Towards a Framework to Incorporate NFRs into UML Models

Subrina Anjum Tonu, Ladan Tahvildari
Software Technologies Applied Research Group
Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
{subrina, ltahvild}@swen.uwaterloo.ca

Abstract

Despite the fact that Non-Functional Requirements (NFRs) are very difficult to achieve and at the same time are expensive to deal with, a few research works have focused on them as first class requirements in a development process. We propose a framework to incorporate NFRs, as reusable components, with standard UML notations. Such a framework can also integrate those reusable NFRs with the extracted UML representations of legacy systems during the reverse engineering process. This novel research work uses standard XMI representation of UML models without proposing any extension to it. As a proof of concept, a small case study of a Credit Card System is presented.

1 Introduction

The demand for high quality software system is increasing day by day. Production of a highly organized software system requires separation of concerns [7] which is one of the basic engineering principles. On the other hand, production of a high quality software requires the implementation of all functional and non-functional requirements starting from the design phase to the end of the software life cycle.

As known, all these requirements are changing during the maintenance phase of any software system. The re-engineering of such software systems have gained significant attention in today's software industry. A few research works provide a re-engineering process that addresses such problems in order to incorporate any new or modified (functional and non-functional) requirements. Existing reverse engineering process can extract architectural design of the legacy systems which can be presented in UML model. UML tools also exist to automatically generate deployable source code from UML model specifications. In such an environment, the legacy systems can be modified by adding the NFRs with the extracted UML model out of such system and the source code can be re-generated automatically. In a nutshell, it is necessary to have an environment to attach the NFRs to the target system.

Tahvildari et al. proposed a quality-driven reengineering (QDR) framework which allows specific quality requirements for the migrant systems to be modelled as a collection of soft-goal graphs, and provides a selection of the transformational steps that needs to be applied at the source code level of the legacy system being reengineered [14]. They extended their work and proposed a framework of transformations that aims to improve error-prone design properties and assists in enhancing specific qualities of a software system using a catalogue of OO software metrics [13].

This research is an extension to that work by focusing on the extracted UML representation (from source code) of the legacy systems rather than AST. We also focus on making reusable NFRs and attaching them with the target model. In current practice, the join-point (where the NFR touches the target model) is defined as a part of the NFR itself. As a result, there is very little chance to reuse this NFR in other software design. This research work is a step to remove these shortcomings. First, we identify the functional requirements (FR) and non-functional requirements (NFR) of a legacy system which needs to be re-engineered. Second, we specify the new FRs with the appropriate UML diagrams and we specify a template of NFRs using NFR framework [4] and our proposed notations for creating standard UML diagrams. According to this approach, the NFRs do not have any hard coded join-points inside it. We used the concept of *dynamic parameterizations* described in [9]. We also use a knowledge-based concept for building a repository/library of those reusable NFRs. Finally, we integrate those NFRs with the target model where the necessary parameters for defining the join-points come dynamically during run time. We use UML as our design language as it is the most popular modelling language in research community, as well as a general purpose object-oriented language [1]. Our proposed framework is developed in a standard XMI environment.

Paper	Class Diagram	Use Case Diagram	Sequence Diagram	Interaction Diagram	State Diagram	Collaboration Diagram	Need Extension of UML
Lawrence Chung et.al [12]		√					√
Ana Moreira et.al [10]		√		√			√
Luiz Marcio Cysneiros et.al [5]	√	√	√			√	√
Evgeni Dimitrov et.al [6]		√		√	√		

Table 1. Summary of Related Works

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the framework while Section 4 applies the framework on a case study. Finally, Section 5 summarizes the contributions of this work and outlines directions for further research.

2 Related Works

The idea of integrating NFRs with FRs in design level is not a new one. As shown in Table 1 many researchers proposed extensions of UML model for the integration purpose.

Lawrence Chung et al. [12] proposed to integrate NFRs with FRs in UML use case model. They implemented the NFRs by their NFR framework [11, 4] and proposed to associate those NFRs with four use case model elements: actor, use case, actor-use case association and the system boundary. They named these associations as “Actor Association Point”, “Use Case Association Point”, “Actor-Use Case Association (AU-A) Association Point” and “System Boundary Association Point”, respectively.

Ana Moreira et al. [10] proposed a model for integrating crosscutting quality attributes with FRs by UML use case diagram and interaction diagram. They proposed a template for quality attributes with some specific fields (such as description, focus, source, decomposition) and they integrated those quality attributes with FRs by using standard UML diagrammatic representations (e. g. use case diagram, interaction diagrams) extended with special notations.

Luiz Marcio Cysneiros et al. [5] proposed a systematic approach to assure that conceptual models will reflect the NFRs elicited. They uses a vocabulary anchor (LEL) to build both functional and nonfunctional perspectives of a software system. They also showed how to integrate NFRs into UML by extending some of the UML sublanguages, and they presented a systematic way to integrate NFRs into the functional models.

Evgeni Dimitrov et al. [6] described three approaches for UML-based performance engineering. The three approaches are: Approach-1) Direct representation of performance aspects using UML, Approach-2) Expanding UML to deal with performance aspects and Approach-3) Combining UML with formal description techniques. They proposed some extensions to UML use case and state transition diagram.

Our work is different from all these works in the sense that we do not apply our framework for a specific UML diagram, rather than we apply it in a general way for all types of UML model. Besides, we do not propose any extension to UML model, rather than we express the NFRs in a reusable way with the standard UML notations.

3 A Proposed Framework

Building of reusable NFR templates and the integration of them with the extracted UML models of the legacy systems requires a comprehensive framework to relate the integration process with the functional requirements of the target model. The focal point of the proposed research is to exploit the synergy between the area of software requirements analysis (both functional and non-functional) and software re-engineering.

We assume an extracted UML model is available from a legacy system after a reverse engineering process is applied. Our framework starts with such extracted UML model of a legacy system. It consists of three phases as depicted in Figure 1: 1) Identification of FRs and NFRs, 2) Specification of FRs and NFRs and 3) Integration of NFRs.

- **Identification of FRs and NFRs**

  From design documents, release notes, source code, extracted UML model and new user’s requirements for a software system, we identify the new functional and non-functional requirements which need to be added to the system being re-engineered. Our focus is mainly on the desired non-functional properties of the software that it should meet to assure high quality software system.

- **Specification of FRs and NFRs**

  This phase consists of two parts. The first part is to specify the new FRs to be added into the extracted legacy model using standard UML notations. The second part is to search our knowledge-based NFR repository for any similar previously designed reusable NFR that the system may need. In case of the existing NFR design matches partially with the required NFR, the former one needs to be modified according to the required one and to be stored in the repository for future use. If no such NFR can be found, a new NFR template will be created according to the requirements.

- **Integration of NFRs**

  After specifying all non-functional requirements this phase just becomes a NFR weaver that weaves those desired NFRs with the FRs of the target system as shown in Figure 1. The following section elaborates

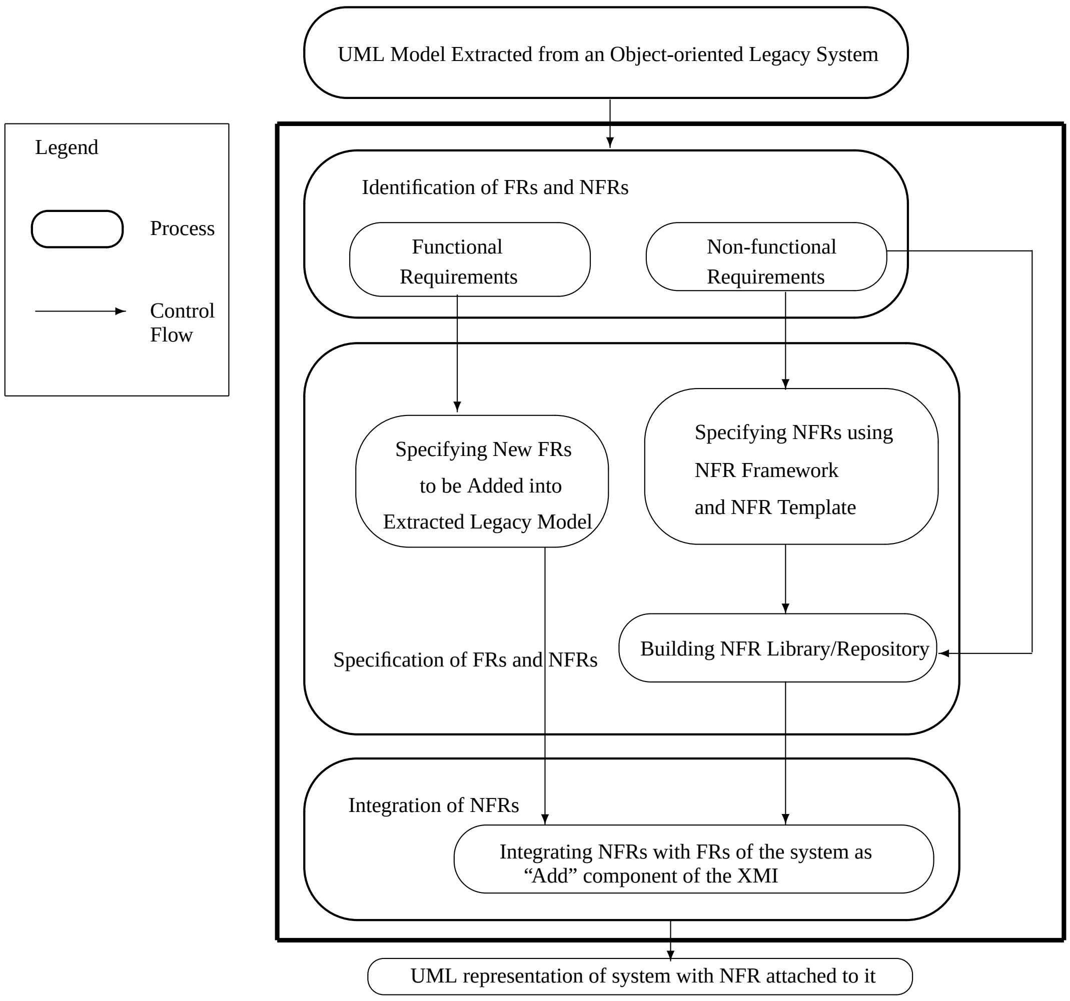

UML Model Extracted from an Object-oriented Legacy System
Legend
Process
Control Flow
Identification of FRs and NFRs
Functional Requirements
Non-functional Requirements
Specifying New FRs to be Added into Extracted Legacy Model
Specifying NFRs using NFR Framework and NFR Template
Building NFR Library/Repository
Specification of FRs and NFRs
Integration of NFRs
Integrating NFRs with FRs of the system as "Add" component of the XMI
UML representation of system with NFR attached to it

Figure 1. A Framework for Integrating NFRs

further on our proposed approach to make the reusable NFRs and to integrate them with the target model.

3.1 A Model for NFR Representation

A number of researchers and practitioners examined how a software or system successfully achieves quality attributes [2, 3, 4, 8]. To represent information about different software qualities, their interdependencies, evaluation of the NFRs upon the target system, detail techniques for specifying methods to arrive at the "target" or "destination" of the design process (operationalizing softgoal), we adopt the NFR framework proposed in [4]. The visualization of the operations of the NFR framework is done in terms of the incremental and interactive construction, elaboration, analysis, and revision of a *Softgoal interdependency graph (SIG)*.

Our work begins after evaluation of SIG. The evaluation procedure, defined by NFR framework, results in a subgraph of the SIG that needs to be integrated with the functional requirements of the target model. Finally, the *target system* in NFR framework describes the final solution of the particular NFR. Our framework maps this solution to achieve a particular NFR with standard UML notations and provides a weaver to weave it with the UML representation of a software system.

3.2 A UML Representation for NFR

We propose a high level notation to design the final solution of the target system to achieve the particular NFR with standard UML notations. Our proposed notation is based on the general actions that can be performed on any entity of UML model. For example, creating a class, inserting attributes to a class, inserting methods to a class, deleting a class, deleting attributes from a class, creating states, creating state-transitions, and so on. The goal of this notation is to provide all the actions which may be needed to create any type of UML diagram, and to specify a template for the UML representation of the NFR. The template is not executed at this point, rather it becomes an ordered set of instructions/commands (similar to a script in UNIX). While the framework is attaching the NFR to the functional requirements of the system, the template commands need to be executed to build the proper UML representation of the NFR.

3.3 A NFR Library/Repository

We use the same knowledge-based approach as NFR framework. We propose to build a NFR library/repository where the past experience, standard techniques and knowledge about particular NFR, the evaluated subgraph of SIG for achieving that NFR and the proposed NFR template to integrate it with the target model can be stored for future use. The library/repository is likely to evolve in the course of time and can help the developers in saving time by supplying previously designed NFR templates as reusable components.

3.4 Weaving NFR with the Target Model

A meta-model in UML describes the UML model by itself . Hence, the manipulation of the meta-model is same as the manipulation of any UML model. For this purpose, we have developed a meta-level NFR weaver where the weaver executes the weaving operations as specified in each NFR template applied onto the initial model. The weaver actually executes the commands specified in the NFR template and generates the corresponding UML representation of the particular NFR. The target model is also a UML model with NFR attached to it. For the compatibility of other UML tools, we are using the standard XMI to generate the UML model. In our framework, the NFR description resulting from SIG is a part of comment inside the XMI and the UML representation of the NFR is a part of the "Add" component of the XMI. By adding the NFR description resulting from NFR framework with the XMI of the model as a comment, we can store all the information for a particular NFR which can be further viewed using our weaver. The output becomes compatible with other tools as it still is in standard XMI format. By adding the NFR as UML "Add" component we can also separate the NFR from the main design of functional requirements of a software system and other operations can also be done on the added NFR. For example "deletion" and "modification"of the NFRs can be done without changing the main design of a software system.

4 A Case Study

A prototype has been written in Java programming language to implement the proposed framework in a semi-automated manner. Due to the space constraint, a small case study is presented as a proof of concept. The case study, we have chosen is a part of the Credit Card System described in [4]. Here is short summary of such system:

> "We consider an information system for a bank's credit card operation. A body of information on cardholders and merchants is maintained. In this highly competitive market, it is important to provide fast response time and accuracy for sales authorization. To reduce losses due to fraud, lost

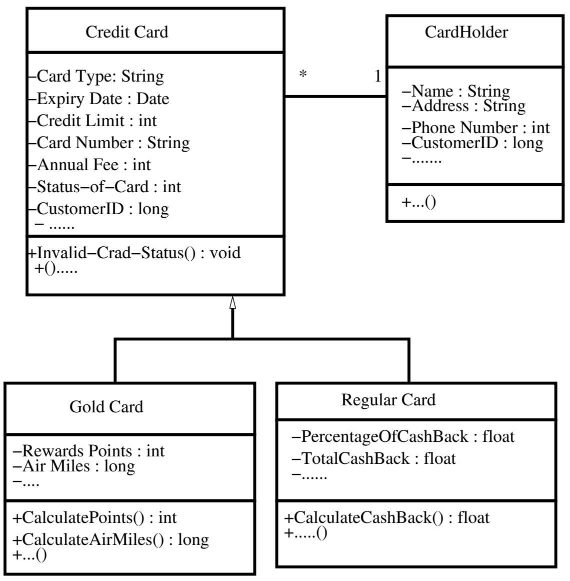
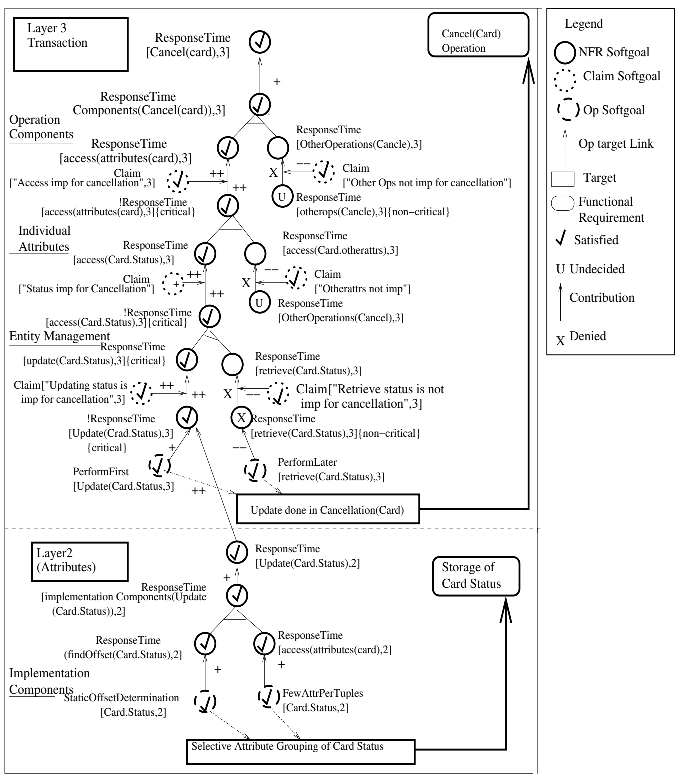


Figure 2. A Class Diagram of Credit Card System



Layer 3 Transaction
ResponseTime [Cancel(card),3]
Cancel(Card) Operation
Legend
NFR Softgoal
Claim Softgoal
Op Softgoal
Op target Link
Target
Functional Requirement
Satisfied
U Undecided
Contribution
X Denied
ResponseTime Components(Cancel(card)),3]
Operation Components
ResponseTime [access(attributes(card),3]
ResponseTime [OtherOperations(Cancle),3]
Claim ["Access imp for cancellation",3]
Claim ["Other Ops not imp for cancellation"]
!ResponseTime [access(attributes(card),3]{critical}
ResponseTime [otherops(Cancle),3]{non–critical}
Individual Attributes
ResponseTime [access(Card.Status),3]
ResponseTime [access(Card.otherattrs),3]
Claim ["Status imp for Cancellation"]
Claim ["Otherattrs not imp"]
!ResponseTime [access(Card.Status),3]{critical}
ResponseTime [OtherOperations(Cancel),3]
Entity Management
ResponseTime [update(Card.Status),3]{critical}
ResponseTime [retrieve(Card.Status),3]
Claim["Updating status is imp for cancellation",3]
Claim["Retrieve status is not imp for cancellation",3]
!ResponseTime [Update(Crad.Status),3] {critical}
ResponseTime [retrieve(Card.Status),3]{non–critical}
PerformFirst [Update(Card.Status,3]
PerformLater [retrieve(Card.Status),3]
Update done in Cancellation(Card)
Layer2 (Attributes)
ResponseTime [Update(Card.Status),2]
Storage of Card Status
ResponseTime [implementation Components(Update (Card.Status)),2]
ResponseTime (findOffset(Card.Status),2]
ResponseTime [access(attributes(card),2]
Implementation Components
StaticOffsetDetermination [Card.Status,2]
FewAttrPerTuples [Card.Status,2]
Selective Attribute Grouping of Card Status

Figure 3. A SIG Performance for Credit Card System

and stolen cards must be invalidated as soon as the bank is notified."

The following sections elaborate further how each step of our proposed framework can be applied to the case study.

4.1 Identification of FRs and NFRs

In the selected case study, the functionality of Credit Card System includes maintaining information on *sales*, *card holders* and *merchants*. Transactions are authorized, and accounts are updated. Stolen cards are cancelled. The non-functional requirements of the credit card system may be performance and security of the transaction. Performance can also be divided as performance of card authorization and performance of card cancellation and so on. Here we consider the "Fast Response Time" to invalidate a card status (when it is lost or stolen) as the target NFR for the selected case study.

4.2 Specification of FRs and NFRs

Figure 2 shows a part of the class diagram of a Credit Card System specifying its FRs.

Figure 3 shows a partial SIG specifying the evaluation of impact of decisions after selecting operationalizing softgoals with respect to card cancellation operation. As shown in Figure 3, the "Performance" quality attribute of Credit Card System can be achieved by selecting attribute grouping of card status. One possible solution for this may be to physically store the card status separately along with few other attributes (card number for example). By applying such a mechanism, when a request comes for the cancellation of a particular card, the status of the card can be retrieved very quickly without any need to access irrelevant information (such as customer information, bonus points calculation) and the status can be updated to "invalid card" very easily. In the context of UML class diagram, one possible way to express this NFR is to create a new class (say "FastResponseTime") for the method Invalid-Card-Status() along with the attributes "Status-of-card" and the "Card-Number" (as primary key). This modification can result in deleting the attribute "Status-of-card" and the method "Invalid-Card-Status" from the original "CardHolder" class.

4.2.1 Buidling NFR Template and Repository

Figure 4 shows a possible set of NFR commands, from our proposed instruction set for class diagram, to specify the NFR "FastResponseTime". To create a template for the above NFR based on the solution discussed in the previous section, the following steps are required:

1. Accepting the join-point of the NFR "Fast Response Time" (here, method Invalid-Card-Status()).

```
 Steps   Commands

 0:       Input Parameters: Method(Single)<param0>
 1:       AttributeList<result0> := getAttributesFromParentClass(<param0>)
 2:       AttributeList<result1> := getSelectedInput("Attributes to Move",<result0>)
 3:       AttributeList<result2> := getSelectedInput("Primary Keys",<result0>)
 4:       ClassNode<result3> := createNewClass("FastResponseTime"
 5:       <result3> := insertMultipleAttribute(<result3>,<result2>)
 6:       <result3> := insertMultipleAttribute(<result3>,<result1>)
 7:       <result3> := insertMethod(<param0>)
 8:       deleteMethodFromParentClass(<param0>)
 9:       deleteAttributesFromParentClass(<result1>)
10:       ClassNode<result4> := getParentClass(<param0>)
11:       createAssociation(<result3>,<result4>,1,1)
Output:   (Step 7(result),  Step 8(Operation), Step 9(Operation), Step 11(Operation)
```

Figure 4. NFR Template Commands

2. Selecting other attributes to move to the separate class for card status (here attributes Card-Number and Status-of-card).

3. Creating a new class (say, FastResponseTime) and building an association with the parent class.

4. Inserting those attributes defined in Step 2 and the method defined in Step 1, into this new class.

5. Deleting the attribute Status-of-card from the original class.

```
public class FastResponseTime{
      public FastResponseTime(){
     }

     void constructTargetSystem(){

          expectParameter(Method);
          MethodName param0 = getParameter();
          AttributeList result0 = getAttributesFromParentClass(param0);
          AttributeList result1 = getSelectedInput("Attributes to Move", result0);
          AttributeListresult2 = getSelectedInput("Primary Key", result0);
          ClassElement result3 = createNewClass("FastResponseTime");
          result3 = insertMultipleAttribute(result3,result1);
          result3 = insertMultipleAttribute(result3,result2);
          result3 = insertMethod(result3,param0);
          deleteMethodFromParentClass(param0);
          deleteAttributeFromParentClass(result1);
          ClassElement result4 = getParentClass(param0);
          createAssociation(result3,result4,1,1);
     }
 }
```

Figure 5. Java Class of NFR Template

In our prototype, we use the advantage of *dynamic class loading* feature of Java. We have built an NFR interpreter which interprets the commands in NFR template and generates the corresponding Java source code for the template. Each template is stored as a Java class in the NFR library/repository. The desirable template would be translated into the following piece of Java code as shown in Figure 5.

4.3 Integration of NFRs

The last step of our framework is to incorporate this NFR template with target design in UML notation. In order to attach the NFR with a method (e.g. Invalidate-Status-of-Card()in the design), the designer needs to supply the corresponding method as a parameter to the template. During the weaving, he/she needs to provide the necessary dynamic parameters to complete the process.

Figure 6 shows the new class diagram after the integration of NFR “FastResponseTime”. After the integration of NFR “FastResponseTime” with the class diagram of Credit Card System, a new class named “FastResponseTime” is created with the attributes “Status-of-Card”, “Card-Number” and the attribute “Status-of-Card” is deleted from the original class “CardHolder”.

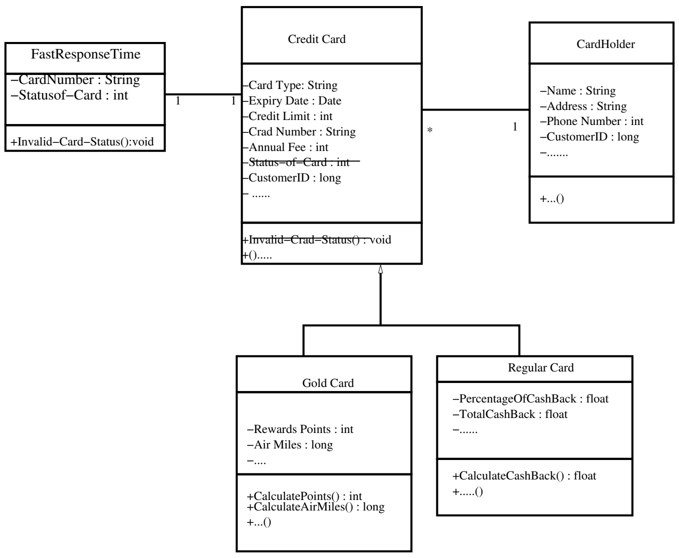


Figure 6. Adding NFR FastResponseTime

5. Conclusion and Future Work

We propose a novel framework for integrating non-functional-requirements with the UML design of a software system which can be applied during the re-engineering process of such a legacy system. The framework can also be used during forward engineering if the developers follow the standard XMI during their model design. Currently, we have built a prototype of the whole framework where the weaver supports the NFR design with class diagrams. The prototype also provides the advantage to draw the Softgoal-Interdependency graph (SIG) and to store the NFR template in the NFR library/repository along with the NFR information that comes from the NFR framework. We are now working on extending the prototype to incorporate all types of UML diagrams.

References

[1] Object management group. Unified Modeling Language Specification version 1.3 bata 1, 1999. available at uml.shl.com.

[2] J. Bergey, M. Barbacci, and W. William. Using quality attribute workshops to evaluate architectural design approaches in a major system acquisition: A case study. Technical Report CMU/SEI-2000-TN-010, Software Engineering Institute, Carnegie Mellon University, 2001.

[3] B. W. Boehm et al. *Characteristics of Software Quality*. Elsevier North-Holland Publishing Company, Inc., 1980.

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

[5] L. M. Cysneiros and J. C. S. do Prado Leite. Nonfunctional requirements: From elicitation to conceptual models. *IEEE Trans. Softw. Eng.*, 30(5):328–350, 2004.

[6] E. Dimitrov and A. Schmietendorf. Uml-based performance engineering possibilities and techniques. *IEEE Software*, 19:74–83, January/February 2002.

[7] W. Hürsch and C. V. Lopes. Separation of concerns. Technical Report NU-CCS-95-03, Northeastern University, February 1995.

[8] International organization for standardization (iso). Technical report. Information Technology, Software Product Evaluation, Quality Characteristics and Guidelines for Their Use, ISO/IEC 9126, 1996.

[9] J.-M. J*éz*é*quel, N. Plouzeau, T. Weis, and K. Geihs. From contracts to aspects in uml designs. In *Aspect-Oriented Modeling with UML, AOSD Workshop, Enschede, Netherlands*, April 2002.

[10] A. Moreira, I. Brito, and J. Arajo. Crosscutting quality attributes for requirements engineering. In *The fourteenth International Conference on Software Engineering and Knowledge Engineering (SEKE’02)*, pages 167–174, July 2002.

[11] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: a process-oriented approach. In *IEEE Transactions on Software Engineering*, volume 8, pages 483–497, June 1992.

[12] S. Supakkul and L. Chung. Integrating frs and nfrs: A use case and goal driven approach. In *2nd International Conference on Software Engineering Research, Management and Applications (SERA’04),Los Angeles, CA*, pages 30–37, May 2004.

[13] L. Tahvildari and K. Kontogiannis. Improving design quality using meta-pattern transformations: a metric-based approach. *Journal of Software Maintenance*, 16(4-5):331–361, 2004.

[14] L. Tahvildari, K. Kontogiannis, and J. Mylopoulos. Quality-driven software re-engineering. *Journal of Systems and Software (JSS)*, 66(3):225–239, 2003.