Model Driven Development with Non-Functional Aspects

Liming Zhu, Yan Liu

NICTA, Australian Technology Park, Eveleigh, Australia
School of Computer Science and Engineering, University of New South Wales, Australia

Abstract

Model Driven Development (MDD) refers to the systematic use of models as primary engineering artifacts throughout a software development life cycle. In recent years, MDD has been increasingly employed to guide development with a focus on system modeling, code generation from models and white-box analysis of models. However, compositional system analysis regarding early Non-Functional Aspects/Properties (NFP) remains difficult. In this paper, we critically review the state-of-the-art of MDD in the context of non-functional aspects and shed some lights on the following two questions:

1) How to model Non-Functional Aspect/Property (NFP). The focus is to understand the different sub-types of a non-functional aspects and its compositional and emergent nature.
2) How models can be used for analyzing Non-Functional Aspect/Property (NFP). This focuses on the analysis models in the form of reasoning frameworks (both qualitative and quantitative) behind each non-functional aspect.

1. Introduction

Model Driven Development (MDD) refers to the systematic use of models as primary engineering artifacts throughout a software development life cycle. The use of models raises the level of abstraction and introduces separation of concerns through different types/layers of models. Subsequently, it allows more early aspects analysis and achieves adaptability, portability and reusability through these models. MDD also makes software development activities, including evaluation, more systematic and formal.

Although MDD has many useful features for tackling these challenges [1, 2], they have not been methodically exploited in the context of early aspects. The question is how to leverage different types of models for NFPs analysis.

2. Models in MDD

It is very important to recognize that there are different types of models in MDD. They can be generally organized into the following categories:

Domain/Requirements Models

Domain models do not model an IT system itself but rather focus on the environment and the context of the IT system. Requirements models sit on the boundary between the environment/context and the IT system, referring to both worlds. In the past, requirements were often captured in natural languages. Increasingly, requirements (including both functional requirements and Non-Functional Requirements (NFR)) are being modeled, using modeling languages like i* , generic NFR models [3-5] and attribute-specific models (a mixture of requirements and measured values) [6, 7].

Architecture/Design Models

Architecture/design models represent an IT system at a certain level of abstraction. A software architecture, which is usually considered as high-level design models, has been traditionally described in an Architecture Description Language (ADL). There have been many ADLs developed in recent years, each with their expressiveness focused on different “aspects” of software systems and application domains [8]. Many
useful ADL features have recently been either absorbed into revisions of the UML, or specified as lightweight (through UML profiles) or heavyweight (MOF) UML extensions.

For architecture/design models, there is lack of effort on specifying non-functional properties of a system[9]. For example, provided/required interfaces in a UML component diagram do not reflect any provided/required NFPs. Although, certain profiles, such as MARTE, are able to annotate models with NFPs, their association with functional interfaces is not clear. This makes early aspects driven compositional analysis difficult. A Service Level Agreement (SLA) is another way of associating some promised aspects with an interface. But they are often done informally. Standards, such as WS-Policy, are encouraging a more formal association between functional interfaces and NFPs.

Analysis Models

The importance of using analysis models to examine characteristics of a system is under-specified in the official MDD guidelines. A model of a system does not have to correspond to functional components or software artifacts. A model can be a unique view of a system simply for analysis. For example, in performance engineering, we can choose to view a system as a queue-based model which has servers and queues. In modifiability analysis, we can choose to view a system as a dependency graph model which has nodes to represent conceptual or implementation elements and edges to represent dependency relationships among them. Although these analysis models have certain mappings to system components and software artifacts, they are essentially for analysis purposes rather than representing the system.

A full MDD approach treats analysis models just as any other types of models and tightly integrates it throughout the life cycle. Some analysis models are not necessarily quantitative models and cannot be easily represented in a semi-formal textual/graphic modeling notation. However, there is usually a logical reasoning framework behind it.

3. Modelling Non-Functional Aspects

A software architecture design is defined as “the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them”[10]. Most architecture/design models reflect this definition by focusing on expressing functional components and connectors, and relationships among them. The “external properties” (often early aspects) of those elements are usually not systematically expressed. This is largely due to the confusion of different terminologies surrounding NFPs. We observe that there are essentially five types of concepts related to NFPs: Provided Required, Requirements, Measured and Predicted.

Let us use an illustration here. Imagine that there is a component (which itself may be a composition) which has a provided interface and a required interface. As mentioned earlier, these interfaces are usually functional interfaces (e.g. a set of operations). The component will provide the functions on the provided interface if the component gets “satisfactory” access from both the explicit required interface (which is usually provided by another component). This also assumes that the implementation of the component is working well and the other, often implicit, environment conditions are met.

1) Provided (value of) NFPs

The meaning of provided NFPs of a component is similar to the meaning of a provided functional interface, except it captures the non-functional side of it. Provided actually means “promised” here in a contractual sense; the reality might be different. A provided NFP can be associated with a component in different ways: 1) with the provided functional interface as a whole 2) with lower-granularity individual operations 3) with the component as a whole but not particularly related to the functional interface (e.g. modifiability and portability).

There are two assumptions here, which are applicable to all following discussions:

- The aspects under discussion are external visible aspects, just like the external visible interfaces. Some external visible properties are not only early aspects, but also the behavior of the system.
- To be more accurate, provided NFP means the provided value of a NFP. There is a value assignment process involved [3]. We sometimes omit the word “value” to make the expression more concise.

2) Required (value of) NFPs

The meaning of required NFPs is similar to the meaning of a required functional interface, except it captures the non-functional side of it. The meaning of “required” here is different from the meaning of “requirement”. To compose two components, the required NFPs from one component should match the provided NFP of another component.
3) Non-Functional Requirements (NFPs Requirements or NFR)

The NFR for a component is essentially what a component should behave under a set of assumptions. If a requirement is about external visible properties and external assumptions, the “should-behave” part (NFP) could be reflected as provided NFP and the assumptions (NFP) could be reflected as required NFPs.

Currently, how to “formally or semi-formally” reflect a NFR in terms of provide/required NFPs on a component, especially in association with functional external interfaces, is not well understood. More will be discussed in the modeling section below. It needs to be understood that:

- representing/modeling a set of related NFR (e.g. [4]) is different from associating NFRs with components/systems (through interfaces or as a whole).
- Simply stating that a NFR is realized or satisfied by a component without some formal/semi-formal semantics using provided/required NFPs does not give a clear picture and does not help compositional analysis.

4) Measured (value of) Non-Functional Properties

Unlike provided NFPs and requirements (which essentially plans, promised and should-behaves), measured NFPs reflect the reality through empirical studies, either through direct measurement or surrogate metrics. Measured aspects are usually represented as distributions over time, and high/low/median/average values.

If the measured value satisfies the requirements, we can state that a component satisfies a NFP requirement (or NFR) and the NFR is indeed reflected through the provided NFP under the assumption of required NFP.

5) Predicted (value of) NFPs

How can we know if a NFR is satisfied in a particular implementation, other than through empirical observation? Empirical observations are important but we often need to know before the actual implementation is completed. If a component is composed from existing components, the compositional NFPs can be predicted through some compositional analysis models.

When it comes to model all these different aspects of aspects, it is clear that effort has been made to capture all the above meanings in different ways [11]. However, because these fine distinctions are not always recognized, there is often an arbitrary mixture of different meanings in a particular attempt.

The word Non-Functional Requirements (NFR) is often used to represent all the different meanings. The meaning of provided NFPs is similar to NFR in many circumstances except the former is the NFP provided by a particular implementation of the NFR. However, the required NFP (assumptions in a requirement) is often omitted and measurement/predictions are not considered and cannot be captured on the model.

4. Evaluating NFPs

4.1 Reasoning Framework for NFP

To evaluate NFP (through prediction or measurement), we need to select an appropriate reasoning framework from a repertoire of available reasoning frameworks for each non-functional property. A reasoning framework consists of independent parameters, dependent parameters (usually proxy metrics for non-functional properties) and functions (sometimes informal causal relationships) between the independent ones and the dependent ones.

A quantitative reasoning framework usually reflects these parameters and functions through more formal and mathematical analysis models (e.g. queuing networks for performance and directed graphs for modifiability). The independent parameters in an analysis models also are mapped to and affected by stimulus/design decisions, while the resulting values in dependent parameters correspond to predicted value of the NFP under investigation.

A qualitative reasoning framework reflects these parameters and functions through identified stimulus/design decisions (independent parameters), the NFP under investigation (dependant parameters) and design patterns/primitive design tactics (recognized causal relationships between stimulus/design decisions and quality consequences).

Quantitative reasoning framework

Prediction Enabled Component Technology (PECT) [12] is a systematic approach that applies several quantitative reasoning frameworks and application-specific information to predict the quality of a component assembly.


- A usage profile is effectively the value or value distribution of independent parameters.
An encapsulated evaluation model is a component-level NFP analysis model. Different NFPs require different models. These analysis models are not based on compositional algorithms.

A composition algorithm enables hierarchical composition of component-level NFP and produces an analysis model for compositions and systems. This algorithm only produces the compositional model not the final evaluation.

An evaluation algorithm extracts relevant measures of certain quality attributes (e.g., hazard probabilities, failure rates, and response times, which could come from both measurement and other prediction) and solves the compositional analysis model analytically, numerically or with simulations.

The availability of effective composition algorithms and evaluation algorithms for most NFP is still lacking.

**Qualitative reasoning framework**

Many of the NFP evaluation methods use informal and qualitative reasoning frameworks. This lack of formality and quantitativeness could be explained by the lack of detailed information, the maturity of various NFP analysis methods and cost. It is completely valid to use qualitative reasoning frameworks in many circumstances. However, they should also be integrated with MDD to its maximum extent.

Bachman, Bass and Klein [13] proposes the concept of design tactics. Design tactics are primitive and reusable methods for achieving certain NFP goals. They imply an informal reasoning framework but often with basis in certain quantitative reasoning frameworks and best practices. Zhu [14] attempted a similar approach by extracting elements of reasoning frameworks from patterns as shown in Figure 1.

Dependent parameters are extracted from general scenarios and problem descriptions. Independent parameters are extracted from architecture tactics and solution descriptions. This effort essentially codify the empirical understanding of best practices (e.g. patterns and tactics often in textual description) into a more formal reasoning framework with the potential to be integrated with MDD.

![Figure 1. Extracting Reasoning Frameworks from Design Patterns](image-url)

Most architecture evaluation methods, such as ATAM [15-17] and its variants do not prescribe any particular type of reasoning framework. When situation allows, a quantitative reasoning framework could be used. In other circumstances, scenario, quality attributes and architectural approaches are elicited to form an informal qualitative reasoning framework.

**5 Modeling Design Rationale and Design Decisions**

So far, evaluations are encapsulated in analysis models and reasoning frameworks. Although the result (predicted NFP) could be put back to the model, the rationales and the connections with the requirements (NFR) are not explicit on the models. There are two related schools of thoughts for solving this: 1) model design rationales and decisions explicitly in models 2) trace elements of an architecture/design model to NFR models.

**5.1 Modeling Design Rationales and Decisions**

Viewing an architecture directly as a set of NFR-affecting design decisions and treating them as first-class entities has been seen as the logical next-step of software architecture research [18, 19]. This has attracted more fundamental research into design decisions [20, 21]. However, the ability to capture design decisions and their relationship with NFRs in architecture descriptions and views is still lacking. The way of capturing such NFP-related information is in separate documentation models or content.
management systems. Such a disconnection has a number of problems:

- As an architecture evolves, synchronization between architecture views and documentations is costly.
- Switching between architecture views and documentation for architecture comprehension is inefficient.
- Architecture decisions are often cross-cutting. One decision relates to multiple architectural elements. Expressing this outside architecture views is difficult.
- The rigor and friendliness to automated analysis of unstructured textual descriptions is problematic.

5.2 Expressing design decisions as cross-cutting aspects

One challenging issue for modeling design decisions is about expressing cross-cutting concerns on a design surface, as design decisions usually relate to more than one architectural elements spread across different places and different abstraction levels. There is some work [22] focusing on different ways of expressing pattern usage on top of an existing architecture that is applicable to our approach: Venn Diagram-Style Annotation, UML Collaboration Notation and Tagged Annotation.

Our approach follows the tagged annotation approach in spirit but creates a separate entity for design decisions in order to aggregate all related information in one place. We also provide a consistency checker to facilitate analysis across design decision entities and architectural elements.

We can identify some important aspects of a design decision [18]:

**Decision**: A general design decision can be expressed in OCL, textual formats and any other appropriate domain specific dialects.

**Design rules**: These define rules which need to be followed by components within a system. A rule simply describes a particular way of doing something. In the profile, OCL or other dialects can be used to capture design rules. Rules are important since they provide a more flexible way of regulating architecture quality properties than structural prescriptions in certain situations such as Ultra-Large-Scale systems [23].

**Design constraints**: Other than design rules, a design decision may contain constraints which specify what the system may or may not do. Similar to above, OCL will be used as expression syntax for design constraints.

**Participating elements**: Participating elements are architectural elements to which a design decision refers to. It essentially extracts the context part of the decision/decision rules/decision constraints expression in OCL.

**Rationale**: Rationales can be captured descriptively in a separate tag along the relationships to NFRs expressed using UML extension mechanisms.

![Figure 2. Design Decision Meta-model](image)

Figure 2 shows the stereotype DesignDecision along with its attributes as described early. It is not necessary for modelers to fill in all the tag values for all the design decision classes except for the decision one. Additional meta-attributes can be added to this design decision meta-model, in order to suit specific needs from architects.

![Figure 3. Extending “Realization” Association](image)

Figure 3 shows one of the stereotypes for modeling relationships between decision decisions and NFRs. A design decision can support/break/help/hurt NFR. One decision can affect multiple NFRs.

![Figure 4. Modelling NFR](image)
We capture this in an extensible UML profile as shown in Figure 4. The super-class NFR stereotype captures the six elements along with a description. Attribute-specific stereotypes inherit and can be further extended to include attribute specific entities, especially response-measures. For example, modifiability NFR includes response measures such as number of dependents, transit average impact and effort.

6. Conclusion

The challenge in using MDD for NFP/early aspects modeling and evaluation is the ability to utilize different types of models for different purposes and understand the subtle differences among NFP/aspects terminologies. Analysis models, architecture decisions, rationales and non-functional properties/aspects all need to be integrated into a single MDD paradigm.

7. References