IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4283

An Approach to Deal with Non-Functional Requirements within UML

Guadalupe Salazar-Zárate and Pere Botella
Department of Llenguatges i Sistemes Informàtics (LSI), Technical University of Catalunya (UPC)
Tel: 34-3-4015641, Fax: 34-3-4017014, {gsalazar, botella}@lsi.upc.es

Ajantha Dahanayake
Information Systems, Delft University of Technology, The Netherlands
Tel: 31 (15) 278-5811, Fax: 31 (15) 278-6632, a.n.w.dahanayake@its.tudelft.nl

INTRODUCTION

This paper presents ongoing work on formulating a methodology to deal with the non-functional aspects of software systems. It is well-known that functional and non-functional requirements are both relevant to software development. It is intended that this methodology could be applied and integrated into the Unified Modeling Language (UML) [Booch98]. Nowadays, UML is a standard modeling language for software systems development [Object00].

Our proposal is to extend the UML language in order to include the elements for conceptual modeling and for visualizing the non-functional information in a similar manner as its counterparts, the functional information models.

The specific objectives of this research are focused on the possibility of developing mechanisms to incorporate non-functional elements in the development of software systems and particularly in the extension of the UML language for integrating non-functional aspects.

The remainder of the paper is organized as follows: section 2 defines the concepts regarding non-functional aspects covered in the research, putting special attention on the standard that defines quality characteristics of software.

Section 3 gives a glimpse on the possibilities for an extension of the UML language in order to include non-functionality and presents the methodology for dealing with non-functional requirements. Section 4 outlines some conclusions and future research.

NON-FUNCTIONAL ASPECTS

Functional and non-functional aspects regarding the external system behavior involve two different ways of evaluating and/or developing a given software system. On one hand, functional aspects are directly connected to *what the system does* i.e. the basic functions that a system (or a system component) must provide. On the other hand, non-functional aspects are related with *how the system behaves* with respect to some observable attributes such as performance, reliability, efficiency, reusability, portability, maintainability, etc. (i.e. some software qualities).

Nowadays, due to the complexity of current software systems, non-functional requirements play a more increasingly important role in the development process. Among the works dealing with non-functionality and without a doubt, one of the most complete is [Chung00]. This approach presents a Framework that enables developers to deal with diverse non-functional requirements of a system in a systematically way, where the requirements are used to drive design by justifying decisions and determining their impact throughout the development process.

Cysneiros et al. [Cysneiros01] introduce an approach also concerned with non-functional requirements that complements the work reported in [Chung00]. They present a strategy to drive elicited non-functional requirements in use cases and scenarios.

To design a software system, we start by defining a set of functional requirements and a set of nonfunctional requirements that describe what behaviors and what characteristics the implemented system must exhibit.

Usually, natural language text is used in non-functional requierements, because its flexibility and adaptability. However, drawbacks, such as ambiguity, inconsistency, and contradictions make this kind of information more difficult to analyze. It clearly results that there is a need to deal comprehensively with such requirements in order to improve the development process and to avoid errors that lead often to greater expenses in software products.

In this project the ISO 9126 standard [ISO/IEC91] will be used, as a starting point to identify non-functional attributes of products that are potentially of relevance to be modeled in a software development process. In a later phase more non-functional attributes will be included. The standard is primarily concerned with the definition of quality characteristics, which are expressed in terms of some high-level features of the software, such as efficiency, reliability and others. The standard collects quality needs with the main idea of defining a *quality model* and using it as a framework for software evaluation. A *quality model* is defined by means of general *characteristics of software*, which are further refined into *subcharacteristics* in a multilevel hierarchy. Measurable *software attributes* appear at the bottom of the hierarchy.

The ISO 9126 standard defines that the characteristics of functionality, reliability, usability, efficiency, maintainability and portability are placed at the top of the hierarchy. An informative annex of this standard provides an illustrative quality model that refines these characteristics.

Software Quality Metrics allows the quantification of the degree to which a software system meets non-functional requirements. Metrics, in the ISO 9126, typically give rise to quantifiable measures mapped on to scales. The rating levels definition determines what ranges of values on those scales count as satisfactory or unsatisfactory. Since quality refers to given needs, which vary from one evaluation to another, then no general levels for rating are possible: they must be defined for each specific evaluation. To determine objectively whether or not a given software product is satisfactory, one needs to define precisely the specific attributes (basic attributes) and not to express them in general qualitative terms, such as reliability, maintainability, portability, efficiency, etc.

THE PROPOSED METHODOLOGY AND UML

UML is a general-purpose visual modeling language for specifying, visualizing, constructing, and documenting the artifacts of software systems [Booch98]. However, UML is mainly focused on functional aspects of the software development. UML offers a drawing notation with semantics to create models (diagrammatic representations of programs)[Rational99]. We would like to have the same visual possibilities to cope with the non-functional aspects of the software systems.

Currently we are exploring how the quality characteristicas, under the ISO 9126 standard, can be transformed into something that can be modeled by using the UML language. These visual elements that represent the non-functional information must have their own seman-

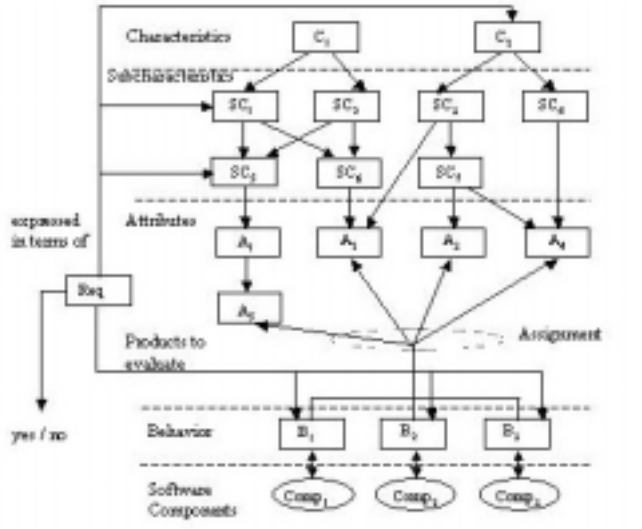
This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

tics and notation. In order to accomplish this, also the new visual elements must be formally defined at metamodel level.

A structure is needed to identify those kinds of elements that can be used to visualize non-functionality in UML diagrams. Therefore, a graphical template containing those quality aspects covered by the ISO 9126 standard must be developed.

One way of achieving this is through the use of a layout developed under "NoFun" [Botella01], which is a formal language for an exhaustive description of the software quality characteristics contemplated in the ISO 9126 standard. In the layout the software quality charateristics of a quality model are broken down into subcharacteristics, as shown in figure 1, until some basic attributes are placed at the bottom of the hierarchy (see [Botella01] for more details) —where *quality requirements* may be defined as restrictions over the quality model.

Figure 1: Layout of a quality model under ISO/IEC framework



Currently, a detailed study is being carried out to determine the possible ways of representing those basic attributes in the context of UML.

The following considerations have been done for achieving the appropriate UML extensions in order to include the non-functional elements.

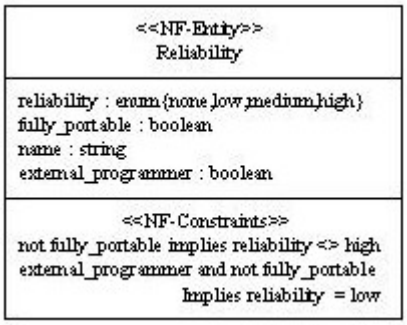
Firstly, we propose to make use of stereotypes to represent the non-functionality. The UML concept of stereotype is the extensibility mechanism that UML offers to extend its modeling vocabulary that allows us to create new kinds of building blocks that can be adapted to our specific problem. Secondly, those software quality characteristics and attributes defined in the layout (see figure 1) can be identified, and some ad hoc stereotypes can be also defined. Besides, the models can be refined by applying different abstraction levels during ulterior stages of the development process.

The UML concept of stereotype can be used to include non-functionality in several ways:

- *In a Class diagram*. At conceptual level, one can create sterotyped classes <<NF-Entity>> to represent a specific quality characteristic. New strereotypes can be formulated and further refined into more specialised ones. In a more detailed level, the Object Constraint Language (OCL) [Warmer99] or an adaptation of the NoFun language can be used to stablish requirements in a form of constraints. An example of this possibility has been studied in [Salazar00].
- *In UML package diagrams*. One can group software elements under a stereotyped package, for example to represent a collection of non-functional stereotyped classes. In this way, a framework for further design specifications is provided that includes the required non-functional attributes.
- *In Use Cases*. The non-functionality can be seen as transformation rules causing that new use cases may be added and/or the structure of the use case description may be changed (or replaced), so that the non-functional requirements are satisfied. Also new actors may be identified in the transformation process. By adding stereotypes for example <<NF-uses>> or <<NF-extends>>, we can identify the uses cases that involve non-functionalities. Constrains can be attached in a special script where the specification can be written using OCL or NoFun.

Among the different proposed usages of stereotypes within the UML diagrams, up to now Class diagrams have only been used to show the possible extensions for non-functionality, as reported in [Salazar00]. Figure 2 shows a stereotyped class with some basic non-functional attributes as an example to define the Reliability degree for a specific software component. In order to establish some constraints, an extra compartment containing OCL expressions is added within the stereotyped class.

Figure 2: An example of a non-functional stereotyped class



In [Cysneiros00] a strategy to deal with the non-functional requirements is introduced. A detailed integration process of the non-functional requirements into the conceptual models is described. Although we do not have experience on the use of the strategy, it seems to be complex and demands the use of specific (self-developed) tools. With our approach we will try to reduce the complexity by using and extending the UML notation.

It is intended to identify the non-functional elements in order to extend the UML vocabulary, first at model level, and in a later step, at metamodel level, to provide a more general extension to the UML vocabulary.

CONCLUSIONS AND FUTURE WORK

Using UML stereotypes as a mechanism for incorporating non-functional elements has been introduced. The possibility of developing other mechanisms to incorporate non-functional elements that can be added as new elements to the UML visual language is still in progress. Currently, we are exploring the possibility to achieve this through the creation of a framework where basic non-functional attributes can be defined, selected, and presented in terms of elements that can be modeled within UML diagrams.

The diagrams (classes, use cases, and packages diagrams) considered in this paper model static aspects of software systems. UML provides also mechanisms for modeling the system dynamic aspects (state, sequence, and activity diagrams) that could possibly incorporate non-functional information. However this has not been done yet, but a detailed study should be done in the future.

ENDNOTES

1 M.G. Salazar-Zárate is holder of a scholarship from CONACyT/ México under contract 122464.

REFERENCES

[Booch98] Booch, G., Jacobson, I. and Rumbaugh, J., (1998); *The Unified Modeling Language Users Guide*; (Addison-Wesley object technology series) Addison Wesley.

[Botella01] Botella, P., Burgués, X., Franch, X., Huerta, M., and Salazar, G.; *Modeling Non-Functional Requirements*. Jornadas de Ingeniería de Requisitos Aplicada. Sevilla, 11 y 12 de Junio 2001. Spain.

[Chung00] Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J.; *Non-Functional Requirements in Software Engineering*; Kluwer Academic Publishers. 2000.

[Cysneiros00] Cysneiros, L.M. and Leite, J.C.S.P. "Using UML to Reflect Non-Functional Requirements". Requirements Engineering Journal Vol.6 No.2 2000 (http://rej.co.umist.ac.uk).

[Cysneiros01] Cysneiros, L.M. and Leite, J.C.S.P. "Driving Non-Functional Requirements to use Cases and Scenarios" XV Simposio Brasileiro de Engenharia de Software, oct 2001 pp:7-20. Brazil.

[ISO/IEC91] ISO/IEC 9126; *Information Technology —Software Product Evaluation—Quality Characteristics and Guidelines for Their Use*. International Organization for Standardization. Geneva. 1991.

[Object00] Object Management Group. Unified Modeling Language Specification, Version 1.3. Technical Report, Object Management Group, 2000. Available at http://www.omg.org/technology/documents/formal/unified_modeling_language.htm

[Salazar00] Salazar-Zárate, G. and Botella, P.; *Use of UML for non-functionals aspects*; 13th International Conference Software & Systems Engineering and their Applications. (ICSSEA '2000). Paris, France. 2000.

[Rational99] Rational Software Corporation: *UML Semantics*, version 1.3. June 1999. Available at http://www.rational.com/uml/resources/documentation/index.jsp

[Warmer99] J. B. Warmer, and A. G. Kleppe. "The Object Constraint Language: Precise Modeling With Uml". (Addison-Wesley Object Technology Series) Adisson Wesley 1999.