Integrating Performance and Reliability Analysis in a Non-Functional MDA Framework*

Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi

Università degli Studi di L'Aquila, Dipartimento di Informatica
{cortelle, adimarco, inverard}@di.univaq.it

Abstract. Integration of non-functional validation in Model-Driven Architecture is still far from being achieved, although it is ever more necessary in the development of modern software systems. In this paper we make a step ahead towards the adoption of such activity as a daily practice for software engineers all along the MDA process. We consider the Non-Functional MDA framework (NFMDA) that, beside the typical MDA model transformations for code generation, embeds new types of model transformations that allow the generation of quantitative models for non-functional analysis. We plug into the framework two methodologies, one for performance analysis and one for reliability assessment, and we illustrate the relationships between non-functional models and software models. For this aim, Computation Independent, Platform Independent and Platform Specific Models are also defined in the non-functional domains taken into consideration, that are performance and reliability.

1 Introduction

The recent evolution of software development activities based on models is rapidly moving the viewpoint of software engineers from a code-centric perspective to a model-centric one. Model-Driven Architecture [11] has substantially contributed to this change of perspective by providing techniques and tools for model creation and management along the software lifecycle. For example, numerous model transformation approaches finalized at the code generation through model refinements have been recently devised [9].

Software modeling is also a well-assessed practice in the non-functional domain. Since decades performance and reliability experts are use to build models for validating software/hardware systems vs non-functional requirements. However, these activities are not stably embedded in the software development process, thus non-functional models creation and management do not follow the same regular refinements applied to the software model.

In order to fill the gap between software development and non-functional validation, in the last few years the research has faced the challenge of automated

* This work has been partially supported by the IST EU project "PLASTIC" (www.ist-plastic.org), and partially supported by the MIUR project "FIRB-PERF" (Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools).

M.B. Dwyer and A. Lopes (Eds.): FASE 2007, LNCS 4422, pp. 57–71, 2007.

generation of quantitative models for non-functional validation from software artifacts[1]. Several methodologies have been introduced that all share the idea of annotating software models with data related to non functional aspects and then translating the annotated model into a model ready to be validated.

None of these methodologies explicitly defines the roles of its models within an MDA context even if the approaches are often presented as *MDA-compliant*. In fact few approaches can be found in literature that aim at embedding non-functional validation activities and models within MDA.

In the reliability domain, UML profiles have been introduced in [13] and [12] to allow software reliability prediction at different levels of the MDA framework. In [13] the authors translate a set of annotated UML 2.0 Sequence Diagrams into an annotated LTS (Label Transition System) that is then interpreted as a Markov model. The solution of the Markov model provides estimations of the software system reliability. In [12], reliability aspects are introduced in a Platform Independent Model through an Abstract Reliability Profile. Then, at the Platform Specific level a new profile is defined that extends the UML Profile for EJB and allows the specification of the reliability support provided by the J2EE platform. Finally transformation rules have been defined to map the source model (PIM) onto the target model (PSM).

In [15] an approach have been defined in MDA that, starting from UML diagrams, derives an analysis model based on the Klaper language. The novelty of this approach is that the transformations are based on two techniques typically used in functional transformations, that are relational and graph grammar-based techniques.

In [17], MDA is viewed as a suitable framework to incorporate various analysis techniques into the development process of distributed systems. In particular, the work focuses on response time prediction of EJB applications by defining a domain model and a mapping of the domain onto a Queueing Network meta-model.

In [14], the authors aim at helping designers to reason on non-functional properties at different levels of abstraction, likewise MDA does for functional aspects. They introduce a development process where designers can define, use and refine the measurement necessary to take into account QoS attributes. Their ultimate goal is either to generate code for runtime monitoring of QoS parameters, or to provide a basis for QoS contract negotiation and resource reservation in the running system.

However, none of such approaches defines a global framework that embeds non-functional aspects within the whole MDA (i.e. at CIM, PIM and PSM levels) in an integrated manner. To fill this lack, in [5] we have introduced a framework that extends MDA to consider non-functional aspects. Such framework, called Non-Functional MDA (NFMDA), embeds, beside the typical MDA models and transformations, new models and transformations related to non-functional validation activities.

[1] The major references for this type of approaches in the field of software performance can be found in [1].

In this paper we provide two instances of the NFMDA framework, one related to performance (namely **PRIMA**) and another one to reliability (namely **COBRA**), both working at platform independent and platform specific level. Since automation is a key factor in MDA, we also discuss the tool support that should be provided to instantiate the NFMDA framework on new approaches.

Goal of this paper is to emphasize, through the instantiation of NFMDA on existing approaches in different non-functional domains, that platform independent/specific aspects actually occur also in non-functional domains, and to demonstrate that our framework nicely supports their manipulation through appropriate types of models and model transformations.

The paper is organized as follows: Section 2 briefly introduces the NFMDA framework, Section 3 discusses the issue of tool support in NFMDA; in Section 4 we instantiate the framework on two approaches for performance and reliability validation, and finally in Section 5 we give the conclusive remarks.

2 Non-Functional MDA Framework

In this section we summarize our extended view of MDA, already presented in [5] and illustrated in Figure 1. Beyond the canonical MDA models and transformations the NFMDA framework contains three additional types of models and three additional types of transformations/relationships (see the right side of Figure 1) that we have introduced to keep Non-Functional (NF) aspects under control. The new types of models are: CINFM, PINFM, PSNFM[2].

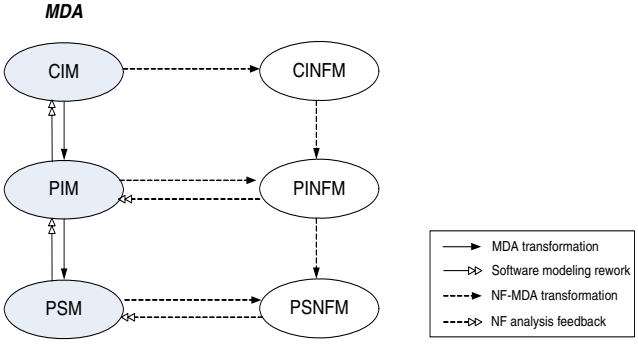

Fig. 1. The NFMDA framework

Figure 2 provides a graphical flavor of NFMDA. All NFMDA instances share the typical MDA models that are placed in the central cylinder of the figure. Each instance is represented as a wing departing from the cylinder, because each

[2] For sake of space we cannot provide details of the NFMDA framework, but they can be found in [5].

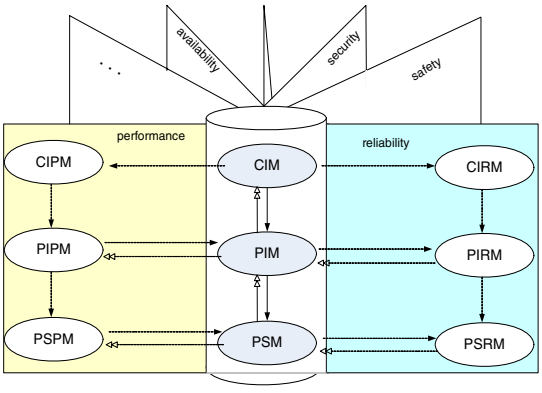

Fig. 2. Views of the NFMDA framework

NF property of a software/hardware system represents only a different view of the system, but it may need different models, languages and tools to be modeled and analyzed.

CINFM - A Computation Independent Non-Functional Model represents the requirements and constraints related to an NF aspect (such as performance, reliability, security, cost, etc) that can be formulated at different levels of detail (e.g. component level, functionality level, system level).

PINFM - A Platform Independent Non-Functional Model is a representation of the business logics of the system along with estimates of NF characteristics, such as the amount of resources that the logics needs to be executed.

PSNFM - A Platform Specific Non-Functional Model contains variables and parameters that represent the software structure and dynamics, as well as the platform where the software will be deployed. In an NF context a platform must include characteristics of the underlying hardware architecture such as the CPU speed and the failure probability of a hardware connection.

The new transformations and relationships we introduced among the models in Figure 1 are:

NFMDA horizontal transformation - It transforms a software model into the corresponding model suitable to evaluate an NF aspect of the system at any level in the MDA hierarchy. In Figure 1, CIM→CINFM, PIM→PINFM and PSM→PSNFM are horizontal transformations. The model transformations belonging to this class share a two-steps structure: the software model is first annotated with additional data, thereafter the annotated model is transformed into a model from which we estimate the NF aspect of interest.

NFMDA vertical transformation - Even though it seems to play in an NF domain a similar role to the one of MDA transformations, such transformation is instead intended to provide an input contribution to the horizontal transformation. In other words, often the horizontal transformations needs some input from the one-step-higher NF model in the hierarchy.

Two types of arrows with double empty peak also appear in Figure 1 to give completeness to the software NF analysis process, and they represent the reverse paths after the analysis takes place.

Dashed arrows with double empty peak represent the feedback that originates from the evaluation of an NF model. Continuous arrows with double empty peak are direct consequences of NF feedback. They represent the rework necessary on the software models to embed the changes suggested from the analysis.

3 Tool Support for the NFMDA Framework

Automation is undeniably a key factor in MDA approaches. To be embedded into the NFMDA framework, an NF validation approach should be supported by a tool that provides automation to all the validation steps, identified in Figure 1, that span from *model generation*, to *model analysis* and *results interpretation*.

However, any NF validation approach cannot be completely automated, as the annotation of software models with non-functional parameters is necessarily a manual step that has to be performed by experts of the non-functional domain of the approach[3].

In general, automated support for *model analysis* is available and downloadable from the net. As opposite, due to the young age of the *model generation* field, few existing approaches generating quantitative models are supported by stable and reliable tools. Even the existing tools, quite often, do not deal with all the transformation rules defined in the approaches, due to the complexity of the transformations and/or the difficulty of the model representation.

However, several researchers are recently spending a lot of effort to implement their methodologies for NF validation of software artifacts in a more systematic way. Two alternative implementation techniques have been considered: (i) ad-hoc algorithms and (ii) model-transformation techniques.

Ad-hoc algorithms make use of programming languages like C and Java. All the logics of the model generation has to be carefully implemented, including the order in which the generation rules must be applied, the management of the internal representation of the source and target models and the traceability among source and target entities.

Model transformation techniques, instead, are based on languages and tools created to provide (general) means for transformations among models specified through meta-models [15]. In this case, the implementation must only cope with the model generation rules without taking care of the way such transformation is actually executed. Moreover, if a transformation language embeds traceability between models, then the approach can also provide a mechanism that traces back, on the source model, the analysis results.

Finally, *results interpretation* and consequent feedback generation are still open points in this domain. At the moment, few and primitive guidelines [18,19,4]

[3] The collection of values for the annotation parameters may be a non trivial activity, but it is out of scope of this paper.

or simple annotations of analysis results on the source models [2] have been proposed in the performance domain, but much work must still be done.

4 Two NFMDA Framework Instances

In this section we embed within the NFMDA framework two existing approaches to transform software models into NF models. The first approach is related to performance modeling and validation [7], the second one to reliability [8]. Both these approaches have been implemented with ad-hoc techniques for the generation and analysis of the quantitative model(s) they are based on.

It is worth noting that several NF validation approaches do not work at both the PIM and the PSM level, but they are suitable for just one of these levels [2,13]. Both approaches presented here work at all MDA levels, and they have been selected to show the complete instantiation of the NFMDA framework on two different NF domains.

4.1 Performance Analysis in MDA

The **PRIMA** methodology has been introduced in [7] and has evolved, on one side, towards validation of mobility-based software systems [10] and, on the other side, towards Component-Based Software Performance Engineering [3]. It was originally conceived as a model-based approach to estimate the performance of software systems ready to be deployed. Due to the type of performance models produced, **PRIMA** has the intrinsic capability of analyzing also software systems at a platform independent level. The prerequisites to apply the approach consist of: modeling the system requirements through an UML Use Case Diagram, modeling the software dynamics through UML Sequence Diagrams, and modeling the software-to-hardware mapping through an UML Deployment Diagram.

Figure 3 represents **PRIMA** as plugged into the NFMDA framework. CIM, PIM and PSM are shown in the left-hand side of the figure. In the right-hand side of the figure the additional models introduced in our framework are shown, which have been renamed in this specific instance as Computation Independent Performance Model (CIPM), Platform Independent Performance Model (PIPM), Platform Specific Performance Model (PSPM). Names of tools able to perform transformation steps have been reported on the edges corresponding to the transformations between models.

The system requirements are expressed by an Use Case Diagram that represents the Computation Independent Model of the system. The Use Case Diagram has been also reported as part of the Platform Independent Model because its annotated version is part of the transformation to the PIPM. The PIM is completed by the Sequence Diagrams that model the application dynamics through a set of system scenarios. Finally, the Platform Specific Model contains different types of information: the name of the platform (e.g. CORBA, J2EE, etc.) determining the type of code that must be generated, and a Deployment Diagram determining the software-to-hardware mapping of the application.

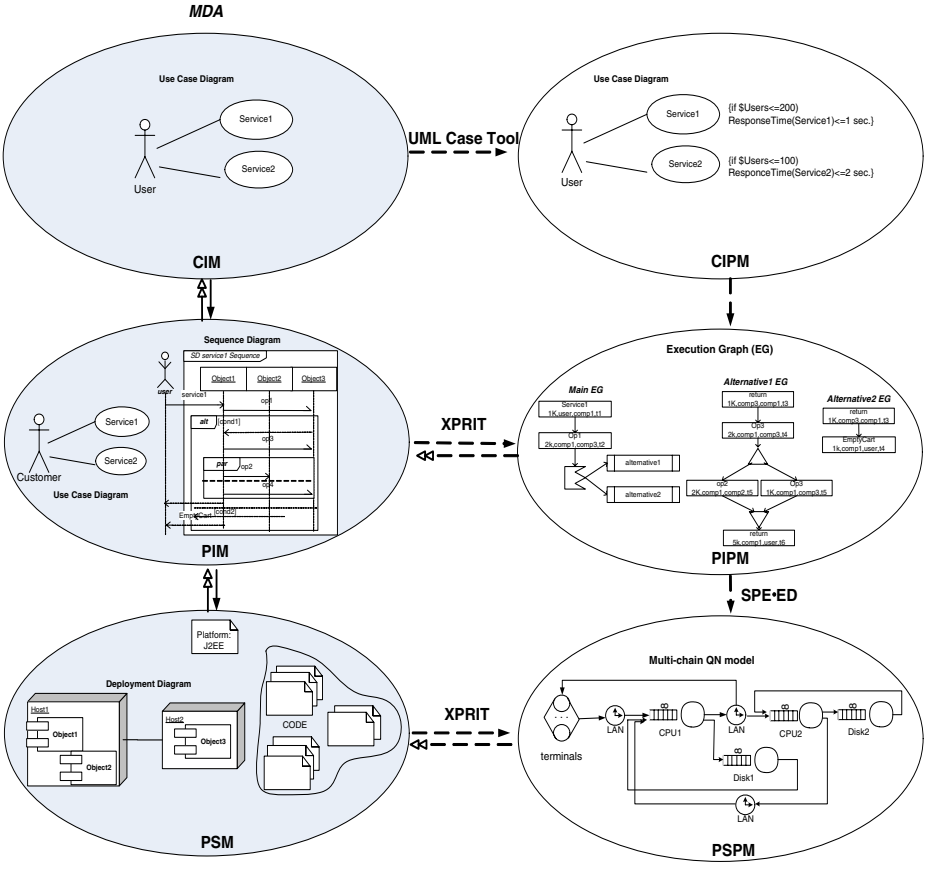

Fig. 3. The **PRIMA** approach

The topmost performance model, namely the CIPM, is obtained by the CIM (i.e. the Use Case Diagram) with annotations on use cases that express the performance requirements of the system. As an example, in Figure 3 a requirement on *Service1* entails that the average response time of such use case must not be larger than 1 second when the customers in the system are less than 200. A similar requirement is associated to *Service2*. The transformation CIM→CIPM can be easily achieved by annotating an Use Case Diagram within an UML CASE tool (e.g. Poseidon [21]).

The PIPM is represented by an Execution Graph (EG) [19]. An EG is basically a flow graph that models the software dynamics, and its building blocks are: basic nodes that model sequential operations, fork and join nodes that model concurrency, loop nodes that model iterative constructs, branching nodes that model alternative paths, and composite nodes that model separately specified macro-steps. In addition to the software dynamics, in an EG a *demand vector* is attached to each basic node to model the resources needed to execute the

corresponding operation. It is worth to note that the amount of resource needed cannot be specified, at this level in the hierarchy, by classical measures like CPU time and disk accesses. Each element of the demand vector represents a high-level metric, such as *screen operation*, *message sending*, etc. [19]. An estimated amount of each metric can be attached to any basic block in the EG. Hence an EG is a Platform Independent Performance Model as high-level metrics, although representing performance data, are not bound to any platform.

In order to generate the PIPM, the PIM is annotated with the following data: (i) Use Case Diagram - probability that an actor enters the system and probability that the actor requires a certain use case (i.e. the operational profile); (ii) Sequence Diagram - size of messages exchanged over the interactions, probabilities over the branching points, average number of loop iterations.

The annotated PIM (i.e. Use Case Diagram and Sequence Diagrams) is then transformed into an EG as follows. Probabilities on the Use Case Diagram are combined to carry out the probability of each use case to occur. The latter represents also the probability that the corresponding Sequence Diagram is executed. Thereafter, an EG is built for each Sequence Diagram by visiting the diagram and piecewise translating each fragment encountered in an EG specific pattern. EG patterns are then combined following the structure of the Sequence Diagram. During the visit, the performance annotations are used to build demand vectors attached to EG basic blocks. Finally, all EGs are lumped into a single one that starts with a branching node, where each EG represents an alternative path. The probabilities over the outgoing paths correspond to the ones carried out from the Use Case Diagram[4]. The tool supporting the modeling of EGs (i.e. SPE·ED [22]) allows stand-alone and worst-case analysis of an EG. Obviously, the validity of the analysis undergoes the trustability of the estimates of model parameters.

In this step, the CIPM brings the target performance results that have to be compared with the one obtained from the PIPM solution. In Figure 3 the XPRIT label on the edges connecting (in both directions) PIM and PIPM represents the name of the tool that automates such transformation [6].

Going down in the hierarchy of Figure 3, the adopted PSPM is a multi-chain QN model. The semantics of the QN is as follows: each service center represents a hardware device, and the jobs traversing the network represent the software load of the devices.

In this approach the PSM→PSPM transformation makes a large use of the PIPM. First, the Deployment Diagram is annotated with information on the internal configuration of each host (i.e. number and speed of CPUs, number and access time of hard disks, etc.). This information is used in the transformation process to build the topology of the QN that represents the hardware platform. The EG structure determines the number of job classes (i.e. chains) that traverse the QN. The combination of values of demand vectors and performance information about the platform (e.g. CORBA) allow to determine the amount of

[4] For sake of space, we cannot provide more technical details of the transformation process; however, readers interested may refer to [7].

resources that each class of jobs requires to each hardware device [19]. In Figure 3 the XPRIT label on the edges connecting (in both directions) PSM and PSPM again indicates the tool that automates such transformation. The SPE·ED label on the edge connecting PIPM and PSPM indicated that the EG modeling tool also allows to elaborate the EG demand vectors in order to parameterize the QN.

It is obvious that at this level in the MDA hierarchy a different type of performance analysis is pursuable. The QN embeds all the software and hardware parameters that are needed for a canonical performance analysis. They have been collected and/or monitored on the actual deployed system. End-to-end response time, utilization and throughput of any platform device can be computed by solving the QN model, and the results can be compared to the measures of the actual system. Once validated, such model can be used for performance prediction. System configurations and workload domains that are unfeasible to experiment in the actual system can be represented in the model in order to study the behavior of the system under different (possibly stressing) performance scenarios.

The outputs of the PSPM evaluation represent the feedback on the PSM, which needs to be modified if performance do not satisfy the requirements. As illustrated in section 2, the model rework on the software side could propagate up to the higher level models in case no feasible change can be made on the PSM to overcome the emerging performance problems.

Behind PIPM/PSPM duality there is the intuitive concept that the performance analysis results can be expressed with actual time-based metrics only after a PIM is bounded to its platform and becomes a PSM. Obviously the results coming out a PIPM evaluation are not useful to validate the model against the system requirements, because they may take very different time values on different platforms. However, the following three types of actions can originate from this analysis.

(i) **Lower and upper bounds** on the system performance can be evaluated if some estimates of the performance of the possible target platforms are available. For example, if the lower bound on a system response time is larger than the corresponding performance requirement, then it is useless to progress in the development process as performance problems are intrinsic in the software architecture. It is necessary to rework on the software models. However, even when the results are not so pessimistic it is possible to take decisions that improve the software architecture.

(ii) In order to identify the most **overloaded components**, the utilization and/or queue length of each service center in our PIPM must be computed vs the system population. An overloaded component has a very long waiting queue and represents a bottleneck in the software architecture. Some rework is necessary in the PIM to remove the bottleneck.

(iii) Either as a consequence of the above decisions or as a planned performance test, different (functionally equivalent) **alternative software designs** can be modeled as PIMs, and then their performance can be compared through their PIPMs in order to select the optimal one.

4.2 Reliability Analysis in MDA

The methodology for reliability modeling and validation that we consider here has been first presented in [16] to estimate the reliability of a component-based software system as a function of the failure probabilities of components and the operational profile. Failures of hardware connections have then been embedded into the model [8]. In order to embed this approach into our NFMDA framework, we have further enhanced the reliability model by embedding failures of hardware sites. We have chosen this approach to the reliability validation for two main reasons: (i) it well suits to be interpreted in platform independent and platform specific views (as we show in this section), and (ii) well-founded transformations from UML models have been proposed to generate the reliability model [8]. For sake of readability, in the remainder of the paper, we will refer to this approach as **COBRA** (COmponent-Based Reliability Assessment).

In Figure 4 we show the **COBRA** approach as embedded into NFMDA. Software models are shown in the left-hand side of the figure and reliability models in the right-hand one. Following the naming that we have adopted in NFMDA, the reliability models are labeled as: Computation Independent Reliability Model (CIRM), Platform Independent Reliability Model (PIRM), and Platform Specific Reliability Model (PSRM). Again, names of tools able to perform transformation steps have been reported on the edges corresponding to the transformations between models.

The software models of **COBRA** (i.e. the ones on the left-hand side of the figure) are represented with the same UML diagrams as in **PRIMA**. Use Case Diagram represents system requirements and, together with Sequence Diagrams, represent the system dynamics at the platform independent level. The Deployment Diagram, together with Sequence Diagrams, represent the mapping of software to hardware in the PSM. In this case, the PSM does not include the software code and the name of the platform.

The topmost reliability model, namely the CIRM, can be obtained by the CIM with two different types of annotations: an annotation attached to an use case expresses a minimum reliability threshold that is required for the functionality corresponding to the use case; an annotation attached to the whole diagram represents a minimum reliability threshold required for the whole system. As an example, in Figure 4, we have annotated a 0.99 reliability threshold for the *Service1* use case, whereas a 0.982 threshold has been annotated for the whole system. We will consider here only reliability thresholds on the whole system. However, the same type of modeling and analysis can be applied at use case level by restricting the ranges of the reliability equations.

PIRM and PSRM are represented in **COBRA** by mathematical equations that link the reliability of the whole system to the failure probabilities of its components. The difference between the models is that in the PIRM the reliability of the system only depends on the failures of software components (independently of the platform they will be deployed), whereas in the PSRM the failures of the

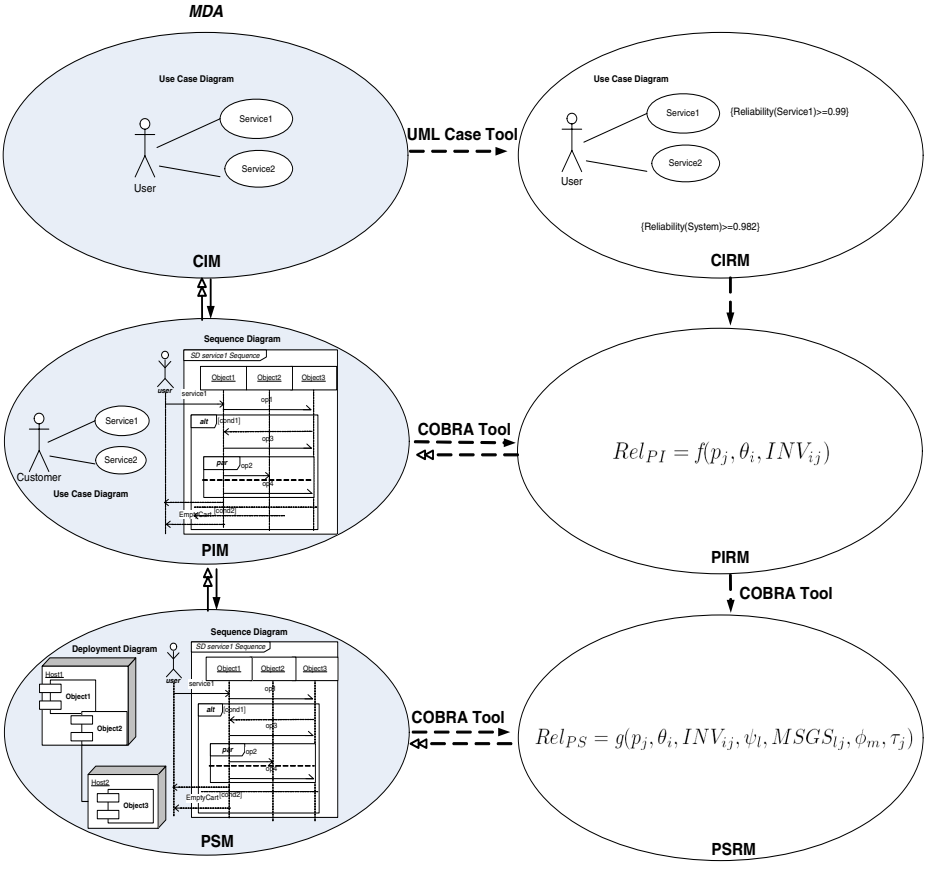

Fig. 4. The **COBRA** approach

platform sites and hardware connections are also taken into account. In both cases, the values assigned to the model variables are extracted from annotations on the UML diagrams of PIM and PSM.

In **COBRA** the PIRM is represented by the following equation [16]:

$$Rel_{PI} = \sum_{j=1}^{K} p_j \cdot \prod_{i=1}^{N} (1 - \theta_i)^{INV_{ij}} \tag{1}$$

where:

- N is the number of software components;
- K is the number of scenarios modeling the system dynamics;
- Rel_{PI} represents the system reliability at the platform independent level, that is the probability of no software failures during the system operation;
- θ_i represents the probability of failure on demand of the component i [20];

- INV_{ij} represents the number of invocations of component i within the dynamics of Sequence Diagram j;
- p_j is the probability of execution of Sequence Diagram j;

In practice, the system reliability in (1) is the probability that none of the components fails during the execution of the scenarios represented by the Sequence Diagrams. This type of reliability model is also known as *fail-and-stop*, in that any single failure of a component represents a failure of the whole system.

In order to generate the PIRM, the PIM is annotated with the following data: (i) Use Case Diagram - operational profile as in **PRIMA**; (ii) Sequence Diagram - number of invocations per component. The PIM→PIRM transformation in this case is trivial, in that values of annotations are extracted from the PIM and properly assigned to the variables of equation (1). In this domain, the CIRM has the same role as the CIPM, in that it brings the required reliability threshold that has to be compared with the one obtained from the PIRM solution.

Going down in the hierarchy of Figure 4, the PSRM is represented by an equation that takes into account also hardware failures, as follows:

$$Rel_{PS} = \sum_{j=1}^{K} p_j \cdot (\prod_{i=1}^{N} (1-\theta_i)^{INV_{ij}} \cdot \prod_{l=1}^{C} (1-\psi_l)^{MSGS_{lj}} \cdot \prod_{m=1}^{S} e^{-\phi_m \cdot \tau_j}) \quad (2)$$

where the following additional variables have been introduced:

- S is the number of platform sites;
- C is the number of hardware connections among platform sites;
- Rel_{PS} represents the system reliability at the platform specific level, that is the probability of no software/hardware failures during the system operation;
- ψ_l represents the probability of failure on demand of the hardware connection l;
- $MSGS_{lj}$ represents the number of messages exchanged over the connection l during the execution of Sequence Diagram j;
- ϕ_m represents the failure rate of the platform site m, that is the inverse of its Mean Time To Failure [20];
- τ_j is the execution time of scenario j over the targeted platform;

In practice, the system reliability in (2) is the probability that none of the components, the hardware connections and the platform sites fail during the execution of the scenarios represented by the Sequence Diagrams. Even in this case there is an underlying assumption of *fail-and-stop*, in that each type of failure induces a failure on the whole system.

Note that the failures of platform sites have been modeled by a continuous (exponential) failure rate, as opposite to all the other ones that have been modeled by probabilities of failures on demand (i.e. as originated from discrete time events). The rationale of this assumption is that hardware sites in practice are never idle (i.e. there are system processes always running), thus their failures may originate in any instant of time.

Obviously equations (1) and (2) represent only an example of reliability model that nicely fits into the NFMDA framework. Other assumptions can be made that bring to more sophisticated reliability models, such as considering error propagation among components or dependencies among hardware and software failures. However, it is out of scope of this paper to discuss limitations of non-functional models, as we have remarked in Section 1.

In order to generate the PSRM, the PSM is annotated with the following data: (i) Sequence Diagram - number of messages exchanged between pairs of components, execution time of the scenario; (ii) Deployment Diagram - probability of failure on demand of hardware connections, failure rates of platform sites. Similarly to the PIM→PIRM transformation, the values of annotations can be then extracted from the PSM and properly assigned to the variables of equation (2).

It is evident, by simply comparing equations (1) and (2), that the PIRM originates an optimistic evaluation of the system reliability, because the value of (1) will be always as large as the one in (2). However, an overestimation of system reliability can be accepted when failure data related to the hardware platform are not yet available, i.e. at platform independent level. In fact, a PIRM solution and sensitivity analysis may support early decisions in the software lifecycle, such as: (i) distributing the development and testing efforts over critical components, (ii) selecting COTS components to be plugged in an existing architecture, (iii) allocating computation complexity in software components and structuring the software architecture.

As soon as platform data are available, a PSRM can be generated and solved. The PSRM solution will support late decisions in the software lifecycle, such as the mapping of software components on platform sites, and the influence of a certain type of hardware connection on the whole system reliability.

The outputs of PIRM and PSRM represent, respectively, feedback on PIM and PSM. The latter ones can be modified if system reliability does not satisfy the requirements. The model rework on the software side could propagate up to the higher level models in case no feasible change can be made on the lower level models to overcome the emerging reliability problems.

5 Conclusions

NFMDA provides an unifying view on the NF validation in MDA, and it can be instantiated in several NF domains like performance, security, availability, etc. Platform Independent and Platform Specific concepts belong to many software validation processes, and bringing them to the evidence of an MDA structure is very advantageous to make these activities acceptable from the software engineering community.

In this paper we have shown how two existing approaches to the performance and reliability validation can be plugged into NFMDA framework. However, modern software systems claim for integrated validation of non-functional aspects, since the validation of a non-functional attribute in isolation (such as

reliability) may not be meaningful due to intrinsic tradeoffs between attributes. For example, authentication mechanisms are usually introduced to improve the security of a system that, at the same time, may seriously degrade the system performance.

The NFMDA framework has been conceived to be compliant with the integrated validation of non-functional aspects, as there is no limitation on the types of non-functional models that can be generated. Obviously, the horizontal transformations may become very complex with a growing complexity of the non-functional models. However an integrated NF analysis, while increasing the complexity of the transformations, may save effort in model annotations (hence in parameter collection) because several parameters can be shared across non-functional attributes. A typical example of parameter sharing is represented by the operational profile. All non-functional aspects are heavily affected from the operational profile hence, once annotated on a software model, it can be used as input to several transformations towards different types of non-functional models.

Several directions may be taken in the future within this context.

First, basing on our experience, several other approaches match this framework in all its structure or in part of it, therefore they shall be easily plugged into NFMDA. We guess that this shall be one of the priorities for the future research in this field.

The suitability of the NFMDA framework for less established and more cross cutting NF aspects (such as security) shall be investigated. Likewise the practical applicability of the framework should be experimented on real case studies; however, experimentation should span over different application domains where different non-functional aspects can be critical.

Large investigation shall be devised to the feedback paths in the NFMDA framework, because the translation of validation results in actual design alternatives is still an open issue in all the NF domains. For example, the use of model transformation techniques that guarantee traceability would be helpful to straightforwardly report the analysis feedback on the software models.

Finally, the NFMDA framework represents the basis to enhance NF validation through: (i) the definition of languages and ontologies for representing CINFM, PINFM and PSNFM, and (ii) model transformation languages, techniques and tools targeting NF models.

References

1. S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, Model-based Performance Prediction in Software Development: A Survey, *IEEE Trans. on Software Engineering*, 30(5):295-310, 2004.
2. S. Balsamo, M. Marzolla, A Simulation-Based Approach to Software Performance Modeling, *Proc. Joint 9th European Software Engineering Conference (ESEC) & 11th SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pp. 363–366, Helsinki, FI, 2003.

3. A. Bertolino, R. Mirandola, CB-SPE Tool: Putting Component-Based Performance Engineering into Practice, *Proc. of CBSE 2004*, pp.233-248, 2004.
4. V. Cortellessa, A. Di Marco, P. Inverardi, Three Performance Models at Work: A Software Designer Perspective, *Electr. Notes Theor. Comput. Sci.*,vol. 97, pp. 219-239, 2004.
5. V. Cortellessa, A. Di Marco, P. Inverardi, Non-functional Modeling and Validation in Model-Driven Architecture, *Proc. of Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA 2007)*, Mumbay, India, 2007 (To Appear).
6. V. Cortellessa V., M. Gentile, M. Pizzuti, XPRIT: an XML-based tool to translate UML diagrams into Execution Graphs and Queueing Networks, *Proc. of QEST 2004 (short papers)*, 2004.
7. V. Cortellessa, R. Mirandola, PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams, *Science of Computer Programming*, Elsevier Science, 44(1):101-129, 2002.
8. V. Cortellessa, H. Singh, B. Cukic, E. Gunel, V. Bharadwaj, Early reliability assessment of UML based software models, *Proc. of 3rd ACM Workshop on Software and Performance*, 2002.
9. K. Czarnecki, S. Helsen, Classification of Model Transformation Approaches, *Proc. of OOPSLA03 Workshop on Generative Techniques in the Context of MDA*, 2003.
10. V. Grassi, R. Mirandola, PRIMAmob-UML: a methodology for performance analysis of mobile software architectures, *Proc. of WOSP 2002*, pp. 262-274, 2002.
11. J. Miller (editor), Model-Driven Architecture Guide, omg/2003-06-01 (2003).
12. G.N. Rodrigues, G. Robets, W. Emmerich, J. Skene, Reliability Support for Model Driven Architecture, *Proc. of WADS 2003, LNCS 3069*, pp.79-98, 2003.
13. G.N. Rodrigues, D. S. Rosenblum, S. Uchitel, Reliability Prediction in Model-Driven Development, *Proc. of Models Conference, LNCS 3713*, pp.339-354, 2005.
14. S. Rottger, S. Zschaler, Model-driven development for non-functional properties: refinement through model transformation, *Proc. of UML 2004*, LNCS 3273, pp.275-289, 2004.
15. A. Sabetta, D.C. Petriu, V. Grassi, R. Mirandola, Abstraction-raising Transformation for Generating Analysis Models, *Proc. of Models 2005 Satellite Events, LNCS 3844*, pp. 217-226, 2005.
16. H. Singh, V. Cortellessa, B. Cukic, E. Gunel, V. Bharadwaj, A Bayesian Approach to Reliability Prediction and Assessment of Component Based Systems, *Proc. of 12th IEEE International Symposium on Software Reliability Engineering*, 2001.
17. J. Skene, W. Emmerick, Model-driven performance analysis of Enterprise Information Systems, ENTCS 82(6), 2003.
18. C.U. Smith, L.G. Williams, Software performance antipatterns. *Proceedings of the 2nd international workshop on Software and performance (WOSP00)*, pp. 127–136, Ottawa, Ontario, Canada, 2000.
19. C.U. Smith, L.G. Williams, Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Addison-Wesley, 2002.
20. K. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, John Wiley and Sons, New York, 2001.
21. www.gentleware.com
22. www.perfeng.com