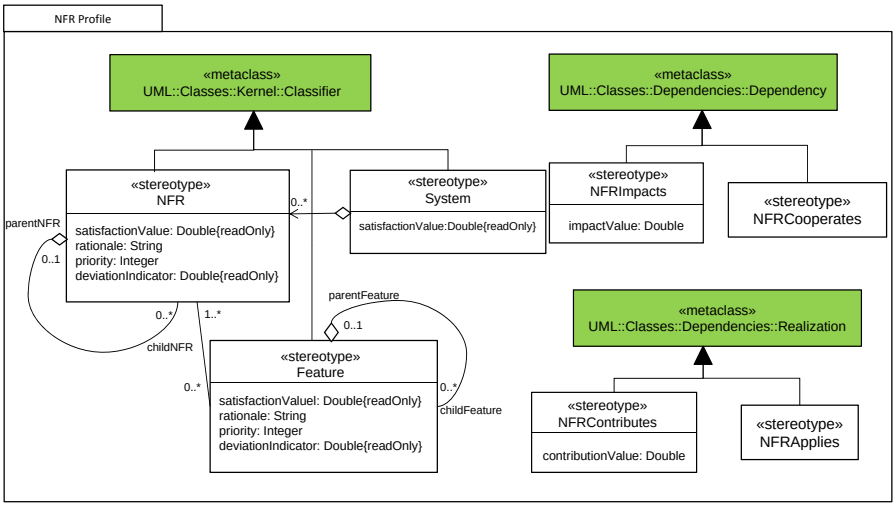

Fig. 1. Profile for Trade-off analysis of NFRs

less important parts in case modification and removal of some features or NFRs is necessary.

`DeviationIndicator`: is a read-only property which is calculated and set automatically. DeviationIndicator takes into account the priority and satisfaction value, and provides a number which indicates to the designer the importance and magnitude of how much the satisfaction of an NFR or Feature has deviated/violated. While the satisfaction value does not reflect user preferences and priorities, the deviation indicator value shows which parts of the system deviated more from the specification (i.e., being fully satisfied) and need to be modified to achieve a better satisfaction level.

There are also several rules on the elements and their relationships described above:

- The allowed range of values to set is between -1 and 1 (except for priority). For example a negative value on the NFRImpacts relationship shows the negative impact of the source element on the target element.
- The satisfaction value for each leaf node is considered to be 1.
- Allowed values for priority are: 1 (very low), 2 (low), 3 (medium), 4 (high), 5 (very high).
- The sum of contribution values of the links connecting children nodes (refinement/lower level elements) to their parent should be less or equal to 1 (maximum is 1).
- Contribution of a child node to its parent is calculated as the satisfactionValue of the child node multiplied by its contributionValue or impactValue.
- The satisfactionValue of a node is, therefore, calculated as the sum of the contributions from all of its children nodes plus the sum of the impacts of all other nodes, divided by the total number of NFRImpacts relationships to it plus 1. In other words, 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), and i_j is the impact value of another node on this parent node, the satisfaction value of the parent node is calculated as:

$$\frac{\sum s_k * l_k + \sum^n i_j}{n+1} \tag{1}$$

This calculation is performed starting from leaf nodes (considering that the satisfaction of leaf nodes is 1) and is calculated recursively upwards toward the top element which is the system.

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

$$\begin{aligned} &DeviationIndicator \\ &Priority - Priority * SatsifactionValue \end{aligned} = \tag{2}$$

Based on this calculation and considering that the SatisfactionValue is always between -1 and 1 and priority is an integer value between 1 and 5, the value of DeviationIndicator will be in the range of $[0, 10]$. 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.

V. Implementation and Usage Example

The profile concepts described in the previous section were implemented in MDT Papyrus [18]. In Figure 2 an example usage of the profile is shown on parts of a mobile phone system. Two NFRs are defined. One on the quality of the taken camera picture and the other one on the battery life. There are two features that contribute to the quality of taken photos: usage of a flash and type of the lens. Also two features are considered for `Battery Life`: adjustment of screen's brightness level and auto standby feature. The contribution of each feature to its parent NFR is annotated using NFRContributes stereotype and its contributionValue attribute. In this system, the use of flash consumes lots of battery. The impact of the `Flash` feature on `Battery Life` is therefore specified using NFRImpacts stereotype and its impactValue attribute which is -0.8. Preferences of the customer in terms of the

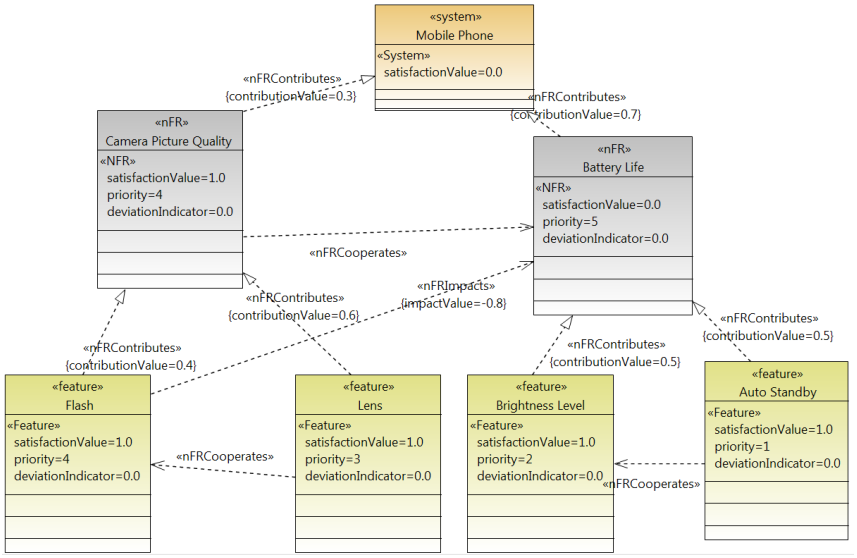

Fig. 2. Example usage of NFR profile for a Mobile Phone (before calculations)

relative importance of NFRs and Features are captured through the priority property of each model element. For example, `Battery Life` has priority level 5 which means it is more important than the `Camera Picture Quality` which has priority level 4. Satisfaction and DeviationIndicator values are initially 0 as no calculation has yet been done on the model.

One point here is that, although these values are assigned subjectively by the system designer, there are methods such as sensitivity analysis [4] that help to increase the confidence in the chosen decision. Now what we need to understand is the impact that having the `Flash` feature has on the system.

By having the necessary information in the model, it is now possible to perform analysis on the model to determine impacts of each design decision on the system and evaluate their dependencies. To traverse the model and perform calculations, we have developed a model-to-model (M2M) transformation using QVT Operational language (QVT-O) [19] in Eclipse [20]. It reads as input a UML model annotated with our profile, traverses the nodes and calculates satisfaction values and writes the results back in the same model. In other words, we use an in-place transformation (i.e. input and output models are the same).

To calculate the satisfaction values, a recursive algorithm is used in the model transformation based on the rules mentioned in the previous section. For each node, all the incoming links that have NFRImpacts or NFRContributes stereotypes are retrieved. If a node does not have any such links (meaning that it is a leaf node), its satisfaction value is set to 1. Otherwise, the source of the link, which is another node, is retrieved and the algorithm continues by calculating the satisfaction value of the source node; hence the recursion.

The algorithm basically applies Formula 1 which is mentioned in the previous section. For example, in Figure 2, the satisfaction values for `Auto Standby` and `Brightness Level` features are set to 1, since they are leaf nodes. The satisfaction value of `Battery Life` is then calculated as the satisfaction value of `Brightness Level` (1) multiplied by the value on the link that connects it to `Battery Life` (0.5), plus the same multiplications for `Auto Standby` (1*0.5) and `Flash` (1*-0.8) which results in 0.20. The discrepancy observed between this calculated value (0.20) and the one in Figure 3 (0.199...) which is calculated automatically through the in-place model transformation on the model is due to the OCL implementation of real numbers that QVT-O uses.

By performing the transformation on the whole model, the satisfaction values are calculated for each node and propagated upwards toward the system element as shown in Figure 3:

$$(1*0.4+1*0.6)*0.3+((1*0.5+1*0.5)-0.8)*0.7=0.44$$

Therefore, the total satisfaction value in this case will be 0.44. Trying the procedure again on the same model but without having the `Flash` feature results in:

$$(1*0.6)*0.3+(1*0.5+1*0.5)*0.7=0.88$$

Similarly, as soon as the satisfaction value for a feature or NFR is calculated, the deviationIndicator property value is also calculated using Formula 2 and set in the model. For the leaf nodes, since their satisfaction values are always 1, their deviationIndicator value will always result in 0. For the `Camera Picture Quality` NFR, the deviationIndicator value is calculated as:

$$4-4*1.0=0$$

which means that there has been no deviation from the desired design. However, for the `Battery Life` NFR, this value will be:

$$5-5*0.2=5-1=4$$

This value shows that there exists some deviation which in

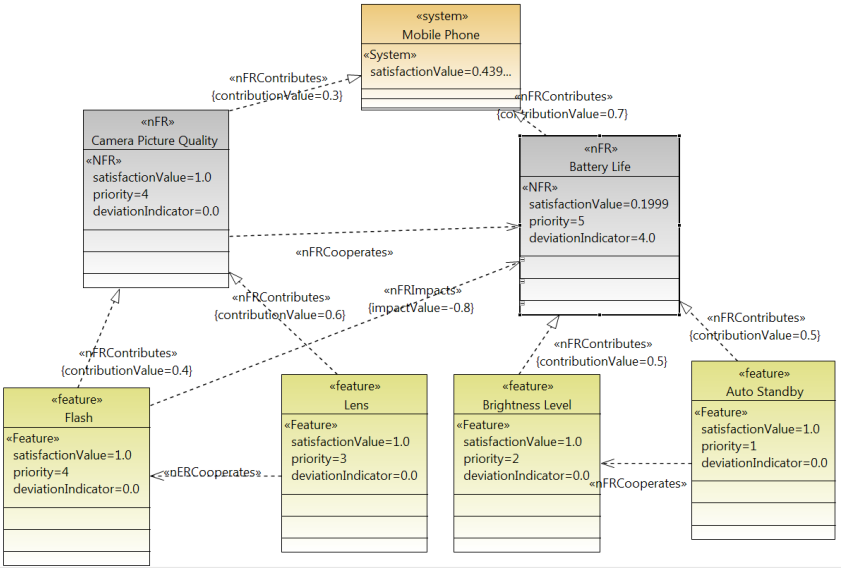

Fig. 3. Analyzed model of the system

this case is due to the side effects of having the `Flash` feature on `Battery Life`. Based on the specification of the actual system that is being developed, this could, for instance, imply that the type of flash is not good enough in terms of energy consumption for this system and needs re-consideration.

Since the calculation of the deviation indicator is based on the priority of each NFR or Feature, in a larger model where deviations in several parts of the system are observed, the user can understand on which parts of the model, modifications should be done and in which order. In other words, parts with higher deviation value indicate more critical problems and deserve more attention.

Figure 4 shows performing the analysis on the same model but without the Flash feature. This model could, for example, represent another family of mobile phones or another usage scenario. It can be seen that the total satisfaction level of the system is higher in this case: 0.88 versus 0.44 in the previous case. The removal of the `Flash` feature, however, also causes some deviation (1.6) for the `Camera Picture Quality` NFR.

Based on these calculations, our suggested approach enables a more accurate comparison of design alternatives considering the interdependencies and trade-offs among different NFRs of the systems helping designers to make better decisions. Also the introduced modeling concepts and calculations above provide for several interesting features. One feature is the possibility to optimize the total satisfaction of the system considering different design alternatives. For example, in the mobile phone system described, if the designer needs to select among several possible solutions that contribute to better image quality, he/she can find the ones that lead to the highest satisfaction value of the system. Another possibility is to have run-time adaptability or re-configuration based on different quality of service levels. For example, if the battery level goes low beyond a certain limit, the system can go into a power-saving mode using features that incur minimum impact on battery consumption or replacing active components with back-up/standby ones which may provide lower quality/fewer services but consume less battery (e.g., in design diversity techniques [21]). Without the analysis introduced here, such a decision may not only be hard, but also will be blind in the sense that the side effects of a feature/component replacement on other aspects of the system will be unknown.

VI. Related Work

There are versatile research works that try to target different issues regarding NFRs. [22] focuses on the problem of informal and separated documentation of design decisions and NFRs. To alleviate the problem, it introduces two profiles to model design decisions and generic NFRs to treat them as first-class entities in software architectures and maintain the traceability of design decisions and architectural elements. NFR framework proposed in [17] is one of the fundamental works in the area of NFRs which is a process-oriented and goal-oriented approach. It uses Softgoal Interdependancy Graph (SIG) to represent NFRs, their refinements and entities that NFRs are applied to (termed as *Operationalization*), and the interdependencies among them to include their impacts and relations. 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. It provides notations to mark critical NFRs in the graph and also an evaluation procedure to determine satisfaction and conflicts of NFRs. The approach suggested in NFR framework is basically a qualitative approach. Moreover, the criticality concept in NFR framework seems more suitable for developers and does not convey enough information for prioritization

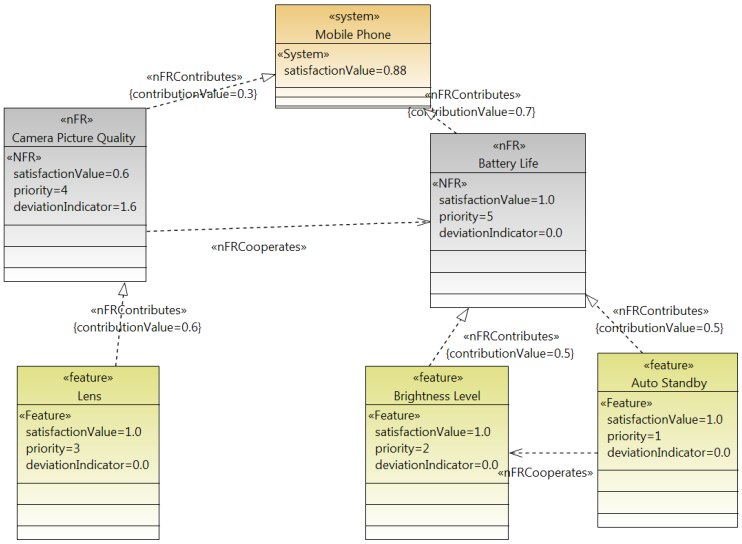

Fig. 4. Analyzed model of the system without the Flash feature

of NFRs particularly from the customer's perspective and also for performing trade-off analysis [4]. [23] offers a UML profile for modeling SIG and concepts of NFR framework to represent NFRs as UML elements in order to integrate them with functional parts of the system (that are modeled in UML). T. Marew et al. [4] introduces Q-SIG which is a quantified version of SIG that enables quantitative evaluation of impacts and trades-offs among NFRs. In this paper, we introduced modeling concepts that enable designers to apply the Q-SIG approach in the form of UML models, and provided a tooling solution for evaluation and trade-off analysis of NFRs on these models using this approach. Process^{NFL} that is introduced in [3] is a textual language for describing non-functional properties during the software development. It offers templates for describing three abstractions that capture different aspects of non-functional properties: NF-Attributes, NF-Properties and NF-Actions. Moreover, the language enables to express the relationships between these abstractions to include dependencies and impacts of non-functional properties in the system. The language is an effort for explicit treatment of non-functional properties during software development.

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 [1], [24] 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. [2] makes use of Requirements Hierarchy Approach (RHA) as a quantifable method to measure and manipulate the effects that NFRs have on a system. It does so by capturing the effects of functional requirements. In [25], an approach for quantifying NFRs based on the characteristics of and information from execution domain, application domain and component architectures is suggested.

SysML [26] which is a UML profile for system engineering also includes a package for generic modeling of requirements (both NFRs and FRs) and the relationships among them. These relationships mainly capture the traceability and hierarchy of requirements (e.g., refinement). While SysML does not specifically focus on NFRs and analysis of them, our approach and SysML can be used together to complement each other.

VII. Conclusion and Future Work

In this paper, a UML profile for modeling NFRs and their dependencies was introduced to enable performing trade-off analysis among them. It was shown how it can help to compare different design models, determine which ones achieve a higher satisfaction of the NFRs, and identify parts of the model which might be good candidates for modification to reach a higher satisfaction level in the system.

One point to note about the proposed approach is the issue of scalability and evaluating how it can be applicable for very complex and large systems. While this is basically a general concern in model-driven engineering, there are some solutions for management of large models which can be considered such as using a multi-view approach and providing better degree of separation of concerns by defining different views over the model of a system [27]. Also in this work, we assumed that the designer can provide values (though subjective) for contribution and impact relationships among NFRs and functional features. As mentioned, there are some techniques that help system designers in providing such quantitative information. However, this may also imply that our suggested approach can be especially more applicable for component-based systems where systems are built out of already existing components and thus more knowledge about their characteristics and behaviors are available (e.g., memory usage, execution time). Moreover, our approach can also be useful in the analysis of software architecture evolution, when new requirements or features are introduced into a system or existing ones are modified [28].

As the continuation of this work, we plan to develop an analysis tool as an Eclipse plug-in that can read as input, models annotated with our NFR trade-off profile and provide total satisfaction values for different NFRs, identify parts contributing negatively to an NFR, and perform calculations for overall optimization of NFRs in the system considering different design alternatives and scenarios. With the help of the tool, when some parts of the system need to be changed and updated, the user can identify the side effects of such changes on other parts and the system as a whole in terms of NFRs, at model level and before implementing the intended changes. The defined profile depicted in this paper is the first step toward enabling such features.

Using the introduced modeling concepts here along with a back-annotation mechanism in model-based development of embedded systems will also be an interesting topic to further investigate and work on. Having such a mechanism, it would be possible to monitor the system and provide feedbacks to the design model about possible violations in the system (or any of its subsystems) in terms of satisfaction levels of their NFRs. Also, the usage of the suggested approach for run-time adaptability and re-configuration of systems is another direction for further investigation.

VIII. Acknowledgements

This work has been partially supported by the CHESS European Project (ARTEMIS-JU100022) [29] and Xdin AB [30].

References

[1] M. Kassab, O. Ormandjieva, M. Daneva, and A. Abran, "Software process and product measurement," J. J. Cuadrado-Gallego, R. Braungarten, R. R. Dumke, and A. Abran, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Non-Functional Requirements Size Measurement Method (NFSM) with COSMIC-FFP, pp. 168–182.

[2] 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.

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

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

[5] Y. Liu, Z. Ma, and W. Shao, "Integrating non-functional requirement modeling into model driven development method," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, December 2010, pp. 98 –107.

[6] T. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM 2006: Formal Methods*, ser. Lecture Notes in Computer Science, J. Misra, T. Nipkow, and E. Sekerinski, Eds. Springer Berlin / Heidelberg, vol. 4085, pp. 1–15.

[7] J. Lee, K. Kapitanova, and S. H. Son, "The price of security in wireless sensor networks," *Journal of Computer and Telecommunications Networking*, vol. 54, pp. 2967–2978, December 2010.

[8] B. Selic, "The pragmatics of model-driven development," *IEEE Software Journal*, vol. 20, pp. 19–25, September 2003.

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

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

[11] L. Chung and J. C. Prado Leite, "Conceptual modeling: Foundations and applications," A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. On Non-Functional Requirements in Software Engineering, pp. 363–379.

[12] "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, 1990.

[13] "Systems and software engineering – Vocabulary (IEEE Standard)," *ISO/IEC/IEEE 24765:2010(E)*, 15 2010.

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

[15] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: a process-oriented approach," *Software Engineering, IEEE Transactions on*, vol. 18, no. 6, pp. 483 –497, jun 1992.

[16] M. Saadatmand, A. Cicchetti, and M. Sjödin, "Uml-based modeling of non-functional requirements in telecommunication systems," in *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, October 2011.

[17] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, ser. International Series in Software Engineering. Springer, October 1999, vol. 5.

[18] MDT Papyrus , http://www.eclipse.org/modeling/mdt/papyrus/, Accessed: February 2012.

[19] QVT Operational Language, http://www.eclipse.org/m2m/, Accessed: February 2012.

[20] Eclipse Modeling Framework Project (EMF), http://www.eclipse.org/modeling/emf/, Accessed: February 2012.

[21] J. P. J. Kelly, T. I. McVittie, and W. I. Yamamoto, "Implementing design diversity to achieve fault tolerance," *IEEE Software Journal*, vol. 8, pp. 61–71, July 1991.

[22] L. Zhu and 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*, ser. SHARK-ADI '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 8–.

[23] 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.

[24] M. Kassab, M. Daneva, and O. Ormandjieva, "Early quantitative assessment of non-functional requirements," Enschede, June 2007. [Online]. Available: http://doc.utwente.nl/64134/

[25] R. Hill, J. Wang, and K. Nahrstedt, "Quantifying non-functional requirements: A process oriented approach," in *Proceedings of the Requirements Engineering Conference, 12th IEEE International*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 352–353.

[26] OMG SysML Specifcation V1.2, http://www.sysml.org/specs.htm, Accessed: March 2012.

[27] M. Panunzio and T. Vardanega, "A metamodel-driven process featuring advanced model-based timing analysis," in *Reliable Software Technologies Ada Europe 2007*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4498, pp. 128–141.

[28] H. Pei-Breivold, I. Crnkovic, and M. Larsson, "A systematic review of software architecture evolution research," *Journal of Information and Software Technology*, July 2011.

[29] CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly, http://chess-project.ning.com/, Accessed: March 2012.

[30] Xdin AB, http://xdin.com/, Accessed: April 2012.