I.J. Information Technology and Computer Science, **2012, 9, 1-8**
Published Online August 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2012.09.01

Integrating Non-Functional Properties in Model Driven Development: A Stepwise Refinement View

Maryam Nooraei Abade
Department of computer engineering, Science and Research Branch, Islamic Azad University Tehran, Iran
S.Nooraei@gmail.com

Zeinab Rajabi
University of applied science and technology Tehran, Iran
rajabi.Ze@gmail.com

Abstract— Most of the refinement approach is about functional property of systems. Non-functional properties are as important as functional one. Without an accurate approach for specifying and refining their behaviors, software models will be regarded as imperfect and imprecise, and as a result, software systems cannot be generated correctly. Therefore, how to model such behaviors and how to stepwise refine these behaviors automatically, have become two critical problems in Model Driven Development. In this paper we present an approach for Non-functional refinement in model driven development using high order transformation languages and traceability property of them. We extend the idea of model refinement to non-functional properties of software and propose a stepwise refinement framework with conformance checking between abstract and concrete descriptions of system model using model transformation. The approach is extendable to all quantitative and quantitative non-functional properties**.**

Index Terms—Model Driven Development, Non-Functional Property, Refinement, Platform Independent Model, P Specific Model

I. Introduction

There are several aspects to reach the desirable level of maturity in software development. Although the impact of non-functional properties (NFP) over software systems has been widely mentioned, there is still a lack of approaches that integrate this type of requirements into the development process to produce cost-effective software. In this paper we extend the notion of model refinement to non-functional properties of software and propose a model driven based refinement framework that has the advantage of a consistent interpretation of the extra-functional properties found on different abstraction level regardless of its usage context and that separates the semantics and the syntax of each property between abstract and concrete descriptions of components during MDD [1]. It suggests distinctive models at the different levels of abstraction and different phases of development during system development. The process of development will consist of the transformations and refinement of models. Transformations serve as a mean of progressive refinement of models from abstract, platform independent, requirement centric towards concrete, platform specific, implementation centric.

During the design of MDE, it is useful to model non-functional properties of a system, like performance, already in early stages of the development process. Developers often see quality of service as a property of software that is checked and corrected once the product is completed. This “fix-it-later” practice is, however, a reason for quality problems in software development. Just like testing is an integral part of the implementation process that should be integrated from the beginning, early NFR modeling enables the developer of a system to make require design decision based on analyses and simulations [2].

This paper is structured as follows: In Section 2, we give a brief introduction into the background of MDE. In Section 3, related work is discussed. The basis of property-driven software engineering is discussed in Section 4. The framework for non-functional model-driven software certification and refinement are presented in section 5 before the paper concludes with Section 6.

II. Background

MDA is a development approach based on UML models, in which business knowledge (Platform Independent Models - PIMs) is maintained separately from technical artefacts, such as design models (Platform Specific Models PSMs) and source code. The successful application of the MDA approach depends on technologies and tools supporting flexible modeling of diverse semantic domains (PIMs and PSMs), and relationships and transformations between them

(deployment of PIMs to PSMs). We use the UML profile mechanism to define classes of analysis models, design models and the mapping between the two. Profiles are denoted using UML and may be injected into any conforming tool, reducing tool tie-in [3].

Refinement is a technique used to transform the abstract model of a software system into a more concrete one. Without an accurate approach for specifying and refining their behaviors, software models will be regarded as incomplete and imprecise, and as a result, software systems cannot be generated automatically [4]. Therefore, how to modeling such behaviors and how to stepwise refine these behavior models automatically, have become two critical problems in MDD.

We find that there is a further exploration worth entering into on the matters of:

- Refining a diversity of non-functional requirements (NFR), using term rewriting augmented by pre/post conditions;
- Refining the functional requirements in concert with the non-functional requirements;
- An applicable, formal and rigorous refinement method which works efficiently, effectively and correctly;
- Application and technology domain restriction.

In particular for safety critical systems it is necessary to make sure that the non-functional properties imposed by a system architecture meet the related requirements as early as possible. Therefore, appropriate architectural transformations have to be applied in the design phase in case the non-functional properties do not fulfill their requirements. As the selection and application of appropriate architectural transformations is a time consuming task and demands for personal effort, there is the idea to automate the architecture evolution process. In this paper, we outline toward to introduce the architecture evolution process and propose the framework that proves the behavioral equivalence of the architecture before and after implementation using Platform Independent and Platform Specific transformation respectively named PIT and PST.

Abstract NFR models can, however, also be used to express requirements in the specification phase of Model-based software development. As the development proceeds, additional requirement models are created to describe the properties of the design, and eventually the implemented detail. In order to prove that the NFRs are met in all these stages, a notion of refinement for NFR is needed. By using this framework in the development process, the developer can check at any time if the requirements are still met and which properties may be violated.

Even if the commissioned component is delivered without NFR specification, it can be reconstructed by reverse engineering methods such as static code analysis and analyses of execution traces [5]. However, as such a reconstructed NFR description can differ from a manually specified one; the refinement calculus still is needed to show the compliance.

2.1 State-Based Model Transformation

The notion of model transformation is an essential element for MDA aiming at automated model transformation. Transformations may be bi-directional. Four different types of transformation are introduced in MDA: PIM to PIM, PIM to PSM, PSM to PSM and PSM to PIM to [6]. He refines the definition of transformation by classifying transformations into two categories: horizontal and vertical. Figure 1 summarizes the relationships of the different types of transformations. It is hard to define a fixed number of PIMs and PSMs for software systems. For example, models may be written with various modeling languages, e.g. UML and ADLs (Architecture Description Language), resulting in several models at the same level of abstraction and for several platforms (e.g. Java and CORBA), which, leads to several PIMs with low different level of abstraction.

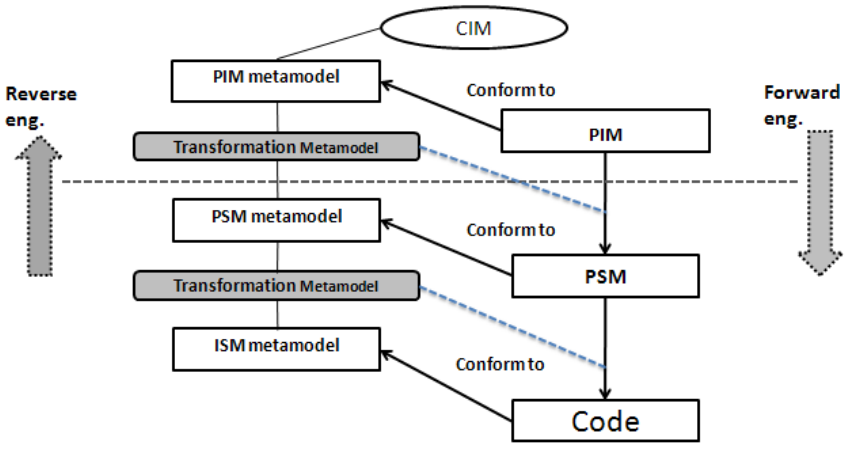

Fig 1: Different levels of abstraction in model driven engineering

Domain-Specific Modeling (DSM) has become an important part of Model-Driven Engineering (MDE) to address the complexity of software systems. A Domain-Specific Modeling can be used to declaratively define a software system with specific domain parameter and specific Non-functional requirements according to the business problem. Various software artifacts (e.g., code, simulation scripts and XML deployment description files) can be generated automatically from the models. By increasing the level of abstraction, DSM leads to hold key rational assets from technology obsolescence, resulting in low-level details to specify a given system, which can lead toward improvements for supporting end-user [7].

III. Related Work

Traditionally, there are two methods to model and refine behaviors, including UML-based methods and formal methods. Recently integrated methods have received increasing attention by taking full advantage of UML and formal methods. Some representatives are reviewed as follows [8]:

3.1 UML-based methods

Based on the UML specifications, codes can be generated automatically through stepwise model refinement, using Model-Driven Architecture (MDA) technology, and this process is measurable [9]. Many researchers have worked on the refinement theory and methodology for UML-based modeling. In [10] a tool to support the refinement of non-functional constraints in UML models has been provided [4]. It formally defines the relationship between the behavior inheritance consistency of a refined model and the behavioral preservation of a refactored model according to the original model.

Several authors have proposed to model NFRs using UML extensions [1] [11], including the OMG standard UML profiles MARTE [12] and QoS-Profile [13]. Others designed a specific meta-model to present NFRs [14] [15].

3.2 Formal methods

Formal specifications can be used to provide a precise supplement to natural language, and can be validated and verified, so leading to the early detection of software specification errors. From early research about proving correctness of programs, such as [16] and [17], a refinement calculus of specifications to codes has developed. Such as the refinement of Z [18] B [19], Event-B [20], ASM [21], etc has been proposed. The refinement calculus, developed independently by Morgan [22] [23] and Morris [24], provides a uniform method for deriving programs from specifications.

Actually automatism is not one of the traditional focuses in the researches of formal modeling and refinement. Besides, there are too many possible directions for each refinement step, and thus it is difficult to achieve efficient automatics for refinement without context and constraints.

3.3 Hybrid ethods

Given a set of NFRs, [14] proposes a set of patterns that satisfy QoS requirements through model transformation.

In [25], the proposed patterns consider architectural aspects. Other examples are [26] and [27]. In these approaches each kind of NFR may be seen as a whole dimension of the software. [28] and [29] propose analyzing each NFR type separately and also to use different abstraction levels for NFRs (at CIM, PIM and PSM levels). As a conclusion, we may say that although several valuable approaches have been proposed that deal with NFRs in the MDD process, none of them propose a stepwise integrated refinement view, which is the goal of this paper.

IV. Non-Functional Property-Driven Software Engineering Approach

The approach we propose in this paper originates from, and contributes to, a broader research framework, which we refer to as a Non-Functional Property-Driven Software Engineering Approach. The roadmap in fact defines an enhanced model-driven software engineering approach where, among the others, models of non-functional properties become first-class entities beside functional properties. The considered non-functional properties describe both characteristics owned by a software system or by parts of it and non-functional requirements the developed software system must satisfy. Figure 2 graphically illustrates the approach by showing the process underlying it. In the figure 2, boxes represent different levels of abstractions, inspired by four layers MOF architecture, bold arrows labeled with meaning ”conform to” represent the conformance relation between the models and the relative meta-model; and, finally, simple arrows show the control flow of the process.

The process in Figure 2 is divided into three parts or integrated field: i) Property Modeling; ii) Improvement-based Software Engineering; iii) Model-Driven Software Engineering.

In the sub-process of property modeling the non-functional properties, include descriptive and prescriptive. Also a non-functional property can be quantitative or qualitative. The former needs to specify the metrics used to quantify it and qualitative properties are related to the concept like security that cannot be measured exactly [30].

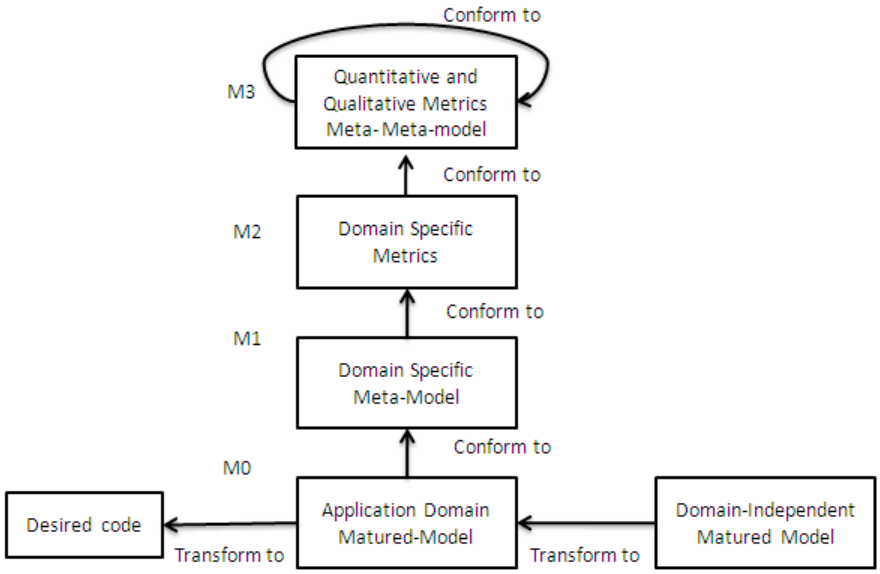

Fig 2: quality-driven model driven engineering

Several attempts have been made to define common non-functional property definition formalism, but the most promising is the upcoming UML MARTE profile. The NFP package of the profile contains a generic meta-model for the definition of NFPs; including qualitative and quantitative ones (Figure 3). The Non-Functional Property has three optional attributes that NFP type, NFP Value Specification and Unit. These properties can be instantiated and attached to any model element of the system model in order to express new aspects of non-functional characteristics.

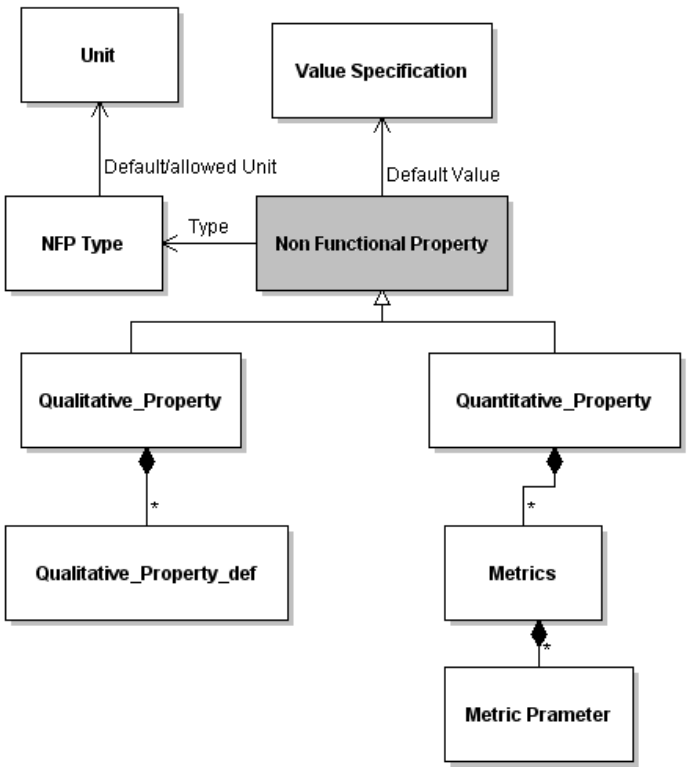

Fig 3: A customized non-functional meta-model

There are two sides to develop systems with non-functional properties: Component developers must implement components in such a way that they have determinable non-functional properties and application designers and the runtime system must use these components so that the non-functional properties required from the application can be guaranteed.

We are not primarily interested in how components must be implemented. Instead, we assume a model-driven development approach with determinable and first entity non-functional properties to be available. Based on this, we provide a framework, which allows

- Component developers to describe the non-functional properties of the components they have developed, and
- Application designers to describe how these components are used to provide guaranteed non-functional properties of an application.

4.1 Refinement of NFR using model transformation

Figure 5 illustrates refinement as a process of transforming an abstract model into a specified design using a sequence of design decision. This is a mapping process between a high-level, abstracted representation and its successive specified descriptions. In the computational refinement of the class of designs, inheritance can be employed as a technique to maintain class characteristics. One of choice to trade off between different NFR at architecture level is using ATAM. Architecture Trade-off Analysis Method (ATAM) [31] is a scenario-based software architecture evaluation method. The goals of the method are to evaluate an architecture level design that takes into account multiple quality properties and to gain insight to whether the implementation of the architecture will meet its requirements.

A rule-based formalism provides computational means to achieve refinement. This is illustrated in figure 5 which is a rule-based representation of the refinement.

Transformations should be developed along a cycle ranging from platform-independent transformations down to platform-specific transformations. Platforms here should be understood as the tools that allow the specification, design and execution of transformations. Such tools may provide various languages to express platform-specific transformations such as J, Visual Basic, or XSLT [32]. This is in line with the separation of concerns between PIM and PSM in the MDA, and leads to the concepts of platform independent transformation (PIT) and platform specific transformation (PST). Platform independent transformations are models of the transformation program, relying on a generic library of simpler transformations and transformation primitives. They will be refined to the point where they can be used as the source to generate platform specific transformations. Then if UML is the language of PIT, PSTs are models of tool specific formalisms or API. For example, while XSLT is a textual language, it is possible to consider a MOF-compliant meta-model of XSLT; the PST is then an XSLT model which is serialized out of sight to its XML representation.

The transformation template in figure 4 demonstrates that existing transformation language rules convert one pre-pattern to one post-pattern. However, an extension is required to support selection between consequent patterns based on an input NFP policy specification. This also shows that the transformation can be defined concisely and largely in a declarative manner, with a complex formal specification of component and connector behavior contained separately in the different ADL repository. Source and target of a transformation rule may be any two levels of the refinement schema in figure 4.

The tracing of transformation between different level results back to a *Trace link(S,T)* relation. These are applications of tracing technique in MDE. This has to be taken into account when transforming a PIM_{NF} into a PSM_{NF}.

Transformation rules

```
{
    Source Pattern
    //Match precondition in source model
    {
        Precondition
    }
    Target Pattern
    //Match postcondition target model
    {
        Postcondition
    }
    //Add Traceability Link between Source and Target elements
    Trace link( Source , Target)
}
```

Fig 4: Customizable model to model transformation

V. Non-Functional Requirement Model-Driven Development

The proposed framework to integrate non-functional property in model driven development has been shown in the figure 5. It has four levels of abstraction:

PIM_{NF}: Non-Functional Platform Independent Model is a representation of the business logics of the system along with estimates of non-functional characteristics, such as the amount of resources that the logic needs to be executed. The model must be expressed through a notation that allows the specific type of analysis) e.g., Queuing Networks for performance analysis, or Bayesian Belief Networks for reliability analysis (In some cases, it is not possible to obtain numerical values for platform independent non-functional analysis. This is due to the relation between typical non-functional metrics (such as response time) and platform characteristics (such as network latency) that are not available in a PIM_{NF}. Therefore, within the

PIM_{NF} class we intend to represent models that permit to estimate certain metrics with an acceptable degree of approximation. Relying on the measured metrics and on the data available about the model parameters, a PIM_{NF} evaluation can represent a good support to early development decisions.

Domain experts can specify a system's usage in terms of workload, user behavior and parameters (i.e., abstract characterizations of the parameter instances users utilize). Software architects may also construct usage models from requirements documents.

This pure abstract behavior model is at the top of its refinement process, which cannot be the refinement model of any other behavior models; an implementation model is at the bottom of its refinement process, which cannot be refined by any other behavior models.

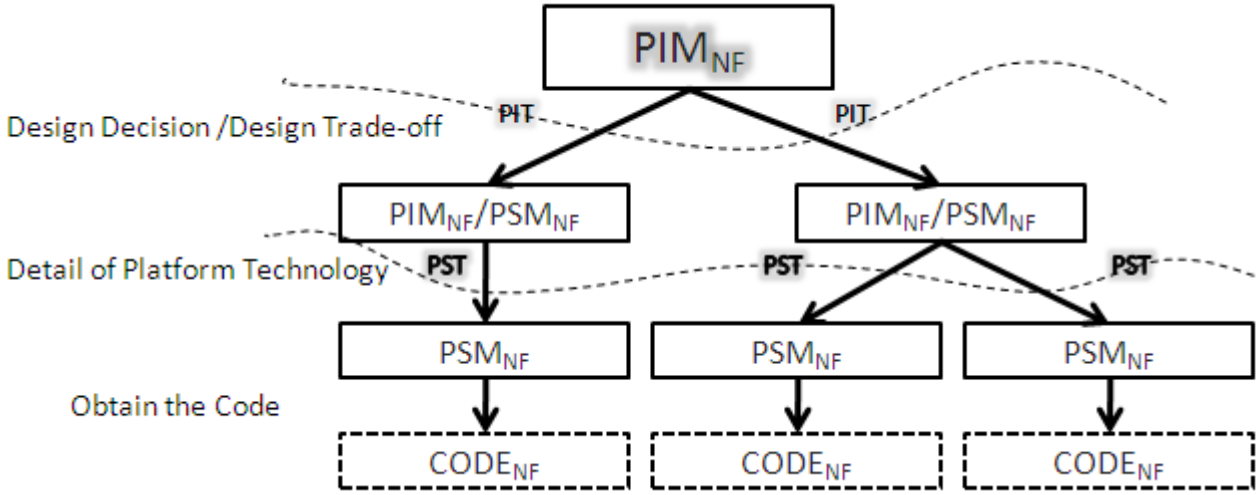

Fig 5: The approach for non-functional model-driven architecture framework

PIM_{NF}/PSM_{NF}: The value of this layer is twofold: firstly, it makes easy analysis, communication and understanding of the system design at a higher level than detail of platform and code. Secondly, in the transformation from PSM to code, the PSM can act as a PIM (independent of programming language abstractions), allowing multiple mappings from model to code. This level makes the approach more flexible. Since the Non-functional property implementation of a component is influenced by many factors according to the architectural design decision and trade off between requirements (using methods like ATAM), the refinement framework should take this into account by offering several levels of refinement. The model may be constructed from the description of the behavioral requirement expressed in the architecture models. This model mixes PIM and PSM levels that specifies some functionality satisfying the NFRs.

The PIT takes PIM_{NF} as input and produces PIM_{NF}/PSM_{NF}, stands for those NFRs that concern the architecture.

PSM_{NF}: A Non-Functional Platform Specific Model contains variables and parameters that represent the software structure and dynamics, as well as the platform where the software will be deployed. In a classical MDA approach a platform is represented by a set of subsystems and technologies that provide a coherent set of functionalities through interfaces and clear usage patterns (e.g. J2EE, CORBA, etc.). In a non-functional context a platform must also include the characteristics of the underlying hardware architecture, such as the CPU speed and the failure probability of a hardware connection. The results of the analysis of a **PSM_{NF}** can be used as a target for comparison to the actual system metrics. This model can then be used to explore the system behavior in the real world (e.g. extremely heavy workloads).

As an example of the approach, consider a performance analysis tool like AnyLogic developed by XJ Technologies. This technology is a leading provider of dynamic simulation tools, technologies and consulting services for business applications using multi-method simulation tool, and allows combining different methods in one model. The object-oriented model design paradigm supported by AnyLogic provides modular and incremental construction of large models. The simulation engine is based on Java technology (platform specific), which makes it possible to use the functionality provided by the Java runtime library in simulation models.

One of the shortcomings of current approaches in this area is the lack of formalisms that provide foundations for automated architecture synthesis. This certainly sounds like an interesting direction to explore. Transformations have the potential for encoding development knowledge, so this may be useful. There is already some work out there that uses transformations for ensuring non-functional properties to explore this literature.

VI. Conclusion And Future Work

Non-functional properties of software should be specified early in the development process. In a distributed process of software development, this means that non-functional requirements must be taken into account in the specification, and the developing party of a component needs to deliver the implemented component with the precise description of its non-functional properties and a conformance checking that guarantees the satisfying of implemented component requirements. Round-trip mapping between architecture design decision and implementation is also an interesting area of research (to pass from PIM_{NF} to PIM_{NF}/PSM_{NF}), which aims at keeping architecture and implementation in sync.

The proposed development process brings together two techniques that are used to ensure the quality of model-driven software: Non-functional engineering and software refinement. The novelty of this approach is the integrated refinement view of non-functional properties based on the standard description of MDA. Using sophisticated NFR language descriptions like the RDSEFF formalism of the Palladio Component Model [33] or $CQML^+$, developers can make NF predictions at early stages of the development process, but also checking if the Non-functional requirements are met by the final product or not, need suitable measures.

The future work will focus on using different formal methods like finite automata and the resource demand calculus, which make it possible to prove valid NFP refinement on different levels of abstraction.

References

[1] Wada, H., Suzuki, J., and Oba, K. A Model-Driven Development Framework for Non-functional Aspects in Service Oriented Architecture. Int. J. of Web Services research, 2008, 5(4):1-31.

[2] Zhu, L., and Liu, Y., Model Driven Development with Non-Functional Aspects. EA @ ICSE, 2009.

[3] OMG. UML 1.4 Specification, OMG Document formal/04-07-02, 2002.

[4] Straeten, R.V.D., Jonckers, V., and Mens, T. A formal approach to model refactoring and model refinement. Software and System Modeling, 2007, 6 (2) 139–162.

[5] Kugele, S., Haberl, W., Tautschnig, M., and Wechs. M. ,Optimizing Automatic Deployment Using Non-functional Requirement Annotation". ISoLA, 2008.

[6] Ramljak, D., Puksec, J., Huljenic, D., Koncar, M. and Simic, D. ,Building enterprise information system using model driven architecture on J2EE platform. In: Proceedings of the 7th international conference on telecommunications, ConTEL, 2003.

[7] Thomas, D. , MDA: Revenge of the modelers or UML utopia? IEEE Software, 2004, 21 (3) 15–17.

[8] Back, R.J., Correctness Preserving Program Refinements: Proof Theory and Applications, Mathematical Center Tracts 131, Mathematical Centre, Amsterdam, The Netherlands, 1980.

[9] Röttger, S., and Zschaler, S. Model-Driven Development for Nonfunctional Properties: Refinement through Model Transformation. In: Proc. <<UML>> Conf. , 2004.

[10] Röttger, S., Zschaler, S. ,Tool support for refinement of non-functional specification. Software and System Modeling, 2007, 6 (2): 185–204.

[11] Fatwanto, A., and Boughton, C., Analysis, Specification and Modeling of Non-Functional Requirements for Translative Model-Driven Development. ICCIS, 2008.

[12] The OMG. UML Profile for MARTE, Beta 2, 2008.

[13] Gallotti, S., Ghezzi, C., Mirandola, R., and Tamburrelli, G.2008. Quality Prediction of Service Compositions through Probabilistic Model Checking. QoSA, 2008.

[14] Solberg, A., Oldevik, J., and Aagedal, J., A Framework for QoS-aware Model Transformation using a Pattern-based Approach. DOA, 2004.

[15] Gogolla, M., Büttner, F., and Richters, M. USE: a UML-based specification environment for validating UML and OCL. Science of Computer Programming, 2007, 69 (1–3): 27–34.

[16] Hoare, C.A.R., An axiomatic basis for computer programming, Communications of the ACM 12 1969, (10): 576–583.

[17] Dijkstra, E. ,A Discipline of Programming, Prentice Hall,1976.

[18] Woodcock, J., and Davies, J. Using Z: Specification, Refinement and Proof, Prentice Hall, 1996.

[19] Abrial, J. ,The B-Book – Assigning Programs to Meanings, Cambridge University Press, 1996.

[20] Abrial, J.R., Cansell, D., and Méry, D. Refinement and reachability in Event_B, in: Proceedings of the 2005 Formal Specification and Development in Z and B (ZB), Springer-Verlag, 2005, pp. 222–241.

[21] Borger, E. ,The ASM refinement method. Formal Aspects of Computing, 2003,15 (2): 237–257.

[22] Morgan, C., and Robinson, K., Specification statements and refinement. IBM Journal of Research and Development, 1987, 31 (5): 546–555.

[23] Morgan, C., and Vickers, T. On the Refinement Calculus, Springer-Verlag, 1993.

[24] Morris, J.M., A theoretical basis for stepwise refinement and the programming calculus, Science of Computer Programming, 1987, 9 (3): 287–306.

[25] Sterritt, A., and Cahill, V. Customisable Model Transformations based on Non-functional Requirements. IEEE Congress on Services, 2008.

[26] Ardagna, D., Ghezzi, C., and Mirandola, R. Rethinking the use of Models in Software Architecture. QoSA , 2008.

[27] Rodrigues, G., Rosenblum, D., and Uchitel, S. 2005. Reliability Prediction in Model-driven Development. MoDELS.

[28] Cortellessa, V., Marco, A. Di., and Inverardi, P. Non-Functional Modeling and Validation in Model-Driven Architecture. WICSA , 2007.

[29] Ameller, D., Dealing with Non-Functional Requirements in Model-Driven Development. Requirements Engineering Conference (RE), 18th IEEE International, 2010.

[30] Monperrus, M., equel, J.-M. J ´ ez ´., Baudry, B., Champeau,J., and Hoeltzener, B. ,Model-driven generative development of measurement software. Software and Systems Modeling (SoSyM), 2010.

[31] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, S., The Architecture Tradeoff Analysis Method. Proc. 4th IEEE International Conference on Engineering of Complex Computer Systems, pp. 68-78, 1998.

[32] B ´ezivin, J. From object-composition to model-transformation with the MDA. In Proceedings of TOOLS-USA , 2001.

[33] Becker, S., Koziolek, H., and Reussner, R. The Palladio component model for model-driven performance prediction, Journal of Systems and Software 82 pp. 3–22, 2009.

M.Nooraei is a Ph.D. Candidate in Computer Engineering Department, at Islamic Azad University Science and Research Branch, Tehran, Iran. She is a faculty member in the department of Computer Engineering of Islamic Azad university. Her research interests include model driven development issues, model synchronization, and change propagation.

Z.Rajabi received her BS at Isfahan University of Technology in 2005 and MSc in Information Technology Engineering (eCommerce) at Nooretuba University in 2012.

