Software and Systems Modeling (2021) 20:1777–1791
https://doi.org/10.1007/s10270-021-00953-3

EXPERT VOICE

Integrating the analysis of multiple non-functional properties in model-driven engineering

Dorina C. Petriu[1]

Received: 29 June 2021 / Accepted: 26 August 2021 / Published online: 28 November 2021

Abstract
This paper discusses the progress made so far and future challenges in integrating the analysis of multiple Non-Functional Properties (NFP) (such as performance, schedulability, reliability, availability, scalability, security, safety, and maintainability) into the Model-Driven Engineering (MDE) process. The goal is to guide the design choices from an early stage and to ensure that the system under construction will meet its non-functional requirements. The evaluation of the NFPs considered in this paper uses various kinds of NFP analysis models (also known as quality models) based on existent formalisms and tools developed over the years. Examples are queueing networks, stochastic Petri nets, stochastic process algebras, Markov chains, fault trees, probabilistic time automata, etc. In the MDE context, these models are automatically derived by model transformations from the software models built for development. 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, which cannot be exhaustively explored. Another challenge in balancing the NFPs of a system under construction is due to the fact that some NFPs are conflicting—when one gets better the other gets worse—so an appropriate software process is needed to evaluate and balance all the non-functional requirements. The integration approach discussed in this paper is based on an ecosystem of inter-related heterogeneous modeling artifacts intended to support the following features: feedback of analysis results, consistent co-evolution of the software and analysis models, cross-model traceability, incremental propagation of changes across models, (semi)automated software process steps, and metaheuristics for reducing the design space size to be explored.

Keywords Non-functional properties · Model-driven engineering · Analysis integration

1 Introduction

Model-Driven Development (MDD) is a paradigm that makes systematic use of models as the primary artifacts of the software development process [11]. MDD utilizes *abstraction* to separate the software model from the underlying platform models, and *automation* to generate code from models. Moreover, the focus on models facilitates the analysis of Non-Functional Properties (NFPs) of the software under development (e.g., performance, reliability, availability, scalability, security, safety, maintainability, etc.) with the help of different formal NFP analysis models (e.g., queueing networks, stochastic Petri nets, Markov chains, stochastic process algebra, probabilistic time automata, fault trees, etc.) By definition, an NFP of a software system is a constraint on the way in which the system implements and delivers its functionality. NFPs are used to express Non-Functional Requirements (NFR), which represent the quality constraints that the system must satisfy according to the project contract. Using formal models for NFP analysis brings more "engineering" into the software development, leading to the paradigm known as Model-Driven Engineering (MDE). MDE is a systematization of MDD to all software engineering activities [11, 4], including the NFP analysis based on formal models. This paper discusses the progress made so far, as well as future challenges of integrating the model-based analysis of NFPs into the MDE process.

Figure 1 illustrates the software development process steps that integrate the analysis of a NFP (such as performance [40, 49] or dependability [9]) into the MDE process: (a) edit the software development model and select the views

✉ Dorina C. Petriu
petriu@sce.carleton.ca

1 Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada

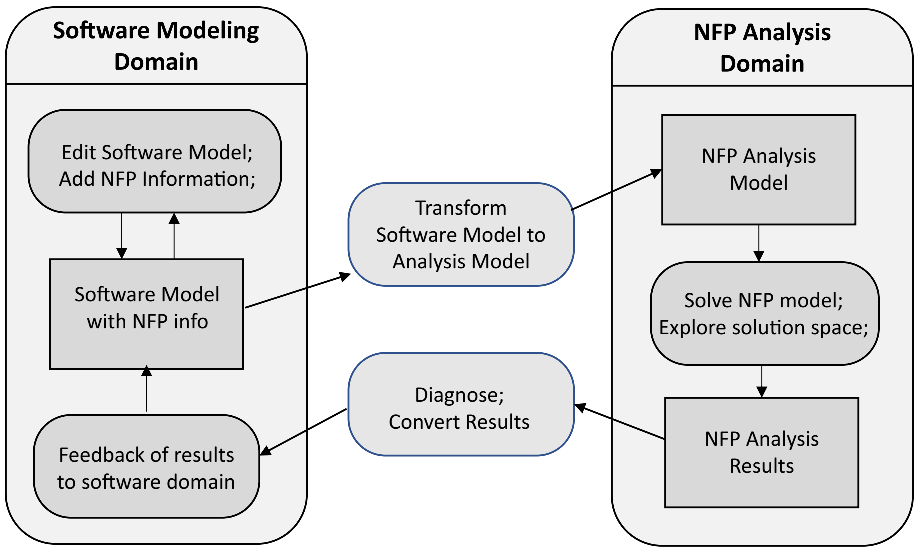
Software Modeling Domain
Edit Software Model; Add NFP Information;
Software Model with NFP info
Feedback of results to software domain
Transform Software Model to Analysis Model
Diagnose; Convert Results
NFP Analysis Domain
NFP Analysis Model
Solve NFP model; Explore solution space;
NFP Analysis Results

Fig. 1 Integrating the analysis of a single NFP in the MDE process

used for NFP analysis; (b) add information specific to the property to be verified to the software model; (c) use model transformation to generate an NFP analysis model from the annotated software model; (d) analyze the NFP model using existing solvers and diagnose NFP-related problems; and (e) give feedback for improvement to designers. The steps are repeated until the NFP meets the requirements. Similar round trip sub-processes exist for the analysis of other NFPs.

In order to execute the software process shown in Fig. 1, the following tool support is required: (a) modeling language support for adding NFP-related information to the software model; (b) tool support for the *forward path* including model-to-model transformation of the annotated software model to an NFP analysis model, which is then solved with an existing solver; (c) tool support for the *backward path* including the analysis of the results, diagnosis of NFP problems and giving feedback to designers; and (d) software process steps describing the entire workflow for MDE, which integrates the analysis of multiple NFPs. Please note that the backward path may take place either in the NFP analysis domain or in the software modeling domain [5].

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 hard [1] 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. Hence metaheuristic approaches are employed when searching for good solutions, as discussed in Sect. 2.1. Another challenge is due to the fact that some NFPs are conflicting (for example, security and performance [25, 48]). 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.

The integration approach discussed here is based on an ecosystem of inter-related heterogeneous modeling artifacts, such as software and NFP analysis models and metamodels, model transformations, solvers, inter-model traceability models and metamodels, and analysis results. An important role of the ecosystem is to support consistent co-evolution of the software and NFP analysis models. The ecosystem contains a megamodel that describes the modeling artifacts included in the ecosystem, their relationships, plus other relevant metadata needed for model management [11]. The ecosystem and its megamodel help automating the software process steps that involve multiple modeling artifacts, as the megamodel 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 the necessary models and metamodels from a repository, 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 developers or their judgment/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 above, future research needs to investigate new metaheuristics approaches for reducing the size of the design space to be explored in the search for an optimal design solution.

The paper is organized as follows: Sect. 2 discusses the related work and current limitations. Section 3 takes performance as an example of NFP to be analyzed and presents examples of source model, target model and transformation from source to target. Section 4 discusses the objectives and related research questions. Section 5 presents the research challenges for each objective, and Sect. 6 concludes the paper.

2 Background

2.1 Related work

NFP analysis models As already mentioned, the emergence of MDE, which is based on abstraction and automation, enables not only the generation of code from models, but also the generation of formal analysis models for NFP verification. Over the years, many formalisms and tools for the analysis of different NFPs have been developed, as for example queueing networks, stochastic Petri nets, stochastic process algebras, Markov chains, fault trees, probabilistic time automata, formal logic, etc. The research challenge is to bridge the gap between MDE and existing NFP analysis formalisms and tools rather than to "reinvent the wheel."

For some NFPs (for instance performance) a variety of analysis models can be used. A performance model is an abstract representation of a real system that captures its performance properties—mostly related to the quantitative use of resources during run-time behavior—and can reproduce its performance. The model can be used to study the performance impact of different design and/or configuration alternatives under different workloads, leading to advice for improving the system. Performance evaluation of a model may be done either by solving a set of equations by some analytical (possibly numerical) methods or by simulating the model and collecting statistical results. Analytical performance models are usually based on underlying stochastic models, which are often assumed to be Markov processes. A Markov process is a stochastic process with discrete state space, where all information about the future evolution of the process is contained in the present state, and not on the path followed to reach this state. Markov models suffer from a problem known as state space explosion, whereby its number of states grows combinatorially with the model size. This may introduce severe limitations in the size of performance models that can be solved. Examples of analytical performance models used for software are queueing networks, stochastic Petri nets, stochastic automata networks, and stochastic process algebra.

Queueing Networks (QN) [32] are well-known performance models, good at capturing the contention for resources. QN have been successfully applied in previous work to the performance analysis of computer-based systems, software systems, cyber-physical systems [8, 17, 38]. Efficient analytical solutions exist for a class of QN (*separable* or *product-form* QN), which make it possible to derive steady-state performance measures without resorting to building the underlying state space. The advantage is that the solution is faster and larger models can be solved. The disadvantages consist in restrictions on model assumptions (e.g., service time distributions, arrival process, scheduling policies). Similar to the approach for product-form QN, approximate solutions have been developed for non-separable QN. There are many extensions to QN in the literature. One of them, Layered Queueing Networks, will be used as target model in this paper (see Sect. 3.3).

Stochastic Petri Nets (SPN) [1] are good flow models able to represent concurrency, but not as good at representing resource contention and especially queueing policies. Efficient solutions exist only for a limited class of SPN; the most interesting models are solved with Markov chain-based solutions.

Stochastic Automata Networks [41] are composed of modular communicating automata synchronized by shared events and executing actions with random execution times. The main disadvantage is the state space explosion of its Markovian solution.

Stochastic Process Algebra introduced in [24], takes a compositional approach by breaking the system into smaller subsystems, easier to model. This approach is based on an enhanced process algebra, Performance Evaluation Process Algebra (PEPA). The compositional nature of the language provides benefits for model solution as well as model construction. The solution is based on the underlying Markov process.

Source models NFP analysis models are derived from source software models (or selected views thereof) annotated with information specific to the property to be verified. How to add such annotations is a question already addressed by many researchers and practitioners, e.g., [9, 40, 49]. In the UML world, this kind of annotations are done via UML profiles, a standard extension mechanism supported by UML editors. Profiles are domain-specific interpretations of UML and therefore can be seen as Domain-Specific Languages defined by extending or restricting UML [11]. The Object Management Group has adopted two standard profiles for performance, schedulability, and time annotations: an earlier *UML Profile for Schedulability, Performance, and Time (SPT)* defined for UML 1.x [36], and a later replacement *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)* for UML 2.x [37]. The adoption of SPT and MARTE laid the groundwork for research on the automatic generation of different kinds of performance models from annotated UML [8, 17].

In the case of security analysis, a UML profile named UMLsec defined in [27] is used to model and systematically verify the correctness of security protocols. In [25], the Alloy Analyzer tool is used to analyze security assertions of models expressed in UML and OCL, which are automatically transformed into the Alloy language, a fully declarative first-order logic language designed to model complex systems. Another approach to security analysis in component-based software architecture models [42] uses threat specification, detection, and treatment following two levels of abstraction: (a) logical specification of threats using first-order and modal logic as a technology-independent formalism; (b) formalization and verification using Alloy [42].

In the case of dependability and its many attributes (e.g., availability, reliability, fault tolerance, safety, maintainability) there is a body of work surveyed in [9], where different solutions proposed in the 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 [22]. A specialization to the MARTE profile called Dependability Analysis Model (DAM) profile for dependability analysis is presented in [10].

Software development processes Many software development processes (also known as software development methodologies) have been defined in software engineering. Some of them have been adapted to the model-driven paradigm. This paper is interested in a subset of MDE 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 [45], risk reduction-based process in [25], performance antipattern-based process in [44, 47, 6], quality-aware MDE methodology DICE for data-intensive cloud applications in [13] and fault-tree analysis of safety–critical systems in [7].

Design space exploration 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 [2]. 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 [33] for the optimization of application deployments across a cloud, based on the use of a Layered Queueing Network (LQN) model [50, 19]. 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 given in [34], 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 [29], 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 [25]. 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 [51] 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 (e.g., by removing bottlenecks). The disadvantage is that the recommendations for change are given at the LQN level and are not automatically propagated to the software domain. Another methodology for data-intensive cloud systems called DICE [13] relies on UML metamodels annotated with information about data, data processing, and data transfers. The DICE quality analysis tool chain covers simulation, verification, and architectural optimization. These tools are coupled with feedback analysis methods to help the developer iteratively improve the application design, based on monitoring data obtained from test or production environments.

Ecosystems of models There are two existing directions of research relevant to ecosystems of models: global model management [11, 18], and multi-paradigm modeling [31, 35]. Both consider a system of inter-dependent heterogeneous models, described by a top-level "model of models" (named megamodel in [11, 18]) 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 behavioral 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 behavioral views representing key performance scenarios; all views are annotated with performance information using MARTE [49]. After the performance analysis, recommended 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 NFP analysis integration in MDE addresses several related work limitations listed below:

- There is less related work in the literature on automating the backward path (including the NFP result analysis, diagnosis, and feedback to developers) than the forward path (including the transformation of software models into analysis models).
- The software and analysis models are defined separately in different languages, and cross-model queries and constraints are not usually supported.
- Normally, there is no traceability support between the software and analysis models, which makes it difficult to automate the import of analysis results in the software model context.
- Often there is no support for (semi)automatic co-evolution of the software and analysis models, so the co-evolution must be done manually.
- Usually there is no support for incremental propagation of changes between the software and analysis models.
- Many software process steps are performed manually, which makes the whole process inefficient and error prone.
- The design of the software under construction and its underlying platforms have many degrees of freedom, spanning a very large discontinuous design space, whose NFPs cannot be exhaustively explored. New metaheuristic approaches are necessary for reducing the size of the design space to be explored in the search for an optimal design solution.

3 Performance analysis integrated to MDE

3.1 Source model: UML model with performance annotations

This section presents performance analysis as an example of integration of NFP analysis into the MDE process. First, we look at the domain model for performance analysis in order to understand what kind of performance information needs to be added to UML software models. Performance is determined by how the system behavior uses system resources. Scenarios define execution paths with externally visible end points. Quality of Service (QoS) requirements (such as response time, throughput, utilization, and probability of meeting deadlines) can be placed on scenarios. In the SPT and MARTE profiles, the performance domain model describes three main types of concepts: *resources*, *scenarios*, and *workloads* [36, 37]. The resources used in the system are represented in UML structural views (e.g., in class, component or deployment diagrams); resources can be active or passive, logical or physical, software or hardware. Some of these resources belong to the software itself (e.g., critical section, software server, lock, buffer), others to the underlying platforms (e.g., process, thread, processor, disk, communication network).

A scenario is composed from scenario steps joined by predecessor–successor relationships, which may include fork/join, branch/merge and loops. A step may represent an elementary operation or a whole sub-scenario. Scenarios are represented in UML in behavioral views (e.g., interaction diagrams, statecharts or activity diagrams). Quantitative resource demands for each step must be given in the performance annotations. Each scenario is executed by a workload, which may be open (i.e., requests arriving in some predetermined pattern) or closed (a given number of users or jobs).

In this section we use an e-commerce system model from the literature [17] to show how performance is analyzed. The e-commerce source model contains a deployment diagram and three activity diagrams representing performance-critical scenarios selected for performance analysis. Figure 2 represents an annotated deployment diagram, showing the software components of the run-time architecture and their allocation to hardware processing nodes. The system has three classes of customers with a population of \$N1, \$N2 and \$N3 users, respectively, running the scenarios *BrowseCatalog*, *BrowseCart*, and *PlaceOrder,* shown in Figs. 3a, b and c.

The deployment diagram in Fig. 2 contains a set of UML nodes stereotyped as *«device»* and *«gaExecHost»* that represent physical computational resources with processing capability, and a set of software components stereotyped as *«artifact»*, each deployed on a device. An artifact may be also stereotyped as *«scheduler»*, which represents a kind of *ResourceBroker* that creates access to its brokered *ProcessingResource* or resources following a certain scheduling policy [37].

Each activity diagram shown above represents a scenario that models the interactions between software components. The behavior of each participating component is modeled inside an *ActivityPartition* (also known as a *swimlane*) which bears the name of the component executing the actions. A partition contains different types of action nodes and control nodes linked together by edges. Examples of types of UML action nodes are: (a) *AcceptEventAction*: executed when a certain event (e.g., the arrival of a call) has been triggered; (b) *SendSignalAction*: responsible for creating and transmitting signal instances to a target object; (c)*CallOperationAction*: sends a message representing an operation call request to a target object and waits until a reply is received; and (d)

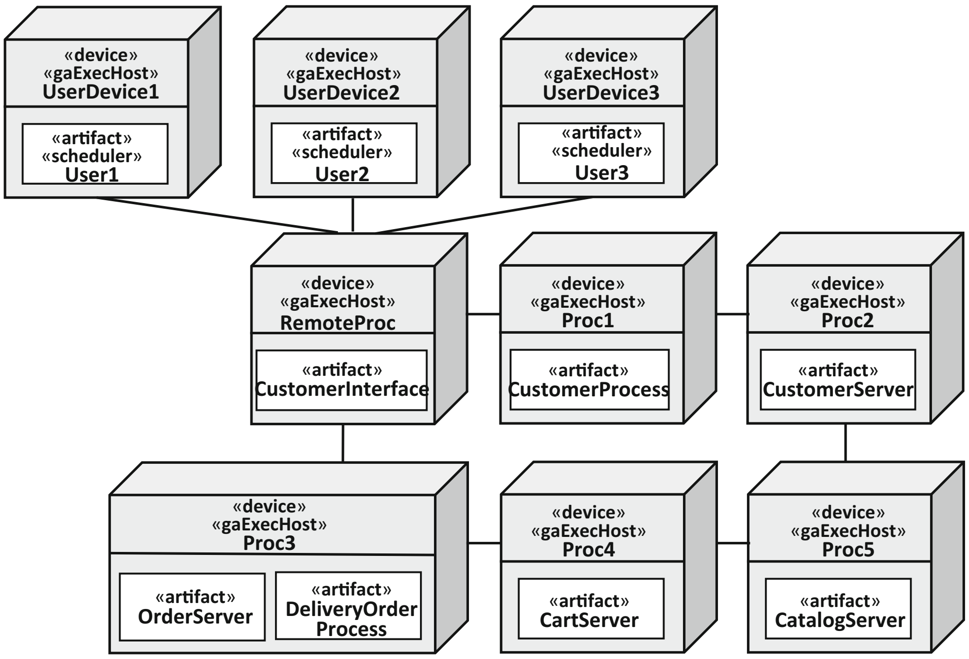
«device» «gaExecHost» UserDevice1
«artifact» «scheduler» User1
«device» «gaExecHost» UserDevice2
«artifact» «scheduler» User2
«device» «gaExecHost» UserDevice3
«artifact» «scheduler» User3
«device» «gaExecHost» RemoteProc
«artifact» CustomerInterface
«device» «gaExecHost» Proc1
«artifact» CustomerProcess
«device» «gaExecHost» Proc2
«artifact» CustomerServer
«device» «gaExecHost» Proc3
«artifact» OrderServer
«artifact» DeliveryOrder Process
«device» «gaExecHost» Proc4
«artifact» CartServer
«device» «gaExecHost» Proc5
«artifact» CatalogServer

Fig. 2 E-commerce system: deployment diagram

Opaque Action: a type of UML abstract class considered as an executable node included within the behavior. The MARTE stereotype *«PaStep»* applied to action nodes represents scenario steps, each having the following performance-related attributes: *hostdemand* (required CPU execution time), *prob* (probability of an optional step) and *rep* (number of step repetitions) [37, 43].

The control nodes are responsible for the flow of tokens between other nodes. Examples of control nodes are the *InitialNode,* which indicates the starting point of a scenario, and the *FinalNode,* which indicates the termination point. *ForkNode*, *JoinNode*, *MergeNode*, and *DecisionNode* are other examples of control nodes. Other model element type used in activity diagrams is *ControlFlow*, an activity edge responsible for passing tokens from its source to its destination node. The activity edges interconnect activity nodes to form a graph that represents the behavior of an activity as a sequence of subordinate units.

3.2 Target model

The target model of this transformation is the Layered Queueing Networks (LQN) [50, 19, 20], an extension of Queueing Networks, which can represent nested services (i.e., a server may also be a client to other servers). An LQN model is an acyclic graph whose nodes are either software tasks (parallelograms) or hardware devices (circles) and the arcs denote service requests. Figure 4 shows the LQN model generated for the e-commerce system, and Fig. 5 shows the LQN metamodel.

Existing analytic LQN solvers [20] compute the steady-state performance of a system with static allocation of resources. In the cases where the resources are dynamically allocated on demand at run-time, the steady-state solution for each configuration of interest must be computed separately.

The tasks with outgoing but no incoming arcs play the role of clients (also called reference tasks). The intermediate nodes with both incoming and outgoing arcs are usually software servers and the leaf nodes are hardware servers. A software or hardware server node can be either a single-server or a multi-server from a queueing network perspective.

Software tasks have entries corresponding to different offered services (represented as smaller parallelograms inside the tasks). A server entry may be described in two ways: (a) sequence of up to three phases; or (b) activity graph. In the first case, *phase 1* is the portion of service when the client is blocked, waiting for a reply from the server. Example of entries with phases are the entry of task *Artifact1* and *Artifact2* in Fig. 6c. At the end of phase 1, the server will reply to the client, which will unblock and continue its execution. The remaining phases of the entry, if any, will be executed by the server in parallel with the client. The resource demands of a phase are execution time demands to its processor and service requests to entries of other tasks. A more detailed description of an entry can be done by an activity graph; an example is the entry of task *Artifact3* in Fig. 6c. The activities are represented by rectangles connected by precedence arcs, and form a directed graph, which may branch into parallel threads of control or may choose randomly between different branches. Just like phases, activities have execution time demands and can make service requests to other entries. The

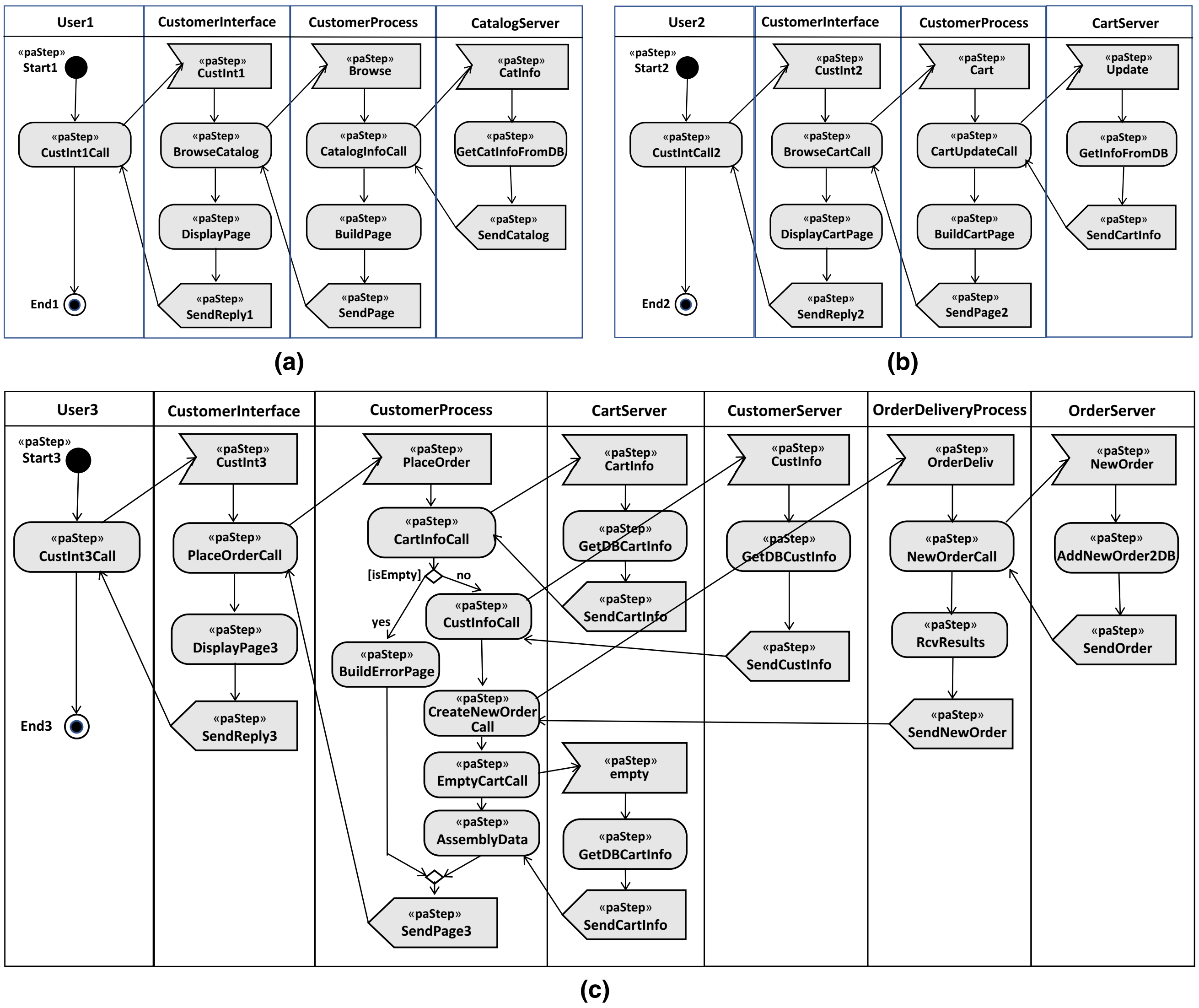

Fig. 3 **a** Activity diagram for BrowseCatalog. **b** activity diagram for BrowseCart. **c** activity diagram for PlaceOrder scenario

blue dashed line in Fig. 4 and the model elements it traverses (tasks, entries, processors) corresponds to the execution path for the *BrowseCatalog* scenario from Fig. 3a, while the other elements are generated from the remaining two scenarios.

The mapping between the model elements of the source and target models is discussed in Sect. 3.3. The LQN metamodel shown in Fig. 5 is based on the XML schema defined in [20], slightly simplified (as explained in [23]). The root model element of the LQN metamodel is *Lqnmodel,* which contains one or more *Processor* model elements. In turn, each *Processor* contains one or more *Task* elements running on it. A *Task* is composed of entries or task-activities. *Entry* is the parent of *Entry-phase-activities* model element, which is the container of *Activity*. *Activity* is the parent of children of type *Synch-call* and *Asynch-call*. *Task-activities* element contains three types: *Activity*, *Precedence,* and *Reply-entry* (which is the parent of *Reply-activity*). In addition, the elements *Pre*, *Pre-or*, *Pre-and*, *Post*, *Post-or,* and *Post-and* are all specializations of *Precedence* model element.

3.3 Model transformation

The transformation from the source model presented in Sect. 3.1 to the target model in Sect. 3.2 is implemented using the Epsilon Transformation Language (ETL) and Epsilon Object Language (EOL) from the Epsilon family [28], which contains languages specialized for different model management tasks, including model transformation, comparison, validation, etc. The transformation presented in more detail in [23] contains 17 matched rules (with optional guards) and 26 operations. The transformation engine applies each rule *R(E)* to all source element instances of type *E*. If a defined guard *g(E)* evaluates to *true* for a given *E* instance, or the guard *g(E)* is not defined, the rule will generate one or more

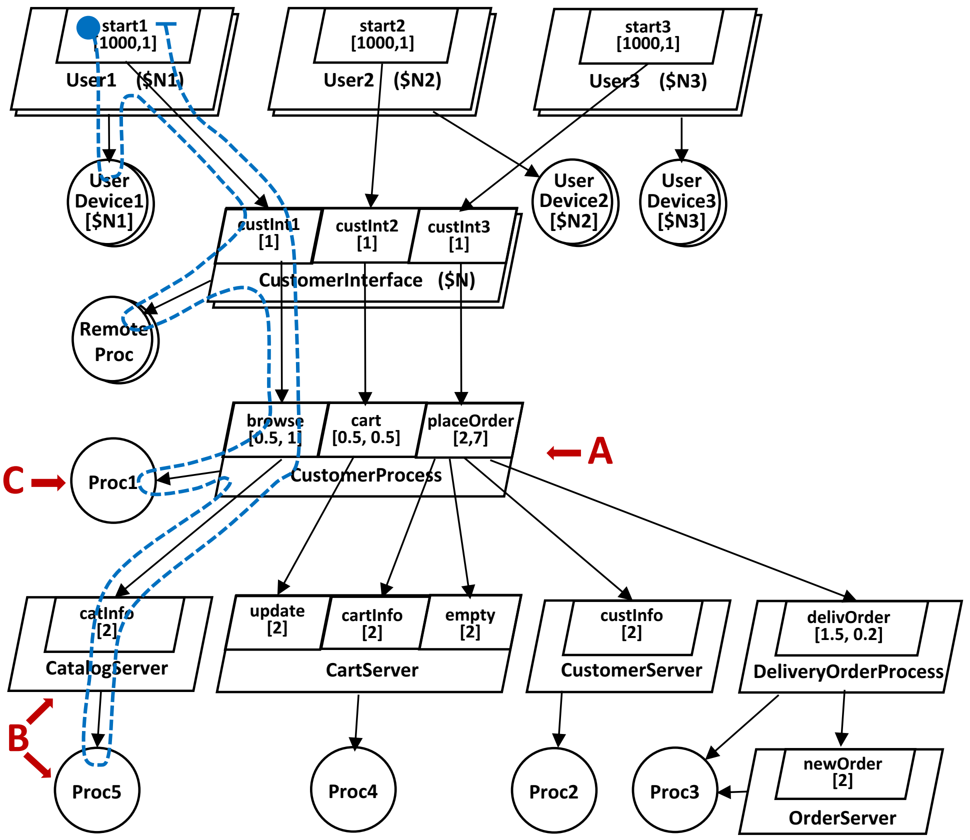

Fig. 4 LQN target model of the e-commerce system

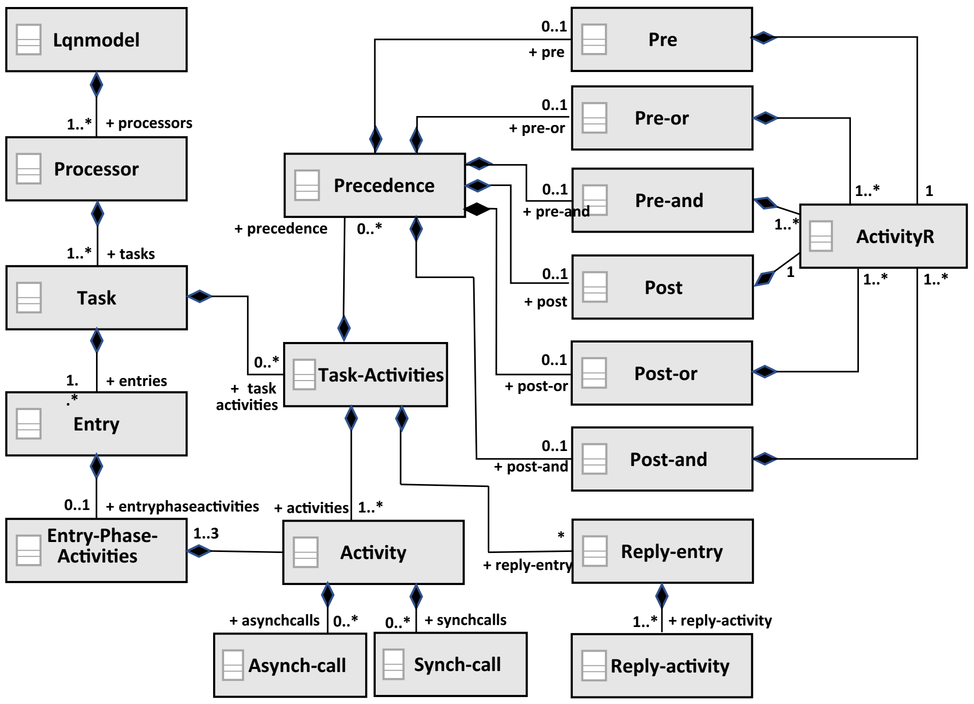

Fig. 5 LQN metamodel

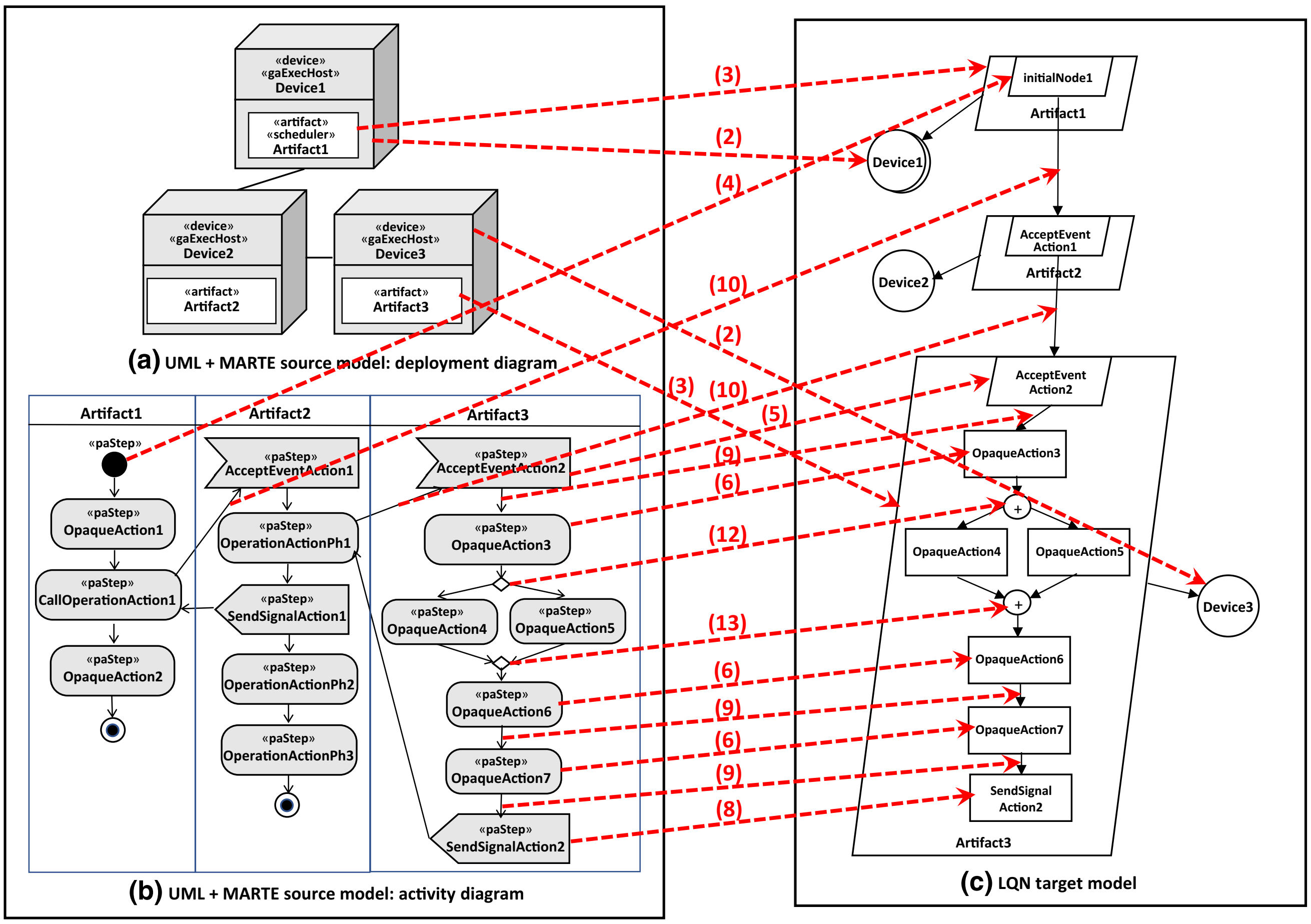

Fig. 6 High-level view of the mapping between the source and target models

target elements instances *{T1,…,Tn}* and will initialize their properties. Table 1 represents the mapping between the elements of the UML + MARTE source model and the LQN target.

Figure 6 illustrates with dotted red arrows the high-level mapping according to Table 1 between some of the elements of the source model in Figs. 6a, b, and the target model in Fig. 6c.

3.4 Performance analysis

This section presents a brief performance analysis of the e-commerce system model described in Sect. 3 (see also [3]). The source model is given in Figs. 2 and 3, and the target LQN model in Fig. 4. As mentioned in Sect. 3.1, there are three classes of users, each running a different scenario: *BrowseCatalog*, *BrowseCart,* and *PlaceOrder*. Class 1 has the largest number of users. In the experiments, a constant ratio between the numbers of users was kept: for 48 users in class 1, there is one user in each of the other two classes. For each configuration, the LQN model was solved under a variable load (from 100 to 800 users in total). Figure 7 shows

Table 1 High-level mapping between the source and target model elements

UML Model Element *E*	MARTE Stereotype	LQN Element *T*
1. Model	None	Lqnmodel
2. Device	GAExecHost	Processor
3. Artifact	Scheduler	Task
4. InitialNode	PaStep	Entry
5. AcceptEventAction	PaStep	Entry
6. OpaqueAction	PaStep	Activity
7. CallOperationAction	PaStep	Activity
8. SendSignalAction	PaStep	Activity
9. ControlFlow	None	Precedence
10. ControlFlow	None	Synch-Call
11. ControlFlow	None	Asynch-Call
12. DecisionNode	None	Precedence
13. MergeNode	None	Precedence
14. ForkNode	None	Precedence
15. JoinNode	None	Precedence

the average throughput and response time obtained with the LQN solver for a class 1 user for the four configurations described below.

A. This is the base case, where the multiplicities of all tasks and processors are 1, except for the $User_i$ tasks (each with its own processor) and the *CustomerInterface* task, which has a thread for every user. The results in Fig. 7 show that the response time and throughput curves have each a knee starting at around 200 users. The performance deteriorates very quickly after the knee, so the operating point should be selected before the knee. The analysis of task and processor utilizations shows that this is a typical case of software bottleneck, with *CustomerProcess* (indicated by a red bold letter A in Fig. 4) being the task that saturates first, limiting the concurrency level and the utilization of resources below the bottleneck. An appropriate solution is to raise the number of threads for *CustomerProcess*. This is purely a software solution, as no software resources are added.
B. The solution for case A is to raise the number of threads of *CustomerProcess* from 1 to 50. The results B in Fig. 7 show that the response time and throughput improve considerably, and the knee of the curves moves to the right, being able to accommodate more than double the number of simultaneous users before the knee than in Case A. The next bottleneck is *CatalogServer* processor, followed immediately by the *CatalogServer* process.
C. The solution for case B is to raise the number of software threads of *Catalog Server* to 50 and its processor multiplicity to 5. The performance improvement from case B to C is less important than from A to B, but still, the knee moves further to the right, accommodating at least 100 more simultaneous tasks. The next bottleneck is the processor of *CustomerProcess*.
D. After increasing the multiplicity of the *CustomerProcess* processor to 5 in case C, the response time improves further, and the knee moves beyond 800 users (where we stopped increasing the workload because the performance was satisfactory).

This simple example illustrates how the LQN performance model can be used to predict the performance effects of different configuration changes. Here the actual diagnosis of performance problems (i.e., detecting the system bottleneck) and finding a solution for alleviating the problem was done by a human analyst. In future, we intend to integrate the performance analysis with performance diagnosis algorithms that will enhance the level of automated support offered to software developers by MDE tools.

4 Objectives and research questions

The overall objective of the integration approach discussed in the paper 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 judgment 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 integration of NFP analysis into MDE has several specific objectives and related research questions, as described in the rest of the section.

4.1 Ecosystem of modeling artifacts supporting synchronized co-evolution

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 (also known as megamodel) that specifies the modeling artifacts which 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 megamodel in order to describe the various kinds 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 megamodel?

RQ-A2 How to synchronize the co-evolution of the software model and the corresponding NFP analysis models in

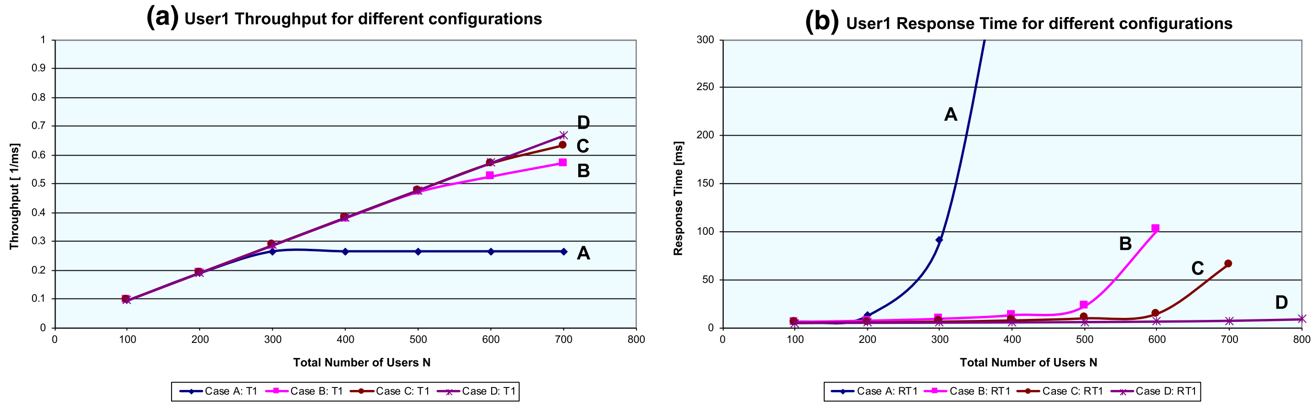

Fig. 7 Throughput and response time of the e-commerce system for different configurations

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 itself is unidirectional?

4.2 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?

4.3 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 effectively points to good design solutions obtained by removing the bottleneck.

4.4 Software process automation

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

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 for different configurations 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 megamodel 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 judgment/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?

5 Methods and 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) [11]. Also, models for different NFPs and their metamodels will be used (e.g., queueing networks, layered queueing networks, general stochastic Petri nets, stochastic reward nets, fault trees, etc.), and existing NFP solvers will be invoked as a black box.

In terms of technical space, future research will focus on open-source platforms such as Eclipse, which offers implementations of the OMG standards to be used: UML and MARTE, SysML, 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, transformation languages which can express cross-model constraints and queries, such as the family of languages Epsilon [28] developed over Eclipse, needs to be considered.

All the objectives described in Sect. 4 include the evaluation of the methods and techniques that will be developed. Appropriate case studies will be selected to verify how effectively the proposed methods work, and to identify their advantages and limitations. Below are discussed approaches specific to each objective.

A. Ecosystem of modeling artifacts

A software model may have many views representing different structural and behavioral 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 behavioral views representing key performance scenarios; all views are annotated with performance information using MARTE [37, 43]. 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 work investigated different co-evolution cases with a smaller semantic gap: (a) the co-evolution of model instances with metamodel changes [14, 15], or (b) the co-evolution of a transformation with metamodel changes [21]. In both cases, there are situations where automated co-evolution is impossible without developer intervention, so we expect to find something similar (i.e., some of the changes may be propagated automatically, while others require human decision). Other related work refers to consistent change propagation, where changes to one part of a model may affect other parts of the same model and/or even other models. The approach in [30] uses model state trees to determine possible propagation strategies (i.e., repair sequences) to pass on the changes until a consistent model state is reached. Another aspect to investigate is how the properties of the transformation affect the co-evolution.

B. Inter-model traceability

A starting point is to investigate the traceability modeling techniques such those proposed in [39, 46], and to extend them with the capability of tracing NFPs. There are at least two cases to consider: (a) building the traceability model between two whole models at once; and (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 NFP 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. Another 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, behavior and deployment) and performance analysis results [16].

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. The aim is 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 significantly reduce 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 [47]. Future research needs to 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. An aspect to consider is 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 megamodel, 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. Future research needs to investigate the expressive capabilities of different scripting languages for automating the software process and to extend them if necessary. Different software processes will be used as case studies for evaluation, with different activities and various kind of information required. An example is the process for reducing risk by selecting appropriate security solutions [25], while at the same time taking into account other NFPs, such as performance, reliability, scalability, availability, reliability, and cost.

6 Conclusions

In general, experience in conducting model-driven performance analysis and other non-NFPs in the context of MDE shows that the domain is still facing several challenges:

- *Human qualifications*. Software developers are not trained in all the formalisms used for the analysis of performance and other kind of NFPs, which leads to the idea of hiding the analysis details from developers. However, the software models must be annotated with extra information for each NFP, and the analysis results must be interpreted in order to improve the designs. A better balance needs to be found between what to be hidden and what to be exposed to the software developers.
- *Consistent model evolution.* The analysis of different NFPs may require source models at different levels of abstraction/detail. The challenge is to keep all the models/views consistent while making changes to improve a given NFP.
- *Tool interoperability.* Experience shows that it is difficult to interface and to integrate seamlessly different MDE tools and NFP analysis tools, which were created at different times with different purposes and maybe running on different platforms or platform versions.
- *Software process.* Integrating the analysis of different NFP raises process issues. For each NFP it is necessary to explore the state space for different design alternatives, configurations, workload parameters in order to diagnose problems and decide on improvement solutions. The challenge is how to compare different solution alternatives that may improve some NFPs and deteriorate others, and how to decide on trade-offs.
- *Propagation of changes through the model chain.* Currently, every time the software design changes, a new analysis model is derived in order to redo the analysis. The challenge is to develop incremental transformation methods for keeping different model consistent, instead of starting from scratch after every model improvement.

The challenges of integrating multiple NFP analysis techniques into the model-driven software engineering process span all the categories of grand challenges in MDE identified in [12]: *technical challenges* (split into *foundation, domain*, and *tool* challenges), and *social* and *community* challenges.

Future research in this area 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. Given that many software companies have adopted some forms of MDE as shown in [26], 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 resolved 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 [4].
- Avoid cancelation 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 do not 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.

References

1. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with generalized stochastic petri nets. In: Wiley Series in Parallel Computing. John Wiley and Sons, Hoboken (1995)
2. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: a systematic literature review. IEEE Trans. Softw. Eng. **39**(5), 658–683 (2013)
3. Altamimi, T., Hasanzadeh Zargari, M., Petriu, D.C.: Performance analysis roundtrip: automatic generation of performance models and results feedback using cross-model trace links. In: Proceedings of CASCON'2016 (2016)
4. Ameller, D., et al.: Dealing with non-functional requirements in model-driven development: a survey. IEEE Trans. Softw. Eng. (2019). https://doi.org/10.1109/TSE.2019.2904476
5. Arcelli, D., Cortellessa, V.: Software model refactoring based on performance analysis: better working on software or performance side?. In: Proceedings of Formal Engineering Approaches to Software Components and Architectures (2013)
6. Avritzer, A., Britto, R., Trubiani, C., Russo, B., Janes, A., Camilli M., van Hoorn, A., Heinrich R., Rapp, M., Henß, J.: A Multivariate characterization and detection of software performance antipatterns. In: Proceedings of the 2021 ACM/SPEC Int. Conference on Performance Engineering (ICPE '21) (2021)
7. Al Shboul, B., Petriu, D.C.: Automatic derivation of fault tree models from sysml models for safety analysis. J. Softw. Eng. Appl. (JSEA) **11**(5), 204–222 (2018)
8. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. IEEE Trans. Softw. Eng. **30**(5), 295–310 (2004)
9. Bernardi, S., Merseguer, J., Petriu, D.C.: Dependability modeling and analysis of software systems specified with UML. ACM Comput. Surv. **45**(1), 1–48 (2012)
10. Bernardi, S., Merseguer, J., Petriu, D.C.: A dependability profile within MARTE. In: Software and Systems Modeling (SoSyM), pp. 313–336. Springer, New York (2011)
11. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, Synthesis Lectures on Software Engineering series. Morgan & Claypool Publishers, California (2012)
12. Bucchiarone, A., Cabot, J., Paige, R., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**, 5–13 (2020)
13. Casale, G., Ardagna, D., Artac, M., Barbier, F., Di Nitto, E., Henry, A., Iuhasz, G., Joubert, C., Merseguer, J., Munteanu, V.I., Pérez, J. F., Petcu, D., Rossi, M., Sheridan, C., Spais, I., Vladusic, D.: DICE: quality-driven development of data-intensive cloud applications. In: IEEE/ACM 7th International Workshop on Modeling in Software Engineering, MiSE'2015 (2015)
14. Cicchetti, D., Di Ruscio, R. Eramo, A. Pierantonio, A.: Automating co-evolution in model-driven engineering. In: Proceedings of 12th International IEEE Conference on Enterprise Distributed Object Computing Conference, EDOC'08, pp. 222–231 (2008)
15. Cicchetti, D., Di Ruscio, R. Eramo, A. Pierantonio, A.: Managing dependent changes in coupled evolution. In:Paige, R.F. (Ed.), Proceedings of 2nd International Conference of Theory and Practice of Model Transformations ICMT 2009, LNCS Vol. 5563, pp. 35–51. Springer, New York (2009)
16. Cortellessa. V, Martens, A., Reussner, R., Trubiani, C.: A process to effectively identify "Guilty" Performance Antipatterns, In: Proc. of Fundamental Approaches to Software Engineering, LNCS Vol. 6013, pp 368–382. Springer, New York (2010)
17. Cortellessa, V., Di Marco, A., Inverardi, P.: Model-Based Software Performance Analysis. Springer, New York (2011)
18. Favre, J.M., Nguyen, T.: Towards a megamodel to model software evolution through transformations. Electron. Theor. Comput. Sci. **127**, 59–74 (2005)
19. Franks, R.G., Al-Omari, T., Woodside, C.M., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE Trans. Softw. Eng. **35**(2), 148–161 (2009)
20. Franks, R.G., Maly, P., Woodside, C.M., Petriu, D.C., Hubbard, A., Mroz, M.: Layered queueing network solver and simulator user manual. Department of Systems and Computer Engineering, Carleton University, Ottawa ON, Canada, https://www.sce.carleton.ca/rads/lqns/ (2013)
21. García, J., Diaz, O., Azanza, M.: Model transformation co-evolution: a semi-automatic approach. In: Czarnecki, K., Hedin, G. (Eds.), Proceedings of Int. Conference on Software Language Engineering SLE 2012, LNCS Vol. 7745, pp. 144–163. Springer, New York (2013)
22. Gokhale, S.S.: Architecture-based software reliability analysis: overview and limitations. IEEE Trans. Depend. Secure Comput. **4**(1), 32–40 (2007)
23. Hassanzadeh Zargari, M.: Automatic derivation of LQN performance models from UML software models using epsilon, master thesis, department of systems and computer engineering, Carleton University, Ottawa ON, Canada (2016)
24. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
25. Houmb, S., Georg, G., Petriu, D.C, Bordbar, B., Ray, I., Anastasakis, K., France, R.: Balancing security and performance prop-

erties during system architectural design. In: Mouratidis H. (Ed) *Software Engineering for Secure Systems: Industrial and Research Perspectives,* pp. 155–192. ICI Global (2011)
26. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: Proceedings of the 33rd International Conference on Software Engineering ICSE '11, pp. 471–480. (2011)
27. Jürjens, J.: Secure systems development with UML. Springer, New York (2005)
28. Kolovos, D., Rose, L., García-Domínguez, A., Paige, R., *The Epsilon Book*, https://eclipse.org/epsilon/doc/book/EpsilonBook.pdf (2018)
29. Koziolek, H., Reussner, R.: A model transformation from the palladio component model to layered queueing networks. In: Proc. of Performance Evaluation: Metrics, Models and Benchmarks, SIPEW 2008, LNCS Vol. 5119, pp. 58–78. Springer (2008)
30. Kretschmer, R., Khelladi, D.E., Lopez-Herrejon, R.E., Egyed, A.: Consistent change propagation within models. Softw. Syst. Model. **20**(3), 539–555 (2021)
31. de Lara, J., Levendovszky, T., Mosterman, P.J., Vangheluwe, H.: Second international workshop on multi-paradigm modeling: concepts and tools. In: Models in Software Engineering, pp. 237–246. Springer, New York (2008)
32. Lazowska, E., Zahorjan, J., Scott Graham, G., Sevcik, K.S.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice Hall, Hoboken (1984)
33. Li, Z.W., Woodside, C. M., Chinneck, J.W., Litoiu, M.: "CloudOpt: Multi-goal optimization of application deployments across a cloud. In: Proc. of Int. Conference on Network and Service Management CNSM 2011, pp.1–9. (2011)
34. Martens, A., Koziolek, H., Becker, S., Reussner, R.: 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 ICPE'2010, pp. 105–116. (2010)
35. Mosterman, P., Vangheluwe, H.: Computer automated multi-paradigm modeling: an introduction. SIMULATION **80**(9), 433–450 (2004)
36. Object Management Group: *UML Profile for Scheduling, Performance and Time*, Version 1.1, formal/05-01-02 (2005)
37. Object Management Group: *UML Profile for Modeling and Analysis of Real-Time and Embedded systems* (MARTE) Version 1.0, OMG doc. formal/2009-11-02 (2009)
38. Pagliari, L., D'Angelo, M., Caporuscio, M., Mirandola, R., Trubiani, C.: To what extent formal methods are applicable for performance analysis of smart cyber-physical systems?. In: Proceedings of the 13th European Conference on Software Architecture (2019)
39. Paige, R., Drivalos, N., Kolovos, D., Fernandes, K., Power, C., Olsen, G., Zschaler, S.: Rigorous identification and encoding of trace-links in model-driven engineering. Softw. Syst. Model. **10**(4), 469–487 (2011)
40. Petriu, D.C., Alhaj, M., R. Tawhid, R.: Software performance modeling. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (Eds.) *SFM 2012: Formal Methods for MDE*, LNCS 7320, pp. 219–262. Springer (2012)
41. Plateau, B., Atif, K.: Stochastic automata network of modeling parallel systems. IEEE Trans. Softw. Eng. **17**(10), 1093–1108 (1991)
42. Rouland, Q., Hamid, B., Jaskolka, J.: Specification, detection, and treatment of STRIDE threats for software components: modeling, formal methods, and tool support. J. Syst. Archit. **117**, 102073 (2021)
43. Selic, B., Gérard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems. Morgan Kauffmann Publisher, Burlington (2013)
44. Smith, C.U., Williams, L.G.: Software performance anti-patterns. In: Proc. of Int. CMG Conference, CMG'2001, pp 797–806 (2001)
45. Smith, C.U., Williams, L.G.: Performance solutions: a practical guide to creating responsible, scalable Software. Addison-Wesley, Boston (2002)
46. Taromirad, M., Matragkas, N.D., Paige, R.: Towards a multi-domain model-driven traceability approach. In: Proceedings of the 7th Workshop on Multi-Paradigm Modeling co-located with MODELS'2013, pp. 27–36. Miami, Florida (2013)
47. Trubiani C., Di Marco A., Cortellessa V., Mani N., Petriu D.C.: Exploring synergies between bottleneck analysis and performance antipatterns. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014), pp. 75–86. Dublin, Ireland (2014)
48. Woodside, C.M., Petriu, D.C., Petriu, D.B., Xu, J., Israr, T., Georg, G., France, R., Bieman, J.M., Houmb, S., Juerjens, J.: Performance analysis of security aspects by weaving scenarios extracted from UML models. J. Syst. Softw. **82**, 56–74 (2011)
49. Woodside, C.M., Petriu, D.C., Merseguer, J., Petriu, D.B., Alhaj, M.: Transformation challenges: from software models to performance models. Softw. Syst. Model. **13**(4), 1529–1552 (2014). https://doi.org/10.1007/s10270-013-0385-x
50. Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S.: The Stochastic rendezvous network model for performance of synchronous client-server-like distributed software. IEEE Trans. Comput. **44**(1), 20–34 (1995)
51. Xu, J.: Rule-based automatic software performance diagnosis and improvement. Perform. Eval. **67**(8), 585–611 (2010)

Dorina C. Petriu is a Distinguished Research Professor (Emeritus) in the Department of Systems and Computer Engineering at Carleton University, Ottawa, ON, Canada. Her main research interests are in the areas of software performance modeling and model-driven development, with emphasis on integrating the analysis of non-functional properties into the software development process. She was a contributor to two standard profiles adopted by the Object Management Group, which extend UML with qualitative and quantitative concepts for modeling and analyzing real-time systems. She served as program co-chair and participated in the steering and program committees of numerous international conferences and workshops. Dr. Petriu is a Fellow of the Canadian Academy of Engineering, Fellow of the Engineering Institute of Canada, Senior Member of I.E.E.E. and member of A.C.M..