Challenges in Integrating the Analysis of Multiple Non-Functional Properties in Model-Driven Software Engineering

Dorina C. Petriu
Carleton University
Department of Systems and Computer Engineering
Ottawa ON, Canada, K1S 5B6
petriu@sce.carleton.ca

ABSTRACT
This vision paper discusses the challenges of integrating the analysis of multiple Non-Functional Properties (NFP) in the model-driven software engineering process, where formal analysis models are generated by model transformations from annotated software models. The paper proposes an integration approach based on an ecosystem of inter-related heterogeneous modeling artifacts intended to support consistent co-evolution of the software and analysis models, cross-model traceability, incremental propagation of changes across models and (semi)automated software process steps. Another goal is to investigate new metaheuristics approaches for reducing the size of the design space to be explored in the search for a design solution that will meet all the non-functional requirements.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Modeling techniques, Performance Attributes; D.2.8 [Software Engineering]: Metrics - performance measures.

General Terms
Performance, Design, Verification.

Keywords
Model-driven engineering, Non-functional properties, model-driven analysis, ecosystem of models.

1. INTRODUCTION
This vision paper discusses some of the challenges raised by the seamless integration of the analysis of multiple Non-Functional Properties (NFPs), such as performance, reliability, availability, fault-tolerance, scalability, security, maintainability, cost, etc., into the Model-Driven Engineering (MDE) process. The purpose is to guide the design choices from an early stage and to insure that the system under construction will meet all its nonfunctional requirements. The NFP analysis uses formal models (also known as quality models) based on existing formalisms and tools (e.g., queueing networks, stochastic Petri nets, stochastic process algebras, Markov chains, fault trees, probabilistic time automata, etc.). Such analysis models can be automatically generated by model transformations from the software models built for development.

WOSP-C'15, January, 2015, Austin, TX, USA.

The following ingredients are necessary when integrating the analysis of a single NFP (for instance, performance) in the model-driven software development process: a) modeling language support for adding performance annotations to the software design model; b) tool support for the *forward path* that includes model-to-model transformation of the annotated software model to a performance model, solving the performance model and obtaining the performance results; c) tool support for the *backward path* that includes analyzing the performance results, finding the problem, suggesting changes for improvement in the performance model, which are translated into refactoring advice for the software model, and d) a software process describing the entire workflow for model-driven software development, which integrates the analysis of multiple NFPs. A more detailed discussion of the above ingredients is presented in section 2.

The big picture becomes even more complex if we consider the integration of the analysis of multiple NFPs. Developing software systems that exhibit a good trade-off between multiple NFPs is difficult because the design of the software under construction and its underlying platforms have a large number of degrees of freedom spanning a very large discontinuous design space that cannot be exhaustively explored, so metaheuristic approaches are employed. Another challenge is due to the fact that some NFPs are conflicting (for example, security and performance). Therefore, the developers must make trade-off decisions to improve one property at the expense of the other, but also need to balance the respective properties, so that eventually all non-functional requirements are met.

This paper proposes an integration approach that addresses some limitations in the existing work: i) there is little work on automating the backward path; ii) the software and analysis models are isolated and cross-model queries and constraints are not supported; iii) there is no traceability support between the elements of different types of models (e.g., between software model elements and the performance model elements generated from them); iv) co-evolution of software models and corresponding analysis models is not supported; v) incremental propagation of changes across models is not supported; and vi) many software process task are manual, being slow and error prone.

The proposed integration approach is based on an ecosystem of inter-related heterogeneous modeling artifacts, such as: software and analysis models and metamodels; model transformations;

solvers; inter-model traceability models and metamodels; analysis results, etc. An important role of the ecosystem is to support consistent co-evolution of the software and analysis models. The ecosystem contains a top-level model that describes the modeling artifacts contained in the ecosystem, their relationships, plus other relevant metadata needed for model management. The ecosystem and its top-level model help automating the software process steps that involve multiple modeling artifacts, as the top-level model describes the dependency relationships between models and other necessary metadata. (An example of such a step is the derivation of an analysis model for a given NFP, which requires retrieving from a repository the necessary models and metamodels, invoking one or more transformations, passing the right parameters, and registering the newly produced model(s) into the repository). The aim is to relieve the developers from manual model management operations as much as possible during the software process, asking for human intervention only when new information needs to be provided by the designers or their judgement/decisions are required. The purpose of this automation is two pronged: to raise the efficiency and usability of the NFP analysis during MDE and to enhance the quality of the software products.

In order to deal with the design space explosion problem mentioned before, the proposed research will investigate new metaheuristics approaches for reducing the size of the design space to be explored in the search for an optimal solution (more detail in sections 3 and 4).

The paper is organized as follows: section 2 discusses the related work and current limitations, section 3 presents the objectives and related research questions, section 4 discusses methods for the proposed approach, and section 5 gives the conclusions.

2. RELATED WORK

2.1 Current state of the art

The emergence of model-driven software engineering, which is based on abstraction and automation, has enabled not only the generation of code from models, but also the generation of formal analysis models for NFP verification. Such models are derived from software models (or selected views thereof) annotated with information specific to the property to verified. How to do such annotations was a question already addressed by many researchers and practitioners. In the UML world, such annotations are done via UML profiles, which are a standard extension mechanism supported by UML editors. OMG has adopted two standard profiles for performance and schedulability annotations: an earlier profile, SPT [21] for UML 1.x and a more recent replacement, MARTE [22] for UML 2.x. The adoption of SPT and MARTE has enabled research on the automatic generation of different kinds of performance models from annotated UML, as surveyed in [9]. In the case of security analysis, a UML profile named UMLsec defined in [15] is used to model and systematically verify the correctness of security protocols. In [13], the Alloy Analyzer tool is used to analyze security assertions of models written in UML and OCL, which are automatically transformed into the Alloy language, a fully declarative first-order logic language designed to model complex systems. In the case of dependability and its many attributes (availability, reliability, fault tolerance, safety, maintainability) there is a body of work surveyed in [2] where different solutions proposed in literature for dependability specifications via ad-hoc UML profiles and approaches for generating analysis models are discussed. A survey on architecture-based software reliability analysis is found in [12]. In [3] it is proposed the DAM profile for dependability analysis, specialized from MARTE.

In software engineering have been defined many software development processes (also known as software development methodologies), some of which have been adapted to the model-driven paradigm. In our research we are interested in a subset of software processes which include the verification of one or more NFPs based on quantitative analysis models. Examples are the Software Performance Engineering process proposed by Smith in [25], the risk reduction-based process from [13] and the performance antipattern-based process from [26][8].

A relevant research challenge is how to use multiple NFP analysis models in order to find good design (preferably optimal) solutions. A thorough survey on software architecture optimization methods is found in [1]. In principle, the problem of balancing multiple NFPs lends itself to multi-criteria optimization, but in practice the complexity of the system and the size of the design space make the problem intractable. According to the literature, traditional optimization methods have been used mostly in cases where a single NFP analysis model was required. For instance, integer linear programming is used in [19] for the optimization of application deployments across a cloud, based on the use of a Layered Queueing Network (LQN) model [30]. When multiple NFP models are considered, metaheuristic search techniques (e.g., genetic algorithms, simulated annealing, etc.) are used to find better (if not the best) design models. An example is [20], where a multi-criteria genetic algorithm is applied to software architectures modeled with the Palladio Component Model (PCM), supporting quantitative performance, reliability, and cost prediction, where the performance model is obtained by a PCM-to-LQN transformation [17], the reliability model by a PCM-to-Markov Chain transformation and the cost by a simple additive model.

Another approach for balancing different NFPs is using decision support systems for reasoning under uncertainty, based on Bayesian Belief Network models to derive fitness scores for alternative designs [13]. The uncertainty of the problem domain is represented through conditional probabilities, which specify the modeler's belief about the strengths of the cause-effect relations between different domain entities represented in the model. A different approach for finding good design solutions when a single NFP is considered at a time makes use of rule-based techniques. For instance, in [31] diagnostic and design-change rules are used to automate the performance analysis and to explore design changes using an LQN model, until an acceptable solution is found. The advantage is that this approach gives insight into the causes for poor performance and how to fix them.

There are two existing directions of research relevant to ecosystems of models: global model management [5] [11] and multi-paradigm modeling [18] [23]. Both consider a system of inter-dependent heterogeneous models, described by a top-level "model of models" (named megamodel in [5]) intended to support the inter-working of models and inter-operation of languages.

Different types of models and modeling artifacts are involved in the MDE process, which include software development models and formal analysis models for different NFPs. A software model may have many views representing different structural and behavioural aspects of the system. Each analysis model for a given NFP is derived from a specific set of system views extended with extra information characteristic to the NFP of

interest. For instance, a performance model is derived from structural views representing the high-level software architecture and the software to hardware allocation, as well as a few behavioural views representing key performance scenarios; all views are annotated with performance information using MARTE [29]. After the performance analysis, changes in the performance model for improving the system performance must be propagated back to the corresponding software views and eventually to the main software model and all its other views.

2.2 Current Limitations

The proposed integration approach addresses a number of limitations listed below, which are found in the existing work:

a) Although there is a lot of work on transforming software models into analysis models, there is much less work on automatic analysis and diagnosis of NFP problems, and on giving feedback for improvement to the software developers.

b) The models are isolated and their relationships, although known by the developers, are not formally recorded, so they cannot be used for automation.

c) There is no traceability support between the software and analysis models, which makes it impossible to automate the import of analysis results in the software model context.

d) There is no support for (semi)automatic co-evolution of the software and analysis models, so the co-evolution has to be done manually or is not considered at all.

e) There is no support for incremental propagation of small changes between the software and analysis models.

f) Many software process tasks are performed manually, which makes the whole process inefficient and error-prone.

3. OBJECTIVES AND RESEARCH QUESTIONS

The overall objective of the proposed integration approach is to add more "engineering" to model-driven software engineering by supporting the seamless integration of the analysis of multiple NFPs into the MDE process. Different NFP analysis models based on appropriate existing formalisms can be automatically derived by model transformations from the software models built for development, as explored in previous research. The software models built for development and the NFP analysis models must co-evolve together. An important research effort will go into investigating how multiple NFP analysis models can be used to find a good (preferably optimal) design solution, in which all non-functional requirements are met. Another important aspect of the proposed research is concerned with automating the software process tasks/activities related to NFP analysis as much as possible, asking for human intervention only when the developers need to provide new information and/or their judgement or decisions are required. The intended purpose of such automation is two pronged: a) to raise the efficiency and usability of the NFP analysis during MDE by eliminating error-prone manual model manipulations, and b) to enhance the quality of the software products by verifying the NFPs throughout the development process, from its early phases.

The proposed approach has a number of specific objectives that are described in the rest of the section.

3.1 Ecosystem of modeling artifacts

Objective A. *Development of an ecosystem of modeling artifacts which can support synchronized co-evolution of the software and analysis models.*

Such an ecosystem contains a large number of heterogeneous inter-related modeling artifacts (such as models, metamodels, transformations, trace-links, solvers, parameters and analysis results) and is described by a top-level model that specifies the modeling artifacts that are the members of the ecosystem, their relationships and additional information (such as location) for manipulating them. The following research questions are related to this objective:

RQ-A1. What kind of information should be specified in the ecosystem's top-level model in order to describe the various kind of modeling artifacts contained in the ecosystem, their relationships, the activities to be enacted in order to realize such relationships and the model management operations required for each type of modeling artifact? What is the metamodel of the top-level model?

RQ-A2. How to synchronize the co-evolution of the software model and the corresponding NFP analysis models in the context of an ecosystem specified as in the previous question?

RQ-A3. How can incremental propagation of changes help the co-evolution of two models whose relationship is defined by a model transformation? Can changes be propagated in any direction (e.g., from the software model to an analysis model and vice-versa) even if the transformation is unidirectional?

3.2 Inter-model traceability

Objective B. *Develop support for inter-model traceability between the elements of two models related by a relationship defined by a model transformation.*

The aim is to generate traceability links (stored externally in a new model) between the elements of the two models according to the mapping performed by the transformation. Three research questions correspond to this objective:

RQ-B1. How to extend the current trace-link concept with the capability of mapping expressions that calculate/aggregate quantitative NFP measures in the analysis and software models, by taking into account that each analysis formalism has specific ways of computing the NFP results.

RQ-B2. How to express and execute NFP-related user-defined cross-model queries, which seamlessly navigate between models via inter-model trace links?

RQ-B3. How does inter-model traceability support incremental model transformation? How are the trace-links themselves updated during an incremental model transformation?

3.3 Metaheuristics for multi-NFP optimization

Objective C. *Define and verify metaheuristic approaches for multi-NFP optimization.*

In principle, multi-NFP analysis lends itself to multi-criteria optimization, but there are severe practical limitations to applying traditional optimization techniques due to the very large size of the problem. Researchers use instead metaheuristic search

techniques (e.g., genetic algorithms, simulated annealing, etc.) to find good (preferably optimal) design solutions. However, the few metaheuristics approaches reported so far do not scale up well enough to analyze more than three or four NFPs for models of realistic size. The following research questions are related to this objective:

RQ-C1. Investigate automated diagnosis techniques for a given NFP, which identify not only the cases where the NFP is poor, but also what the causes are and how to fix the problem. How to use such diagnosis techniques as metaheuristics to exclude design space zones where the respective NFP is poor from the search space that considers all NFPs? Alternatively, how to use such diagnosis techniques to identify design space zones where the respective NFP is good, so that such zones would be explicitly included in the search space considering all NFPs?

RQ-C2. How effective are the new metaheuristic approaches from the previous question in reducing the design space to be explored in the search of a better design solution where all NFP meet their requirements?

Of particular interest are metaheuristics based on performance bottleneck diagnosis, which points very effectively to good design solutions obtained by removing the bottleneck.

3.4 Software process automation

Objective D. *Automate as much as possible the software process tasks related to NFP analysis.*

The intent is to develop techniques and tool support for semi-automated process tasks or activities related to NFP analysis for any model-driven software process, by eliminating manual operations that are error-prone and slow. Examples of process tasks to be automated are: a) generation of a given analysis model and of the corresponding traceability model from the software model or views thereof; b) solving an analysis model with an existing solver and producing analysis results; c) performing an NFP diagnosis, d) performing a design space search, etc. The following research questions are related to this objective:

RQ-D1. Considering all the actions, queries, and model manipulations performed in a software process task, what part of the task can be executed automatically based on information found in the top-level model or in any other artifact contained in the ecosystem? When is human intervention absolutely necessary (for instance, to provide new information or to make a judgement/decision)?

RQ-D2. Based on the results of the previous question, how to automate the execution of all process steps that take place between two necessary human interventions?

4. PROPOSED APPROACH

In general, the research methodology for the proposed integration approach will make use of principles, methods and technologies for model-driven engineering, such as software modeling languages, metamodeling, model transformations, model management (including model persistence, co-evolution, global model management, versioning) [5]. We will also use models for different NFPs (e.g., Layered Queueing Networks, General Stochastic Petri Nets, Stochastic Reward Nets, fault trees, etc.), their metamodels (some of which we may have to define) and will invoke existing solvers as a black-box.

In terms of technical space, we will focus on the open-source Eclipse platform, which offers implementations of the OMG standards we intend to use: UML and MARTE, XMI, OCL constraint language and QVT transformation language. A challenge for this research (where inter-model navigation and support for cross-model queries and constraints are needed) is that the standard languages mentioned above do not cross the boundaries of a single model. Therefore, we will consider also the family of transformation languages Epsilon [18] developed over Eclipse, which can express cross-model constraints and queries.

All the objectives described in section 3 include a thorough evaluation of the approaches and methods that will be developed. We will select appropriate case studies to see how effectively the developed methods work, and to identify their advantages and limitations. Below we discuss methodological issues specific to each objective.

A. Ecosystem of modeling artifacts. We will use as a basis two existing directions of research relevant to ecosystems of models: global model management [5] [11] and multi-paradigm modeling [18] [23].

A software model may have many views representing different structural and behavioural aspects of the system. Each analysis model for a given NFP is derived from a specific set of system views extended with extra information characteristic to the NFP of interest. For instance, a performance model is derived from structural views representing the high-level software architecture and the software to hardware allocation, as well as a few behavioural views representing key performance scenarios; all views are annotated with performance information using MARTE. After the performance analysis, changes in the performance model for improving the system performance must be propagated back to the corresponding software views and eventually to the main software model and all its other views. The propagation of changes could take place in the opposite direction too, from the software to the analysis model.

Another issue specific to this objective is the co-evolution of heterogeneous models whose relationship is defined by a transformation, as in the case of software and analysis models. At a minimum, the support for co-evolution should automatically flag the set of model elements that should be changed in a model in order to keep it consistent with changes in a related model. Previous research considered different co-evolution cases with a smaller semantic gap: a) the co-evolution of model instances with metamodel changes [6] [7], or b) the co-evolution of a transformation with metamodel changes [11]. In both cases there are situations where designer intervention is necessary, so we expect to find something similar, i.e. only some of the changes may be propagated automatically, while others require human intervention. We will also investigate how the properties of the transformation affect the co-evolution.

B. Inter-model traceability. We will start by investigating the traceability modeling techniques proposed by Paige et al. [24][27], and will extend them with the capability of tracing NFPs. We will consider two cases: a) building the traceability model between two entire models at once, or b) incrementally updating the trace-links for small changes in the related models.

An example of evaluation case study is to use the traceability between the software and analysis model for mapping expressions that calculate/aggregate quantitative NFP measures in the analysis and software models. A good understanding of how performance

measures are calculated for different types of models and different tools is required. Even if the formulas for different results are formalism- and tool-dependent, we aim for a general approach for mapping NFP results from the analysis domain to the software domain based on inter-model traces. A second case study will consider user-defined cross-model queries that seamlessly navigate between models via inter-model trace links. A third case study will apply the traceability solutions to queries that detect the presence of performance antipatterns in a system. Such a query navigates between different ecosystem elements: a repository of antipattern specifications, different views of the software model (structure, behaviour and deployment) and performance analysis results [8].

C. Metaheuristics for multi-NFP optimization. An important challenge is to define the "design state" of the system, by selecting a few significant design, configuration and allocation variables out of a very large set of possibilities. The selected variables must have a strong impact on the NFPs, while the ones left aside should be less important. We aim to find general criteria for what is to be included in the design state space and why. Another important challenge is to find metaheuristics that reduce significantly the search space. For instance, we know from recent experience with a design space search related to performance antipatterns that bottleneck analysis reduces the search space in combination with the removal of performance antipatterns [28]. In the proposed research, we will investigate how to cast performance diagnosis results obtained from the bottleneck analysis as metaheuristics for the multi-dimensional space search, either to explicitly exclude the sub-space where performance is bad from the search, or to explicitly include the sub-space where performance is good. We will evaluate how much the strength of the bottleneck matters and how often we should repeat the bottleneck diagnosis during the search for a multi-dimensional solution.

D. Software process automation. For a successful automation of the software process tasks concerned with NFP analysis, we need to minimize first the number of human interventions. The process should wait for designer input only if it requires new information that cannot be found anywhere in the ecosystem of modeling artifacts. If the information is hidden in an artifact or the top model, then it should be retrieved by asking the right query. This means that the script automating the software process should be written in a language capable of asking queries that navigate from artifact to artifact, of launching activities (such as model transformations, model solvers or analyzers) by passing the right parameters, which may need to be assembled from different places. The research will investigate the expressive capabilities of different scripting languages for automating the software process and will extend them if necessary. For evaluation we will use different software processes as case studies, with different activities and different kind of information required. An example is the process for reducing risk by selecting appropriate security solutions [13], while at the same time taking into account other NFPs, such as performance, reliability, scalability, availability, reliability, and cost.

5. CONCLUSIONS

The proposed approach will contribute to the integration of NFP analysis techniques into the model-driven software engineering process. It aims to improve the quality of both the software products and the software process, by raising the efficiency and usability of the MDE tool support for NFP analysis.

The proposed research is aligned with the goals of a recent industrial initiative called PolarSys (polarsys.org), an Eclipse Industry Working Group created by large industry players and by tools providers to collaborate on the creation and support of Open Source tools for model-based development of embedded systems for domains such as aerospace, defense & security, energy, health care, telecommunications, transportation. Given that many software companies have adopted some forms of MDE as shown in [4][14] the following benefits to the software industry will flow from the integration:

- Improved quality of the software products, since NFP problems will be detected and solved at an early development stage. Meeting the non-functional requirements is an important and critical attribute for the quality of real-time and/or distributed applications.
- Avoid cancellation of projects because of NFP failures. Although NFP shortfalls are not often documented and publicized, it is common knowledge that many projects fail because they don't meet their non-functional requirements.
- Better productivity in the software industry by automating error-prone steps of the software process and avoiding late fixing of NFP problems. Late fixes are very time-consuming and tend to produce badly structured software, which is difficult to understand and expensive to maintain. Software engineering based on late fixes is unsystematic, costly and cannot give any early indication whether the project is on the right track.

ACKNOWLEDGMENTS

This research was partially supported by the Natural Sciences and Engineering Research Council (NSERC), through the Discovery and Strategic Projects programs.

6. REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya. 2013. Software architecture optimization methods: A systematic literature review, *IEEE Transactions on Software Engineering*, Vol 39, No. 5, pp. 658-683.

[2] Bernardi S, Merseguer J, Petriu D. 2012. Dependability modeling and analysis of software systems specified with UML. *ACM Computing Surveys*. 45(1): 1-48.

[3] Bernardi S, Merseguer J, Petriu D. 2013. *Model-Driven Dependability Assessment of Software Systems.* Springer.

[4] F. Bordeleau, 2014. Model-Based Engineering: A New Era Based on Papyrus and Open Source Tooling, *First Workshop on Open Source Software for Model Driven Engineering* (OSS4MDE'14), co-located with MODELS'2014, Valencia, Spain, September 2014.

[5] M. Brambilla, J. Cabot, M.Wimmer, 2012. *Model-Driven Software Engineering in Practice*, Morgan & Claypool.

[6] A.Cicchetti, D. Di Ruscio, R. Eramo, A. Pierantonio. 2008. Automating co-evolution in model-driven engineering, *Proceedings of 12th International IEEE Conference on Enterprise Distributed Object Computing Conference*, EDOC'08 pp. 222-231.

[7] A.Cicchetti, D. Di Ruscio, R. Eramo, A. Pierantonio, 2009. Managing Dependent Changes in Coupled Evolution, In

Proceedings of Second International Conference of Theory and Practice of Model Transformations ICMT 2009, (Richard F. Paige, Ed.), Springer LNCS Vol.5563, pp.35-51.

[8] V. Cortellessa, A. Martens, R. Reussner, C. Trubiani, 2010. A Process to Effectively Identify "Guilty" Performance Antipatterns, In *Proc. of Fundamental Approaches to Software Engineering*, Springer, LNCS Vol. 6013, pp 368-382.

[9] V. Cortellessa, A. Di Marco, P. Inverardi, 2011. *Model-based Software Performance Analysis*, Springer.

[10] J.M. Favre, T. Nguyen, 2005. Towards a Megamodel to Model Software Evolution Through Transformations, *Electronic Notes in Theoretical Computer Science*, Vol. 127, pp. 59–74.

[11] J. García, O. Diaz, and M. Azanza, 2013. Model Transformation Co-evolution: A Semi-automatic Approach, In *Proceedings of Int. Conference on Software Language Engineering* SLE 2012 (K. Czarnecki and G. Hedin, Eds.), Springer, LNCS Vol. 7745, pp. 144–163.

[12] S. S. Gokhale. 2007. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. on Dependable and Secure Computing*, 4(1), pp. 32-40.

[13] Houmb S, Georg G, Petriu D, Bordbar B., Ray I, Anastasakis K, and France R. 2011. Balancing Security and Performance Properties During System Architectural Design. In Mouratidis H. (Ed) *Software Engineering for Secure Systems: Industrial and Research Perspectives*. pp. 155-192. ICI Global.

[14] J. Hutchinson, J. Whittle, M. Rouncefield, S. Kristoffersen, 2011. Empirical Assessment of MDE in Industry, *Proceedings of the 33rd International Conference on Software Engineering* ICSE '11, pp. 471-480.

[15] J. Jürjens, 2005. *Secure Systems Development with UML*, ISBN: 978-3-540-00701-2. Springer.

[16] D. Kolovos, L. Rose, R. Paige, 2010.The Epsilon Book, https://eclipse.org/epsilon/doc/book/.

[17] H. Koziolek and R. Reussner.2008. A Model Transformation from the Palladio Component Model to Layered Queueing Networks, In *Proc. of Performance Evaluation: Metrics, Models and Benchmarks*, SIPEW 2008, Springer, LNCS Vol. 5119, pp. 58-78.

[18] J. de Lara, T. Levendovszky, P. J. Mosterman, H. Vangheluwe, 2008.Second International Workshop on Multi-Paradigm Modeling: Concepts and Tools, In *Models in Software Engineering*, Springer LNCS Vol. 5002, pp 237-246.

[19] Z.W. Li, C. M. Woodside, J.W. Chinneck, M. Litoiu, 2011. CloudOpt: Multi-goal optimization of application deployments across a cloud, *Proc. of Int. Conference on Network and Service Management* CNSM 2011, pp.1-9.

[20] A. Martens, H. Koziolek, S. Becker, R. Reussner, 2010. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, In *Proc. of 1st joint WOSP/SIPEW International Conference on Performance Engineering* ICPE2010, pp. 105-116.

[21] Object Management Group, 2005. UML Profile for Scheduling, Performance and Time, Version 1.1, formal/05-01-02.

[22] Object Management Group, 2009. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) Version 1.0, OMG doc. formal/2009-11-02.

[23] P. Mosterman, H. Vangheluwe, 2004. Computer Automated Multi-Paradigm Modeling: An Introduction, *Simulation*, Vol. 80, Issue 9, pp. 433-450.

[24] R. Paige, N. Drivalos, D. Kolovos, K. Fernandes, C. Power, G. Olsen, S. Zschaler, 2011. Rigorous identification and encoding of trace-links in model-driven engineering, *Software and Systems Modeling*, Volume 10, Issue 4, pp. 469-487.

[25] C.U. Smith. 1990. *Performance Engineering of Software Systems*, Addison Wesley.

[26] C.U. Smith, L.G. Williams. 2001. Software Performance AntiPatterns, *Proc. of Int. CMG Conference*, pp 797-806.

[27] M. Taromirad, N.D. Matragkas, R. Paige, 2013. Towards a Multi-Domain Model-Driven Traceability Approach, In *Proceedings of the 7th Workshop on Multi-Paradigm Modeling* co-located with MODELS'2013, Miami, Florida, pp. 27-36.

[28] Trubiani C, Di Marco A, Cortellessa V, Mani N, Petriu D. 2014. Exploring synergies between bottleneck analysis and performance antipatterns. *Proceedings of The 5th ACM/SPEC International Conference on Performance Engineering* (ICPE 2014), Dublin, Ireland, pp. 75-86.

[29] C.M.Woodside, D.C. Petriu, J. Merseguer, D. B. Petriu, M. Alhaj, 2014. Transformation challenges: from software models to performance models, *Software and Systems Modeling*, Volume 13, Issue 4, Page 1529-1552, DOI: 10.1007/s10270-013-0385-x.

[30] C.M. Woodside, J.E. Neilson, D.C. Petriu and S. Majumdar, 1995.The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software, *IEEE Tranactions on Computers*, Vol. 44, No. 1, pp. 20-34.

[31] J. Xu, 2010. Rule-based automatic software performance diagnosis and improvement, *Performance Evaluation*, Vol.67, Issue 8, pp. 585-611.