Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based Architectures

Fabian Brosig, Philipp Meier, Steffen Becker, Anne Koziolek, Heiko Koziolek, Samuel Kounev

Abstract—During the last decade, researchers have proposed a number of model transformations enabling performance predictions. These transformations map performance-annotated software architecture models into stochastic models solved by means of analytical or numerical analysis or by system simulation. However, so far, a detailed quantitative evaluation of the accuracy and efficiency of different transformations is missing, making it hard to select an adequate transformation for a given context. This paper provides an in-depth comparison and quantitative evaluation of representative model transformations to, e.g., Queueing Petri Nets and Layered Queueing Networks. The semantic gaps between typical source model abstractions and the different analysis techniques are revealed. The accuracy and efficiency of each transformation are evaluated by considering four case studies representing systems of different size and complexity. The presented results and insights gained from the evaluation help software architects and performance engineers to select the appropriate transformation for a given context, thus significantly improving the usability of model transformations for performance prediction.

Index Terms—D.2.11 Software architectures; D.2.10.h Quality analysis and evaluation; D.2.2 Design tools and techniques.

1 Introduction

To ensure that a software system meets its performance requirements, the ability to predict its performance under different configurations and workloads is highly valuable throughout the system life cycle [1], [2]. During the design phase, performance prediction helps software architects to evaluate different design alternatives. At deployment time, it facilitates system sizing and capacity planning. During operation, predicting the effect of changes in the workload or in the system configuration helps avoiding performance problems such as long response times or over-utilized resources. Recent performance prediction approaches for component-based architectures often rely on performance-annotated software architecture models (e.g., UML2, AADL), which are transformed into stochastic performance models, e.g., queueing networks (QN), stochastic Petri nets (SPN), stochastic process algebra (SPA), and then solved to determine performance metrics of interest (e.g., response time or throughput) [2], [3], as illustrated in Fig. 1.

The plethora of modeling notations, transformations,

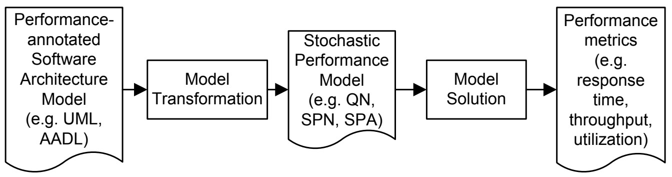
Performance-annotated Software Architecture Model (e.g. UML, AADL)
Model Transformation
Stochastic Performance Model (e.g. QN, SPN, SPA)
Model Solution
Performance metrics (e.g. response time, throughput, utilization)

Fig. 1. Architecture-based performance prediction process.

and solution tools [2], [3] make it hard to choose an appropriate performance prediction approach in a given setting. Approaches based on numerical solvers are known to be fast but often limited in expressiveness to adequately model many realistic situations. Approaches based on simulation are known to be more expressive but often have long execution times leading to high prediction overhead. The intuitively perceived trade-offs between prediction accuracy and solution efficiency in state-of-the-art performance analysis tools are currently not well understood due to the lack of in-depth quantitative evaluations and comparisons. Trade-off decisions between prediction accuracy and time-to-result are important in scenarios i) where a large problem space needs to be explored or ii) when the prediction results need to be available within a certain time window.

Existing performance prediction approaches include transformations from UML into for example layered queueing networks [4], [5], [6], stochastic well-formed nets [7], or stochastic process algebra [8]. Quantitative comparisons of model transformations and/or analysis tools can be found in [9], [10], [11]. However, Balsamo et al. [9] do not compare the respective solver execution

- *F. Brosig is with Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany. E-mail: fabian.brosig@kit.edu*
- *P. Meier is with Exxeta AG, Albert-Nestler-Strasse 11, 76131 Karlsruhe, Germany. E-mail: philstyler@googlemail.com*
- *S. Becker is with University of Paderborn, Zukunftsmeile 1, 33102 Paderborn, Germany. E-mail: steffen.becker@upb.de*
- *A. Koziolek is with Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany. E-mail: koziolek@kit.edu*
- *H. Koziolek is with ABB Corporate Research, Wallstadter Str. 59, 68526 Ladenburg, Germany. E-mail: heiko.koziolek@de.abb.com*
- *S. Kounev is with Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany. E-mail: kounev@kit.edu*

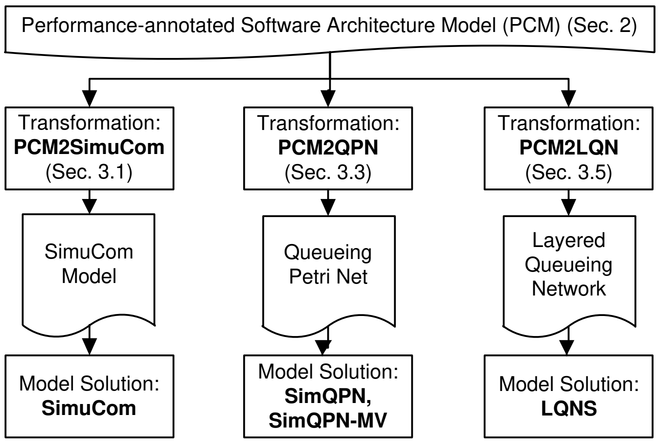
Performance-annotated Software Architecture Model (PCM) (Sec. 2)
Transformation: PCM2SimuCom (Sec. 3.1)
Transformation: PCM2QPN (Sec. 3.3)
Transformation: PCM2LQN (Sec. 3.5)
SimuCom Model
Queueing Petri Net
Layered Queueing Network
Model Solution: SimuCom
Model Solution: SimQPN, SimQPN-MV
Model Solution: LQNS

Fig. 2. Compared model transformations and performance analysis tools.

times, and the comparison conducted by Tribastone et al. [10], [11] does not involve model transformations as the stochastic performance models are modeled by hand.

We compare existing performance prediction approaches that are based on model transformations from a performance-annotated software architecture model to different analysis techniques (cf. Fig. 2). The source model complies to the Palladio Component Model (PCM) [12], a mature, domain-specific modeling language for the performance prediction of component-based software systems [3]. The comparison involves transformations to established modeling formalisms such as Queueing Petri Nets (QPNs) [13], [14] and Layered Queueing Networks (LQNs) [15], [16] as well as different model solving strategies ranging from model simulations to mean-value analysis. We provide an in-depth quantitative evaluation with regard to the trade-offs between prediction accuracy and solution efficiency. Starting with a process-based simulation (PCM2SimuCom [12]) we compare various model solving approaches (PCM2QPN [14] and PCM2LQNS [17]) that are expected to gradually improve prediction speed at the expense of prediction accuracy.

As contributions of this paper (i) we reveal the semantic gaps between typical source model abstractions and the different analysis techniques, (ii) we evaluate the effects of the identified semantic gaps on the performance prediction accuracy, (iii) we evaluate the efficiency of each transformation and respective analysis tool in the context of four representative case studies, and (iv) we consolidate the trade-offs between the performance transformations and analysis tools to provide practical guidance on deciding when to use which tool.

The considered transformations and respective model solution techniques serve as representative examples of the typical model types and solution techniques used in practice for performance prediction. The case studies were chosen to cover different modeling features as well as different types of systems in terms of size and complexity. The presented results and insights can be used to help software architects and performance engineers to select an appropriate model transformation and solution approach for a given context.

In the remainder of this paper, Section 2 first introduces typical elements of a performance-annotated software architecture model. Section 3 introduces the transformations and describes the semantic gaps between target and source model. Our evaluation and comparison of the different analysis methods is presented in Section 4 and discussed in detail in Section 5. After reviewing related work in Section 6, we conclude the paper by giving an outlook on future research activities.

2 Performance-annotated Software Architecture Model

Architecture-level performance modeling languages are powerful tools to describe the performance-influencing factors of a software system in a concise way. Since the architecture is reflected, component resource demands, performance-relevant component control flow and its dependence on service input parameters can be directly modeled [2], [3].

We summarize the main concepts of a performance-annotated software architecture model by means of the Palladio Component Model (PCM) [18]. It is a mature modeling language for model-driven quality analysis of component-based software architectures [3] and has been used in a number of industry-relevant case studies [19], [20], [21], [22], [23], [24]. Currently supported quality predictions include performance, reliability, and maintenance costs, however, in this paper the focus is on performance modeling.

The performance behavior of a component-based software system is a result of the assembled components' performance behavior. In order to capture the behavior and resource consumption of a component, the following four factors are taken into account. First, the component's implementation affects its performance. Additionally, a component may depend on external services whose performance has to be considered as well. Furthermore, both the way a component is used, i.e., its usage profile, and its execution environment are taken into consideration.

PCM models are divided into five sub-models: The repository model consists of interface and component specifications. A component specification defines which interfaces the component provides and requires. A component may be either a basic (i.e., atomic) component or a composite component. A composite component may contain several child component instances assembled with connectors that link required interfaces to provided interfaces. For each provided service, a basic component specification refers to an abstract description of the service's internal behavior denoted as `ResourceDemandingServiceEffectSpecification` (RDSEFF). The system model describes how component instances from the repository are assembled to build an entire system. The resource environment model specifies the execution environment in which the system is deployed. The allocation model describes the mapping of component instances from the system model to resource containers defined in the resource environment. The usage model describes the user behavior. It captures the services that are called, the frequency (workload intensity) and order in which they are invoked, and the input parameters passed to them.

We go into detail how PCM allows us to model service behavior with RDSEFFs. An RDSEFF captures the control flow and resource consumption of a component's service implementation depending on its input parameters passed upon invocation. The control flow is abstracted capturing only performance-relevant actions. As control flow constructs, there are branches, loops, forks with and without a synchronization barrier as well as acquire/release actions for semaphores. An

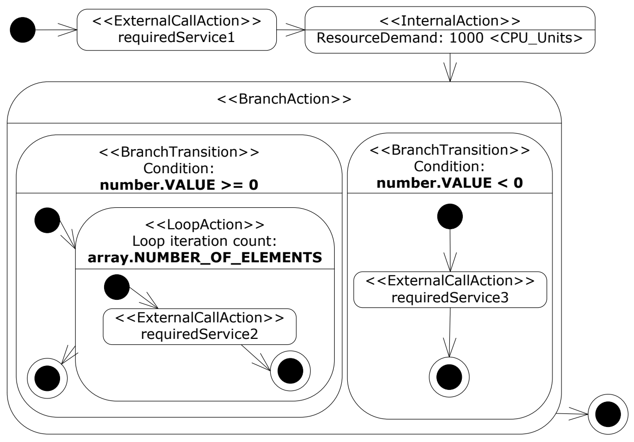

Fig. 3. RDSEFF of a service with signature `execute(int number, List array)` (cf. [18]).

example of an RDSEFF for a service `execute(int number, List array)` is shown in Fig. 3. The RDSEFF starts with an `ExternalCallAction` to a required service, followed by an `InternalAction` and a `BranchAction` with two `BranchTransitions`. The first `BranchTransition` contains a `LoopAction` whose body consists of another `ExternalCallAction`. The second `BranchTransition` contains a further `ExternalCallAction`. The performance-relevant behavior of the service is parameterized with service input parameters. Whether the first or second `BranchTransition` is called depends on the value of service input parameter `number`. This parameter dependency is specified explicitly as a branching condition. Similarly, the loop iteration count of the `LoopAction` is modeled to be equal to the number of elements of the input parameter `array`. PCM also allows to define parameter dependencies stochastically. For instance, branching conditions can be replaced with branching probabilities. Furthermore, the distribution of a loop iteration count can be described with a probability mass function (PMF), e.g., `IntPMF[(9;0.2) (10;0.5) (11;0.3)]`. Then, the loop body would be executed 9 times with a probability of 20%, 10 times with a probability of 50%, and 11 times with a probability of 30%.

Throughout this paper a simplified customer relationship system (CRM) will serve as a running example to explain both the model notations and the transformations. Fig. 4 depicts this CRM system as a PCM model in a compact UML-like notation. The components *Customer Service* and *Marketing System* offer the operations *IUser.ProvideService* and *ISales.Query*, respectively, and are deployed on separate nodes *Application Server 1* and *Application Server 2*, respectively. The third component offers two services *IDBStats.Get* and *IDBQuery.Update* and is deployed on a separate *Database Server* node. The RDSEFFs (activity diagrams) show the control flow of the respective operations and their resource demands. The processing rates of the three server nodes are shown as annotations. Finally, the system usage model is indicated by the annotated use case actors with respective workload specifications. Notice that the model is simplified for illustration purposes and neglects several PCM concepts, such as passive resources, component parameters, or composite components.

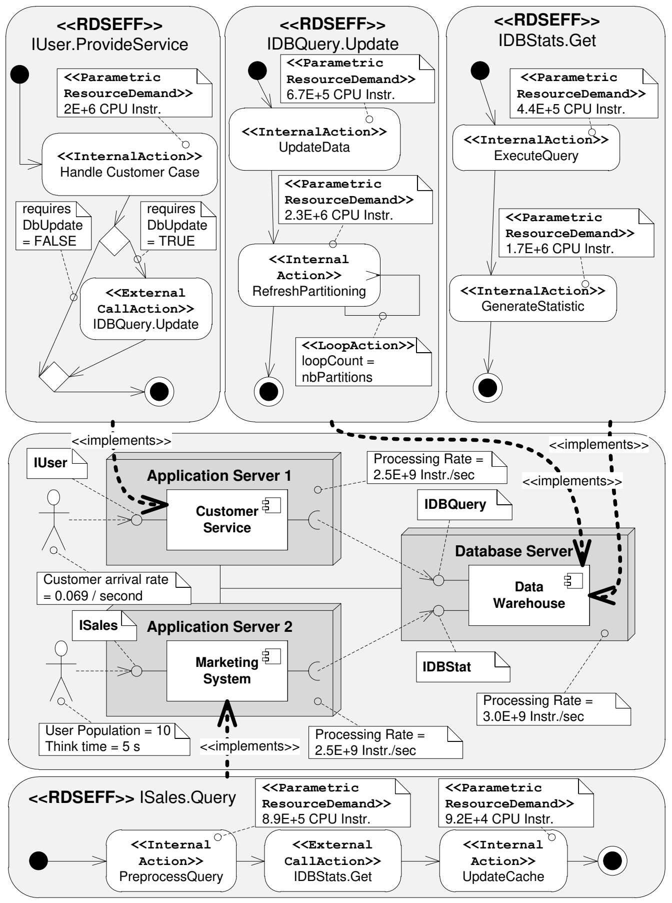

Fig. 4. PCM running example.

3 Model Transformations

This section describes different ways to derive performance predictions from a performance-annotated software architecture model. We ask for a prediction that is both *accurate* and *fast*. We denote the prediction accuracy as the degree to which the prediction reflects the model, i.e., not the model representativeness.

We present model transformations that map performance-annotated software architecture models to different analysis models solved by simulation or analytical techniques. Based on existing PCM model transformations [14], [16], [18] we describe the semantic gaps between typical source model abstractions and the different analysis approaches. Having the semantic gaps identified, we evaluate their impact on the prediction accuracy and time-to-result in Section 4.

Starting with a process-based simulation, which serves as the PCM reference solver, we present various model solving approaches that are expected to gradually improve prediction speed at the expense of prediction accuracy.

3.1 PCM2SimuCom

SimuCom [18] is PCM's reference solver, which supports the full range of PCM concepts. It is based on a transformation of the PCM instance to a process-based discrete-event simulation. The latter simulates the users specified in PCM usage model and derives performance metrics, such as response times, throughput, and utilization. Technically, PCM2SimuCom is a model-to-text transformation, which maps PCM models into Java code. This code is loaded by SimuCom and executed during a simulation run. Due to its full support of PCM features, we use SimuCom throughout this paper as reference solver.

3.2 Solving Parameter Dependencies

Modeling parameter dependencies means describing the performance-relevant behavior of a service's implementation depending on its input parameters passed upon service invocation. With common performance modeling formalisms such as queueing networks (QNs), (i) it is not possible to annotate individual requests with a payload, e.g., parameter settings, and (ii) it is not possible to recalculate model parameters such as resource demands with, e.g., arithmetic expressions on-the-fly at solving time. Parameter dependencies can thus not be directly translated to such modeling formalisms.

However, such modeling formalisms often come with mature model solving tools. Therefore, we apply a pre-processing step to allow using already existing, established solver tools. The pre-processing step aims at resolving parameter dependencies *before* model solving. Starting with the usage model, the parameter settings are propagated through the behavior descriptions. As a result of the pre-processing step we denote as *Dependency Solving* [25], the model parameter descriptions are characterized with probability distributions but independent of service input parameters.

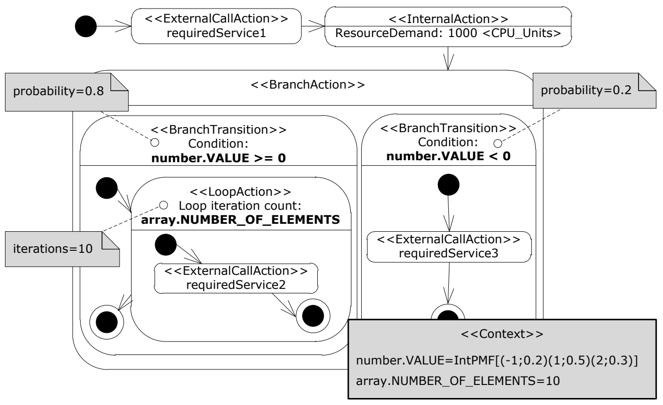

Fig. 5. Solving parameter dependencies.

To illustrate the step of dependency solving, we provide an example in Fig. 5. The figure shows the RDSEFF of Fig. 3 after the pre-processing step. Using parameter settings provided in the *Context* when calling the RDSEFF's service, branch conditions and loop iteration numbers are replaced with expressions that do not refer to parameters anymore. In the example, the two branch transitions get assigned a probability, the loop iteration number is replaced with the concrete number of array elements. Parameter dependencies may not only occur on branch probabilities or loop counts, but also on resource demands, and even propagated parameters supplied to other components. After pre-processing, the model contains concrete resource demands, branch probabilities, and loop counts. It is thus ready to be transformed as if it were a monolithic (i.e., not component-based) model. Note that the full simulation SimuCom does not require the dependency solver, because it inserts the necessary parameter values during the simulation run. This comes at the cost that the simulation must run at least as many times as there are values in the parameter range to provide reliable results, because it uses *one* concrete (randomly chosen) value in each simulation run.

Semantic Gap

The parameter propagation makes a simplification regarding dependencies between stochastic variables. Subsequent uses of the same variable are treated as they were independent but should be stochastically dependent. This leads to only partial support of parameter dependencies for the solving approaches other than SimuCom. In the evaluation in Section 4.2, we discuss an example.

3.3 PCM2QPN

While SimuCom (cf. Section 3.1) provides accurate results, the simulation is not optimized towards time-to-result of the prediction. In this section, we present

PCM2QPN, a model transformation from PCM models to Queueing Petri Nets (QPNs).

We use Queueing Petri Nets since they are a general-purpose modeling formalism that has been shown to lend itself well to modeling and analyzing the performance of distributed systems [26], [27], [28], [29]. As a combination of Queueing Networks and Colored Generalized Stochastic Petri Nets (CGSPNs), QPNs can easily model hardware contention and scheduling strategies as well as software contention, simultaneous resource possession, synchronization, blocking, and asynchronous processing. A mature and highly optimized simulation engine (SimQPN, which is part of the Queueing Petri net Modeling Environment (QPME) [30]) is available. Employing QPNs and SimQPN, compared to SimuCom, we expect a faster performance prediction to the price of only minor inaccuracies.

Queueing Petri Nets (QPNs)

Fig. 6 shows the notation used for QPNs. There are ordinary places, queueing places, transitions and so-called subnet places that contain nested QPNs. While ordinary places, transitions and transition modes correspond to the places of CGSPNs, a queueing place is a combination of a queue (service station) and a place. The behavior of a queueing place is as follows: tokens, when fired into a queueing place by any of its input transitions, are inserted into the queue according to the queue's scheduling strategy. The time tokens spend in the queue includes the time spent waiting for service and the time receiving service which depends on the queue's service time distribution. While residing in the queue, tokens are not available for output transitions of the queueing place. After completion of its service at the queue, a token is immediately moved to the depository, where it becomes available for output transitions of the place. The rest of a QPN behaves like a normal CGSPN. A more detailed treatment of the subject and formal definitions can be found in [13]. Queueing places are normally used to model system resources such as CPUs, disk drives, or network links. Ordinary places are used to implement semaphore and blocking semantics required to model, e.g., thread pools. Tokens in the QPN are used to model requests or transactions processed by the system. Tokens can be distinguished by their color. Thus, token colors are typically used to map request classes. However, note that individual tokens have no identity and the order of tokens is not defined in QPNs.

Transformation and Semantic Gaps

Transforming performance-annotated software architecture models to QPNs requires a pre-processing step to resolve parameter dependencies as described in Section 3.2. Fig. 7 shows an excerpt of the transformed running example that employs a limited number of mapping elements. Together with a specification of a closed workload, it shows how the service behavior

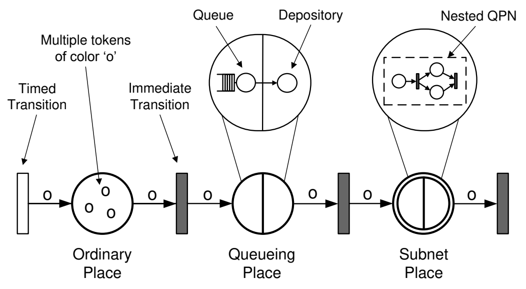

Fig. 6. QPN notation.

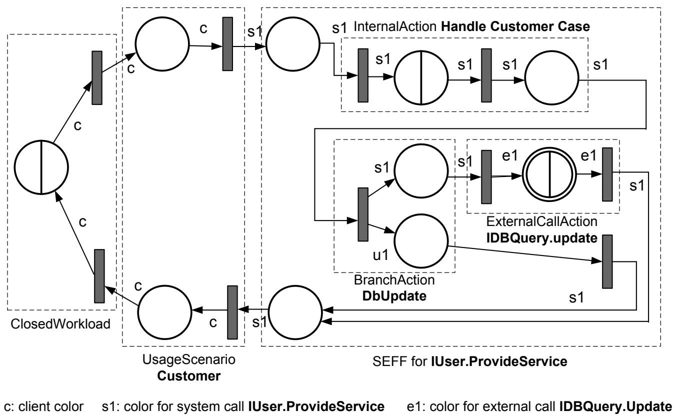
InternalAction Handle Customer Case
ExternalCallAction IDBQuery.update
BranchAction DbUpdate
ClosedWorkload
UsageScenario Customer
SEFF for IUser.ProvideService
c
s1
e1
u1
c: client color s1: color for system call IUser.ProvideService e1: color for external call IDBQuery.Update

Fig. 7. QPN result of transforming *IUser.ProvideService* of the running example.

for *IUser.ProvideService* of the running example is transformed to a QPN. The service behavior mapping for *IDBQuery.Update* referenced in one of the branches within *ProvideService* is represented by a subnet place. The branch itself is implemented by a transition with one transition mode for each branch. The transition mode is weighted with the corresponding branch probability. The example further shows that a new color is used each time a service call is traversed. Details of the mapping of PCM to QPN can be found in [14], [31]. In the following we focus on the semantic gaps between the source model and the QPN mappings on a conceptual level.

Loops: A loop action in usage scenarios or service behaviors can be described by a child behavior that is executed multiple times according to an expression that evaluates to an integer constant I or to an integer type probability mass function (IntPMF). The latter has N possible integer values V_i each with a given probability P_i. The constant case is treated as $N = 1$, $V_i = I$ and $P_i = 1.0$. The next loop iteration does not start until the previous request has completed the execution of the child behavior. The QPN representation of a loop (cf. Fig. 8) can be divided into an inner and an outer part. The outer part consists of the *Loop-Entry* and *Loop-Exit* transitions, as well as of the *Loop-Pool* and *Loop-Depository* places. The inner part consists of the *Loop-Inner-Entry* and *Loop-Inner-Exit* transitions, as well as of

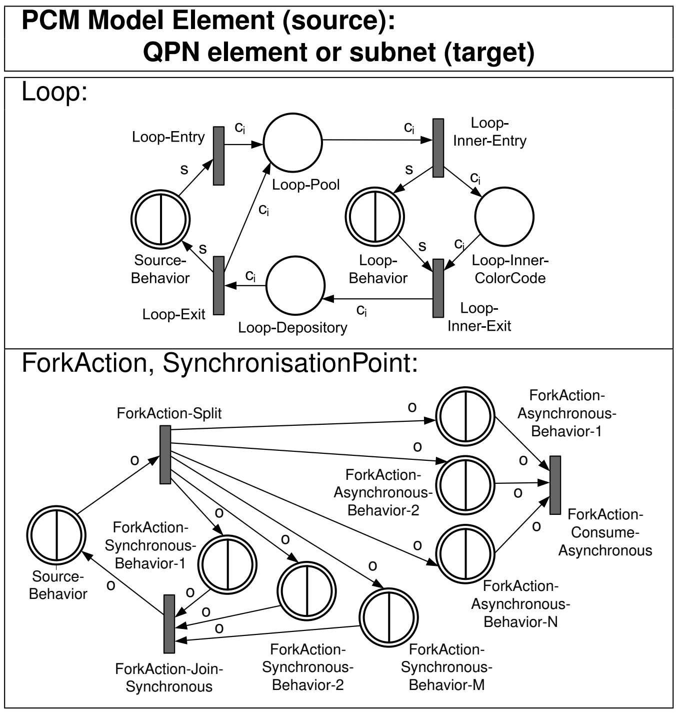
PCM Model Element (source):
QPN element or subnet (target)
Loop:
Loop-Entry
Loop-Pool
Loop-Inner-Entry
Source-Behavior
Loop-Behavior
Loop-Inner-ColorCode
Loop-Exit
Loop-Depository
Loop-Inner-Exit
s
c_i
ForkAction, SynchronisationPoint:
ForkAction-Split
ForkAction-Asynchronous-Behavior-1
ForkAction-Asynchronous-Behavior-2
ForkAction-Consume-Asynchronous
ForkAction-Synchronous-Behavior-1
Source-Behavior
ForkAction-Asynchronous-Behavior-N
ForkAction-Join-Synchronous
ForkAction-Synchronous-Behavior-2
ForkAction-Synchronous-Behavior-M
o

Fig. 8. Conceptual overview of the transformation PCM2QPN. Initial marking and queue/transition configuration omitted for brevity.

the *Loop-Inner-ColorCode* place. The outer part handles the token input from the *predecessor* subnet and the token output to the *successor* subnet. The inner part handles the input and output to and from the child behavior subnet, denoted as *LoopBehavior*, whose input and output tokens are referred to as the *loop input color*.

The number of loop iterations is decided at the loop entry and encoded into a new color. For each of the N possibilities of iteration counts, one color C_i and one mode M_i is generated in the *Loop-Entry* transition. The firing weight of each mode is set to P_i. The *Loop-Inner-Entry* transition takes a token of color C_i from the *Loop-Pool*, generates a token of the loop input color in the loop child behavior denoted as *LoopBehavior*, and generates a token of color C_i in place *Loop-Inner-ColorCode*. The loop input color is used in the *LoopBehavior* to limit the number of modes and colors in the child behavior subnet to a minimum. The iteration count information encoded in C_i is needed only locally in this part of the mapping and is irrelevant inside the subnet representing the child behavior. The *Loop-Inner-ColorCode* place is necessary so that the *Loop-Inner-Exit* transition knows which color C_i to generate in the *Loop-Depository* after the loop input color token exits *LoopBehavior*. The *Loop-Exit* transition contains two different modes for each color C_i. One mode to leave the loop and one mode to return the token of color C_i to the *Loop-Pool* for another child behavior iteration. The exit mode has a firing weight of P_n where n is the iteration count of color C_i. The return mode has a firing weight of $1 - P_n$. The random selection between the two modes for color C_i at the *Loop-Exit* transition behaves like a Bernoulli random variable. The number of loop iterations is therefore geometrically distributed with an expected value of $1/P_n$. Therefore, we choose $P_n = 1/n$ so that the mean number of times that the child behavior is executed equals the expected value $1/P_n = 1/(1/n) = n$. The limitation is that for an individual request the number of times the internal behavior is executed does not necessarily equal n.

Forks with Synchronization Barrier: The second row of Fig. 8) shows the mapping of M synchronous respectively N asynchronous fork actions to QPNs. Both actions can be described with a number of M respectively N *ForkedBehavior*s. A synchronous fork action ends at a *SynchronizationPoint*. Mapped to QPNs, there are three transitions. *ForkAction-Split* consumes a token from the *predecessor* subnet and creates a token in each of the $M + N$ child behavior subnets. *ForkAction-Consume-Asynchronous* consumes tokens from the N asynchronous child behavior subnets, but does not create any new tokens. *ForkAction-Join-Synchronous* waits until a token is available in each of the M synchronous child behavior subnets. It then consumes all of them and creates a new token in the *successor* subnet. When $M = 0$, a dummy ordinary place is created for the synchronized client behavior subnet to prevent the request from getting lost. A limitation applies to the mapping of synchronized forked behavior. The synchronization of two subrequests generated by a single parent request cannot be represented in a semantically equivalent fashion using QPN constructs. Individual tokens carry no identity and it cannot be decided for two tokens whether or not they belong to the same parent request. The tokens are consumed without considering their parent request, introducing an error. The extent of the error depends both on the number of parallel behaviors and on the properties of the child behavior subnets. In summary, besides the inaccuracies introduced by the dependency solver, the QPN mappings of both loops and synchronized forks introduce differences in model semantics.

3.4 PCM2QPN-MV

SimQPN allows the simulation using model parameters characterized with empirical distributions, and can provide prediction results as distributions. However, this comes at the cost of prediction speed, since each request has to be logged during simulation. In this section we describe PCM2QPN-MeanValue (PCM2QPN-MV). PCM2QPN-MV is based on transformation PCM2QPN (cf. Section 3.3) but uses SimQPN-MV as simulator. SimQPN-MV is a variant of SimQPN that limits the simulation to collect only statistics necessary for estimating the mean values of the considered metrics without any information on their probability distribution. Due to the reduced logging overhead during simulation, we expect PCM2QPN-MV to be significantly faster than PCM2QPN.

Transformation and Semantic Gaps

Instead of logging token residence time statistics at each ordinary place and queueing place, SimQPN-MV introduces measurement places. Measurement places are ordinary places that are connected in parallel to service behavior subnets. They collect only statistics about mean token population N and mean token departure rate X. Using Little's Law N/X [32], the mean token residence time in the measurement place can then be obtained. This equals to the mean response time of the connected service behavior.

Probability Distributions: If average metrics such as average service response times are sufficient, PCM2QPN-MV is a valid alternative to derive performance predictions. Thus, if percentiles such as the 90th percentile of a response time are of interest, PCM2QPN-MV is not appropriate. Furthermore, skewness or multi-modality of probability distributions remain undetected.

3.5 PCM2LQN

In order to further improve prediction speed, we apply an analytical solving method providing mean values. To this end, we transform to Layered Queueing Networks (LQNs) [33], [34]. LQNs target the performance analysis of distributed business information systems. With LQNS, there is an analytical solver available that is based on an enhanced version of the Method of Layers [35] solution algorithm that internally relies on the Linearizer approximative Mean Value Analysis (MVA) algorithm including M/M/n queues.

Layered Queueing Networks (LQNs)

In addition to the elements of plain queueing networks, LQNs model software entities and their communication explicitly in a hierarchical structure. An LQN (cf. example in Fig. 9) is an acyclic graph and consists of *processors* (circles) and *tasks* (parallelograms). Processors model hardware entities such as CPUs, hard disks, or networks. Tasks model software entities, such as components, application servers, databases, threads, or buffers. Tasks are arranged in a layered hierarchy, where tasks from upper layers may send requests to tasks from lower layers. Both processors and tasks contain a request queue, from which they serve waiting requests according to a specific scheduling discipline (e.g., FCFS or Processor Sharing). A task can contain multiple *entries* modeling the services provided by the software entity. Entries either directly specify a resource demand to the underlying processor of the tasks, or include a control flow graph containing multiple *activities* that issue such demands. Both entries and activities can also make calls to the entries of tasks at lower layers of the LQNs. The control flow graphs for activities support sequences, branches, loops, and forks.

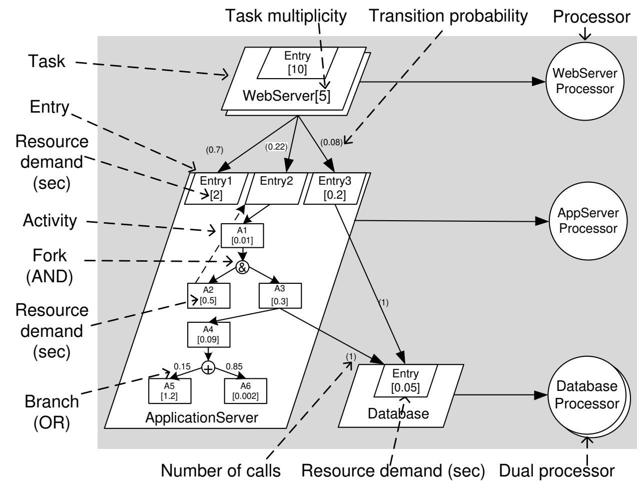

Fig. 9. An example LQN model.

Transformation and Semantic Gaps

Analog to QPNs, transforming performance-annotated software architecture models to LQNs requires a preprocessing step to resolve parameter dependencies as previously described in Section 3.2. Fig. 10 shows a part of the transformed running example in detail. Service behavior *IUser.ProvideService*, directly invoked from the *Customer* usage scenario, is mapped into an LQN task with a task activity graph. For each resource demanding action and resource type, the transformation creates an activity that synchronously calls a task running on the corresponding LQN processor with the corresponding resource demand. The tasks created for service behaviors itself run on dummy LQN processors. Control flow constructs such as sequences, branches, loops can be mapped to their counterparts in LQN task graphs. For example, the branch *DbUpdate* is annotated with probabilities 0.8 and 0.2 in Fig. 10. Calls to other services such as the call to *IDBQuery.Update* are transformed into an LQN activity with zero host demand and a synchronous call to the task representing the called service. A detailed description of the transformation can be found in [16].

Both SimQPN and LQNS abstract away from individual request semantics and use request class semantics instead. For example, it is still known which service each request originated from, but two such requests cannot be distinguished any further. For this reason, request tracking as used to limit loop iterations in loop actions cannot be adequately mapped in LQNS or SimQPN.

Forks with Synchronization Barrier: Because LQNs do not contain individual request semantics, LQNS uses an approximation to estimate the join delays of synchronous fork and join actions [34, Sec. 4.2].

Flexible Parameter Characterizations: Both SimQPN-MV and LQNS provide only mean value predictions, however, with LQNS, all model parameters have to be defined as exponential distributions. Thus, the diverse probability distributions of the input model are reduced to their mean values. Model parameters include

Modeling feature	SimuCom	SimQPN	SimQPN-MV	LQNS
Loops	X	(X)	(X)	(X)
Forks with synchronization barrier	X	(X)	(X)	(X)
Parameter dependencies	X	(X)	(X)	(X)
Response time distributions	X	X	-	-
Flexible parameter characterizations	X	X	X	-
Blocking behavior	X	X	X	(X)
X support (X) partial support - no support				

TABLE 1
Semantic gaps.

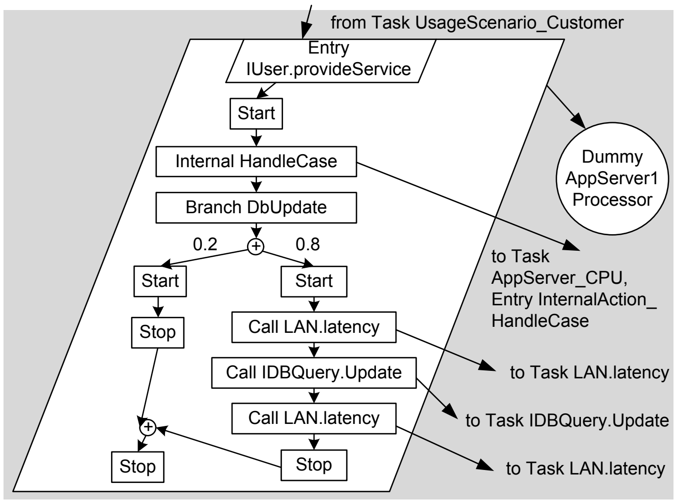

Fig. 10. LQN result of transforming *User.ProvideService* of the running example.

inter-arrival times of open workloads, think times of closed workloads, loop iteration numbers and resource demands.

Blocking Behavior: Blocking behavior that corresponds to the layer structure of an LQN can be captured by using the multiplicity of a task. Thus, a task can represent software resources such as threads of control or buffers [34, Sec. 2.3]. LQNS cannot handle cases where the acquire and release steps are performed by different tasks [36].

3.6 Summary

We identified various semantic gaps between target and source model of the presented transformation-based approaches. Tab. 1 shows an overview, it summarizes which prediction approach supports which modeling feature. While SimuCom provides full support for all mentioned modeling features, SimQPN provides only partial support of synchronous forks and joins, loops as well as parameter dependencies. SimQPN-MV additionally does not return response time distributions. LQNS only partially supports blocking behavior and requires model parameters such as resource demands and inter-arrival times to be exponentially distributed. In the next section, on the hand we evaluate the effect of the individual semantic gaps on the prediction accuracy, on the other hand we evaluate the time-to-result of the presented prediction approach in representative case studies.

4 Evaluation

In this section we first introduce our evaluation goals (Section 4.1). Then we investigate the effects of the identified semantic gaps (Section 4.2). In Section 4.3, we present several case studies to provide end-to-end evaluations of the different model transformations.

4.1 Evaluation Goals

In order to evaluate and compare the advantages and disadvantages of the model transformations and solution techniques introduced in this paper, we ask the following questions:

- What effects have the semantic gaps on the accuracy of performance predictions?
- What is the efficiency of each transformation and respective analysis tool?

To evaluate the first question we compare (a) the performance predictions obtained using the various transformations to different target models with (b) the respective reference results obtained using a complete simulation of the original source model. As source model we use PCM (cf. Section 2). For our experiments we use SimuCom's results as a reference since, due to the use of simulation techniques, it is the only solver that supports the full PCM meta-model. This is in contrast to using measurements on the real system as reference which is used to evaluate model accuracies. However, we want to compare the accuracy of the different transformation-based analysis tools independent of the considered models' accuracies themselves.

A metric to evaluate the second question is the execution time for each transformation and analysis tool chain. For the simulation-based solvers SimuCom and SimQPN, in order to ensure a fair comparison, the respective simulation times and confidence intervals are aligned.

A quantitative comparison of the transformations introduced in this paper to related transformation-based approaches is not in the scope of our evaluation. Other transformations are either not intended for component-based systems or lack stable implementations for a fair

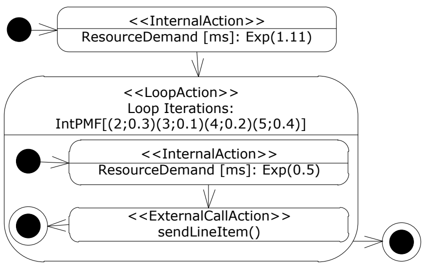

Fig. 11. Loop as part of service `sendOrdersTo-Manufacturing`.

comparison (cf. Section 6). We also do not validate the benefit of component-based modeling here, which has been demonstrated in previous publications [12], [37].

4.2 Semantic Gaps

For each semantic gap between the different transformation-based prediction approaches as they are shown in Tab. 1, we investigate a typical prediction scenario to assess the semantic gap's effect on the prediction accuracy.

Loops

Fig. 11 shows a behavior specification of a service named `sendOrdersToManufacturing`. An order consisting of multiple line items is sent to the manufacturing domain. After a short pre-processing step (an internal action), the service loops through the list of line items and sends them separately to manufacturing. The example is an adapted `newOrder` service of the SPECjEnterprise2010 benchmark[1] (cf. Section 4.3.1). The loop iteration number is described by a PMF. With a probability of 30%, the loop iterates 2 times. Loop counts 3 and 4 have a probability of 10% respectively 20%, 5 loop counts have a probability of 40%. While SimuCom is able to simulate exactly the given PMF, SimQPN introduces inaccuracies because of the geometric distribution of the individual loop iteration numbers in the transformed QPN (cf. Section 3.3). In Fig. 12, we see the PMF obtained using SimuCom in the left diagram. The right diagram shows the PMF of loop iteration numbers when using SimQPN. It is more spread out compared to the original PMF. Note that the resulting average of loop iteration numbers is 3.7, it is the same for both SimuCom and SimQPN. In general, the higher the loop iteration numbers, the more visible is the semantic gap of the loop mapping to QPNs.

Fig. 13 and Tab. 2 compare the response time predictions for service `sendOrdersToManufacturing` derived with SimuCom respectively SimQPN. As expected, the predicted average response times are comparable.

1. SPECjEnterprise2010 is a trademark of the Standard Performance Evaluation Corporation (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjEnterprise2010 is located at http://www.spec.org/jEnterprise2010.

Predicted Response Time (RT) [ms]	Simu-Com	Sim-QPN
average	8.28	8.35
90th percentile	14.47	18.58
95th percentile	16.73	25.03

TABLE 2
Predictions of `sendOrdersToManufacturing`: SimuCom vs. SimQPN.

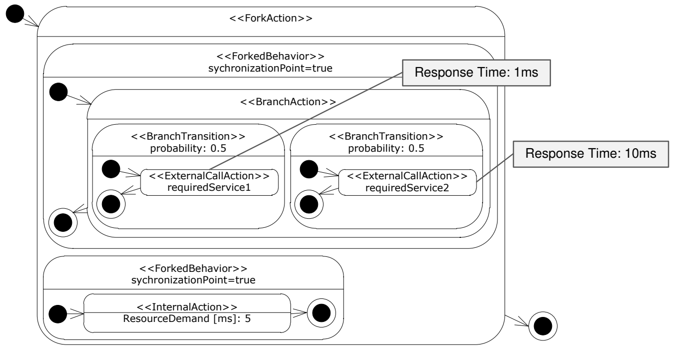

Fig. 14. Two Forks with a synchronization barrier.

The response time distribution is more spread out with SimQPN than with SimuCom although in the example the exponentially distributed resource demands tend to smooth the differences of the predictions.

Forks with Synchronization Barrier

To analyze the semantic gap when modeling forks with a synchronization barrier, we describe an illustrative example. Fig. 14 shows a behavior specification involving a synchronized fork of two threads. One thread (first `ForkedBehavior`) contains a branch of two equiprobable external service calls. The external service calls have a response time of either 1 or 10ms. The other thread (second `ForkedBehavior`) contains an internal action that consumes a resource demand of 5ms. When deriving the response time predictions of SimuCom and SimQPN, we run a closed workload with a low resource utilization level (the resource demand of the internal action of 5ms thus translates into a response time of 5ms). As expected, SimuCom returns an average response time of $(10\text{ms} + 5\text{ms})/2 = 7.5\text{ms}$ for the overall fork action. SimQPN however returns an average response time of 5.9ms with 10 concurrent users. In the transformed QPN, since individual tokens carry no identity, the synchronization barrier cannot distinguish (cf. Section 3.3) if two sub-requests stem from the same parent request or not. The extent of the error depends on the response time variability of the individual forked behaviors and the level of concurrency in the system.

Parameter Dependencies

When resolving parameter dependencies to probability distributions as described in Section 3.2, subsequent uses of the same variable are treated as they were

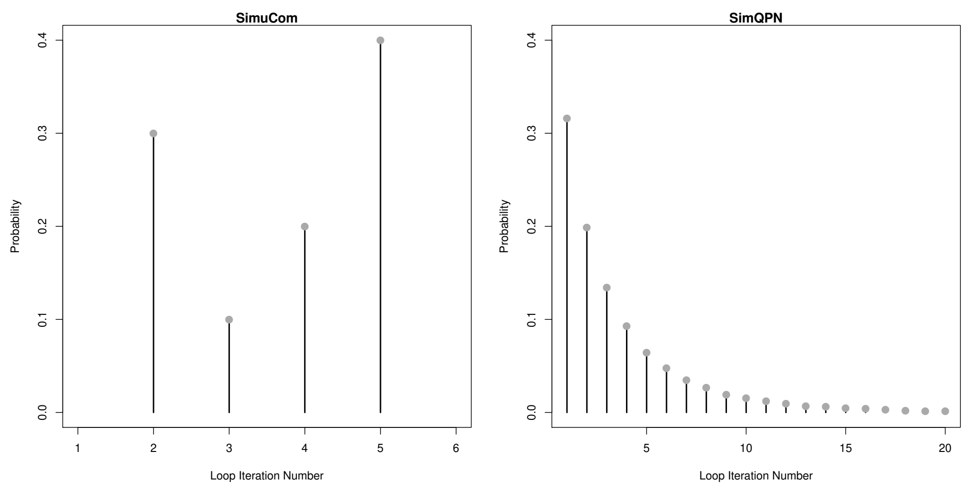

Fig. 12. Loop iteration numbers: SimuCom vs. SimQPN.

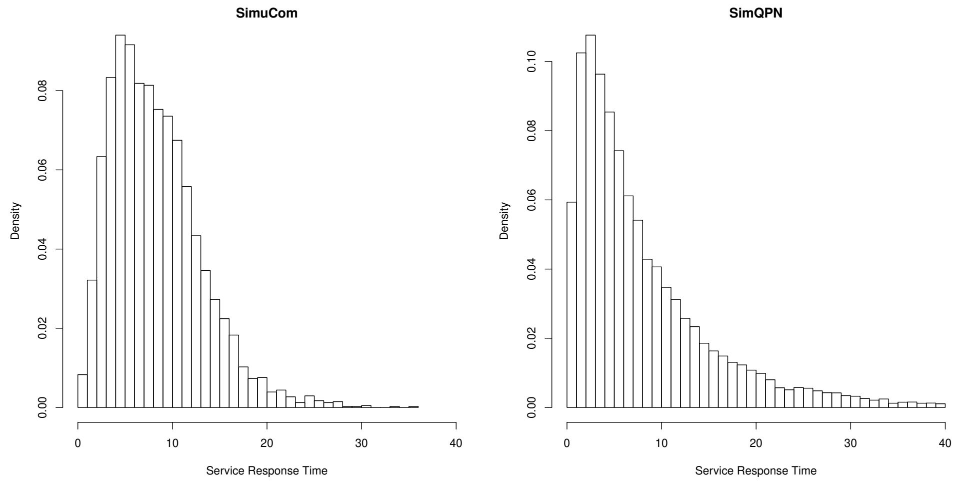

Fig. 13. Response time predictions of service `sendOrdersToManufacturing`: SimuCom vs. SimQPN.

independent but should be stochastically dependent. In the following we investigate the performance-relevant behavior of a time-consuming computation. The resource demand for the computation depends on a parameter X, more specifically, the resource demand is estimated to be 100ms times the value of X. The service implementation illustrated in Fig. 15 does a lookup in a hashtable for small values of X (values smaller than 100) and triggers the computation only for large values of X (values greater or equal to 100). In the example, the value of X is characterized with a PMF. Parameter X has values of 10 and 50 with a probability of 70% respectively 20%. We run a closed workload in a scenario with a low resource utilization. Because there are no waiting times, the resource demands of the internal actions directly translate to response times. Using SimuCom we obtain an average service response time of 1500.9ms as expected. SimQPN however returns an average service response time of 320.9ms. This is because the resource demand of the internal action in the branch transition for values greater or equal 100 is only approximated. The PMF of input parameter X is directly used to characterize the internal action's resource demand *without* considering the branch condition. Solving parameter dependencies without considering subsequent uses of the same variable thus may lead to notable prediction errors.

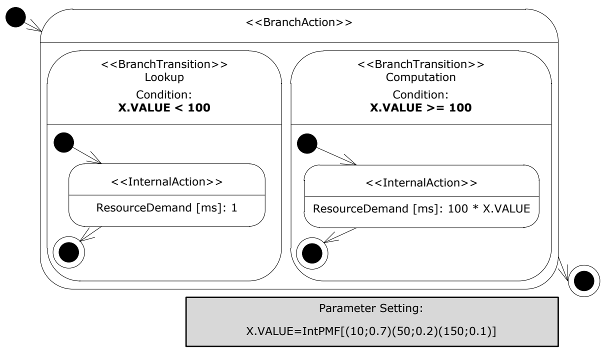

Fig. 15. Resolved parameter dependencies.

Response Time Distributions

Restricting the analysis to mean value metrics obviously can be misleading as shown in Fig. 16 for the Media Store (cf. Section 4.3.1). Considering only the mean response time of 1.31 seconds, we omit all information about the complex underlying probability distribution.

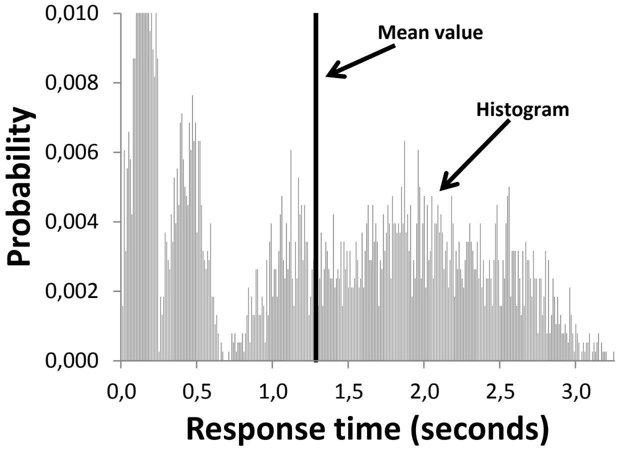

Fig. 16. Response time histogram of the Media Store Scenario 1.

Flexible Parameter Characterizations

With LQNS, all model parameters have to be defined as exponential distributions. Model parameters include interarrival times of open workloads, think times of closed workloads, loop iteration numbers and resource demands. All probability distributions of the input model are thus reduced to their mean values. For loop iteration numbers, this reduction has no effect on the predicted mean response times. However, for the other mentioned model parameters, this reduction has an impact on the accuracy of mean response time predictions. Let us assume there is a simple Queueing Network consisting of only one single server queue and one workload class with an open workload. For a constant inter-arrival time of 2 seconds (i.e., an arrival rate λ of 0.5 requests per second) and a constant resource demand D of 1 second, the predicted mean response time R equals to $D = 1$. If inter-arrival time and resource demand are not constant but exponentially distributed, the predicted mean response time then is $R = \frac{D}{1-U} = \frac{D}{1-(D*X)} = \frac{1}{1-0.5} = 2$, where U, X are the queue utilization respectively throughput [32], [38]. Thus, using LQNS introduces prediction errors if input model parameters need to be converted to exponential distributions.

4.3 Case Studies

In this section we provide end-to-end evaluations of the different model transformations in various case studies. The system under study are described in Section 4.3.1. Section 4.3.2 and Section 4.3.3 describe the experiment setup respectively the experiment results.

4.3.1 Systems Under Study

To increase the evaluation's external validity, we applied the model transformations to four distributed, component-based systems. They represent different domains, such as enterprise systems, web applications, and industrial automation:

- **Media Store** is a plain Java web application for storing and retrieving audio or video files using a MySQL database. The model reflects a use case where a digital watermark is added to downloaded files for copy protection. The model contains a hard disk resource, which is accessed when retrieving files. Resource demands for the Media Store RDSEFFs have been measured using manual instrumentation of the Java implementation [39].
- **SPECjEnterprise2010** is an industry-standard benchmark designed to measure the performance of application servers conforming to the Java EE 5.0 or later specifications. It is modeled after an automobile manufacturer, where dealers place customer orders or interact with suppliers. The system is implemented as a Java Enterprise application deployed on two servers using an Oracle database. Resource demands have been determined using an estimation technique based on measured response times and resource utilization [22].
- **Process Control System (PCS)** is a distributed system to manage industrial processes, such as power generation, oil refinement, or pulp and paper processing. The model focuses on the server-side part of the system, which is implemented in C++ using Microsoft technologies. It features four usage scenarios, for example for transferring sensor data or managing alarm events. Resource demands were determined using the Windows Performance Monitor [20].
- **Business Reporting System (BRS)** is loosely modeled after a management information system formerly analyzed at Carlton University [40]. The analyzed configuration comprises two usage scenarios, nine components, and four servers. Users can retrieve live business data from the system and run statistical analyses. Resource demands of the BRS are based on estimations. To better highlight differences of the analysis tools, we analyze two workload scenarios. Scenario 1 has a constant interarrival time of 1 second (abbr. cons1), while Scenario 2 has an exponentially distributed interarrival time with mean value of 1 second (abbr. Exp(1)).

Fig. 17 shows a component deployment view of each model. Each modeled component may include multiple RDSEFFs which are not shown here for brevity. The complete models can be retrieved from our website[2]. The models range from small size (e.g., Media Store) to large size (e.g., PCS) to demonstrate the scalability of the transformations and analysis tools.

2. https://sdqweb.ipd.kit.edu/wiki/QuanEval_MDSPE

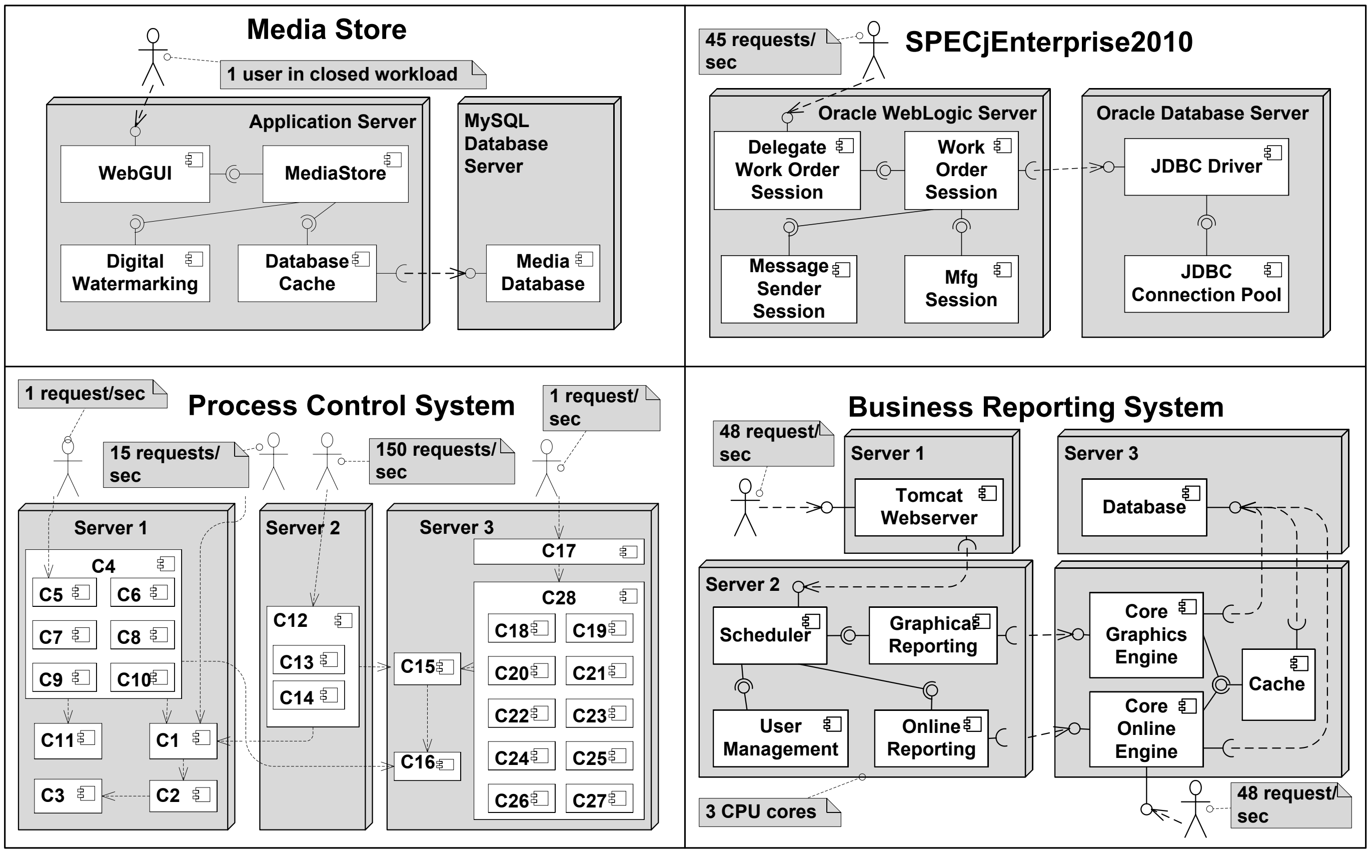

Fig. 17. Combined component and deployment views of the four case study systems evaluated with the introduced transformations and analysis tools. The inter-arrival time of all open workloads is exponentially distributed.

Model	Usg.Sc.	Comp.	RDSEFFs	Int. Act.	Res. Dem.	Ext. Act.	Loops	Branches
Media Store	1	5	14	19	19	11	3	2
SPECjEnterprise2010	1	7	33	33	28	20	1	4
Process Control System	4	39	33	28	28	42	0	10
Business Reporting System	1	9	38	37	37	40	5	4
Key: Usg.Sc. = Usage Scenarios, Comp. = Components, Int. Act. = Internal Actions, Res. Dem. = Resource Demands, Ext. Act. = External Actions								

TABLE 3
Comparison of the complexity of the considered models.

Tab. 3 shows a number of model complexity metrics. The number of calls, branches, and loops refer to the number of entities that are instantiated during the model traversal of both the usage model and the system assembly.

To give the reader an idea of the performance analysis models generated by the transformations, Fig. 18 shows the generated LQN model for the BRS. Furthermore, Tab. 4 compares the size and complexity of the generated LQN and QPN models for the various modeled systems. These models are hidden from the user and the performance metrics resulting from the analysis can be retrieved without knowing the details of each modeling notation.

4.3.2 Experiments

All experiments executing the transformations and analysis tools on the case study models were conducted using Eclipse Galileo 3.5.2 with the following features installed: EMF 2.5.0, Eclipse QVT Operational 2.01, PCM 3.2 Development Build, QPME 1.5.2 Development Build. The system and hardware configuration included: Microsoft Windows XP, JDK 1.6.0 (Update 26), Intel Core2 Duo CPU T9500 at 2.6 GHz. LQNS was available in version 4.1.

For the simulation analyses, we used the same number of measurements for each case study to achieve comparable results. Of the available simulation methods offered by QPME/SimQPN [41], [42], the batch means method was used for all simulation runs, which is the standard method recom-

	QPN					LQN			
	Pl.	Tr.	Cn.	Col.	Qs	Tasks	Proc.	Entr.	Act.
Media Store	51	48	105	17	5	14	14	25	62
SPECjEnterprise2010	102	98	217	40	5	25	25	45	134
Process Control System	170	150	359	53	10	36	36	91	188
Business Reporting System	315	251	710	105	6	50	50	88	290
Key: Pl. = Places, Tr. = Transitions, Cn. = Connections, Col. = Colors, Qs = Queues, Proc. = Processors, Entr. = Entries, Act. = Activities									

TABLE 4
Model elements in the resulting QPNs and LQNs.

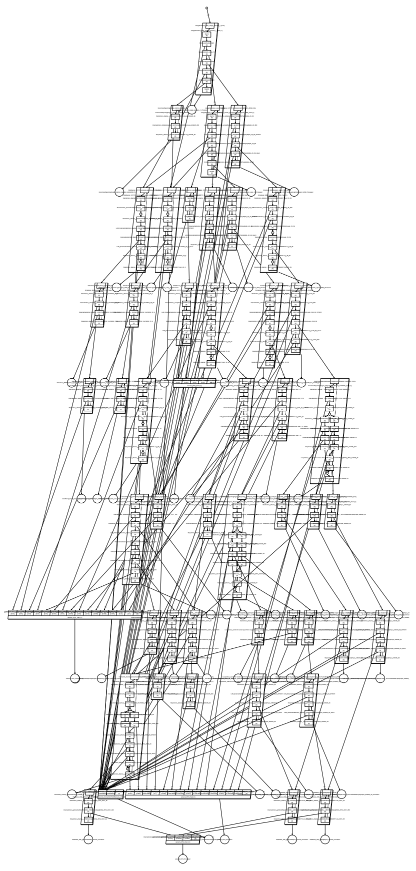

Fig. 18. Generated LQN for the BRS.

mended by SimQPN. For LQNS, the default settings were used: `convergence = 0.00001`, `iteration Limit = 50`, and `underrelaxation = 0.5`.

4.3.3 Results

Tab. 5 shows the prediction results for the different analysis tools. The table shows response times, throughputs, and resource utilizations for each case study system. For simulation-based solvers, point estimates of the considered metrics are shown. All results were very stable with the variation of reported metrics from multiple simulation runs being negligible. The accuracy of the transformations and analysis tools can be derived from the columns labelled 'relDiff', which include the percentaged deviation of each performance metric from the reference metric provided by SimuCom. Performance metrics with a deviation higher than 15 percent from the reference value are marked in light grey in the table.

From the simulators and the analytical solver, there is only one larger deviation: For the BRS system, the predicted response time differs in Scenario 1. While the SimQPN solvers have a slight deviation, the LQNS solver deviates significantly: Both predict a higher response time than SimuCom. The reason for the deviation is the constant interarrival time in this scenario: LQNS does not support constant interarrival times, so the PCM2LQN transformation maps it to an exponentially distributed arrival rate, leading to the strong deviation.

In summary, the prediction results of the analytical and simulation solvers show only a low deviation (< 10 percent) from the reference results in most cases. Thus, in our case studies the identified semantic gaps often do not have a significant impact on the overall performance.

To answer our second question from the beginning of Section 4.1 concerning the efficiency of the various analysis tools, Tab. 6 shows some statistics about the execution times of the analysis tools. In general, SimQPN and LQNS deliver results much faster than SimuCom. SimQPN exhibited a 2 to 9 times shorter execution time, whereas SimQPN-MV was even faster with up to 24 times shorter execution time in some cases. Furthermore, although being a simulation-based solver, SimQPN has been shown to scale well to models of significant size and complexity [43]. LQNS, being an analytical solver, is faster than the other solvers. It exhibited at least 10 times and up to 50 times shorter execution times.

5 Discussion

This section summarizes the trade-offs between the various performance transformations and analysis tools we uncovered in Section 3 and Section 4. We provide practical guidance on deciding when to use which analysis tool. Users can select a suitable transformation and respective solver according to their required features from Tab. 1, as well as their constraints concerning the prediction accuracy and overhead. For example, if response time distributions are needed as part of the prediction, then SimuCom provides the best support at the cost of higher prediction overhead. In situations where the prediction overhead is an important factor, SimQPN is the preferred solver as it executes significantly faster than SimuCom and provides a good balance between

	SimuCom (reference)	SimQPN	SimQPN (relDiff)	SimQPN-MV	SimQPN-MV (relDiff)	LQNS	LQNS (relDiff)
Media Store							
RT(Scenario1)	1,332	1,331	0,0%	1,324	-0,6%	1,288	-3,3%
TP(Scenario1)	0,751	0,751	0,0%	0,755	0,6%	0,753	0,3%
U(AppServer_CPU)	0,341	0,345	1,2%	0,340	-0,3%	0,342	0,2%
U(DBServer_CPU)	1,4%	1,4%	1,6%	1,4%	2,6%	1,4%	2,2%
U(DBServer_HDD)	64,4%	64,0%	-0,6%	64,5%	0,2%	64,4%	-0,1%
SPECjEnterprise2010							
RT(Scenario1)	1,060	1,058	-0,2%	1,065	0,4%	1,065	0,4%
TP(Scenario1)	24,567	24,970	1,6%	24,792	0,9%	25,000	1,8%
U(Oracle_CPU)	29,1%	29,6%	1,7%	29,5%	1,3%	29,6%	1,8%
U(WLS_CPU)	51,2%	52,0%	1,5%	51,9%	1,3%	52,2%	2,0%
Process Control System							
RT(Scenario1)	0,00787	0,00764	-2,9%	0,00763	-3,0%	0,00790	0,4%
RT(Scenario2)	0,06690	0,06636	-0,8%	0,06903	3,2%	0,06710	0,3%
TP(Scenario1)	149,258	149,254	0,0%	149,254	0,0%	149,254	0,0%
TP(Scenario2)	15,001	15,000	0,0%	15,000	0,0%	14,999	0,0%
U(Server1_CPU)	6,5%	6,4%	-2,2%	6,4%	-2,0%	6,5%	-0,5%
U(Server1_HDD)	45,0%	45,0%	0,0%	45,0%	0,0%	45,0%	0,0%
U(Server2_CPU)	1,1%	1,1%	-4,0%	1,1%	-4,0%	1,1%	-4,0%
U(Server3_CPU)	55,0%	54,1%	-1,6%	54,0%	-1,8%	55,1%	0,2%
Business Reporting System							
RT(Scenario1 - cons1)	5,223	5,918	13,3%	5,921	13,4%	7,391	41,5%
TP(Scenario1 - cons1)	1,000	1,000	0,0%	1,000	0,0%	1,000	0,0%
U(Server1_CPU) Scen. 1	45,0%	44,8%	-0,5%	44,9%	-0,2%	45,0%	0,0%
U(Server2_CPU) Scen. 1	68,8%	68,6%	-0,3%	68,8%	-0,1%	68,9%	0,1%
U(Server3_CPU) Scen. 1	74,3%	74,8%	0,7%	74,5%	0,3%	74,3%	0,0%
U(Server4_CPU) Scen. 1	23,5%	23,5%	-0,2%	23,5%	0,1%	23,5%	0,2%
RT(Scenario2 - Exp(1))	7,302	7,256	-0,6%	7,500	2,7%	7,391	1,2%
TP(Scenario2 - Exp(1))	1,002	1,000	-0,2%	0,996	-0,6%	1,000	-0,2%
KEY: RT = Response Time (sec), TP = Throughput (requests / sec), U = Utilization (%), relDiff = relative difference							

TABLE 5
Results accuracy of the various analysis tools compared to SimuCom.

	SimuCom (reference)	SimQPN	SimQPN (relDiff)	SimQPN-MV	SimQPN-MV (relDiff)	LQNS	LQNS (relDiff)
Media Store							
Logical Simulation Time	25000	25000	0,0%	25000	0,0%	n/a	n/a
Number of Measurements	18773	18775	0,0%	18775	0,0%	n/a	n/a
Total Execution Time	31,4	6,4	-79,6%	3,6	-88,5%	1,3	-96,0%
Wall-clock Sim./Ana. Time	25,6	1,7	-93,4%	0,9	-96,5%	0,2	-99,1%
SPECjEnterprise2010							
Logical Simulation Time	500	500	0,0%	500	0,0%	n/a	n/a
Number of Measurements	12257	12485	1,9%	12485	1,9%	n/a	n/a
Total Execution Time	72,6	7,9	-89,1%	5,6	-92,3%	1,8	-97,6%
Wall-clock Sim./Ana. Time	55,6	3,8	-93,3%	2,1	-96,3%	0,5	-99,2%
Process Control System							
Logical Simulation Time	500	500	0,0%	500	0,0%	n/a	n/a
Number of Measurements	74627	74627	0,0%	74627	0,0%	n/a	n/a
Total Execution Time	96,7	11,9	-87,7%	7,2	-92,6%	3,9	-95,9%
Wall-clock Sim./Ana. Time	65,4	1,6	-97,6%	2,6	-96,0%	2,0	-96,9%
Business Reporting System							
Logical Simulation Time	9994	10000	0,1%	10000	0,1%	n/a	n/a
Number of Measurements	10016	9997	-0,2%	9997	-0,2%	n/a	n/a
Total Execution Time	591,5	74,5	-87,4%	34,1	-94,2%	4,0	-99,3%
Wall-clock Sim./Ana. Time	587,6	35,4	-94,0%	26,6	-95,5%	1,8	-99,7%
All times in seconds							

TABLE 6
Comparison of the efficiency of the various analysis tools.

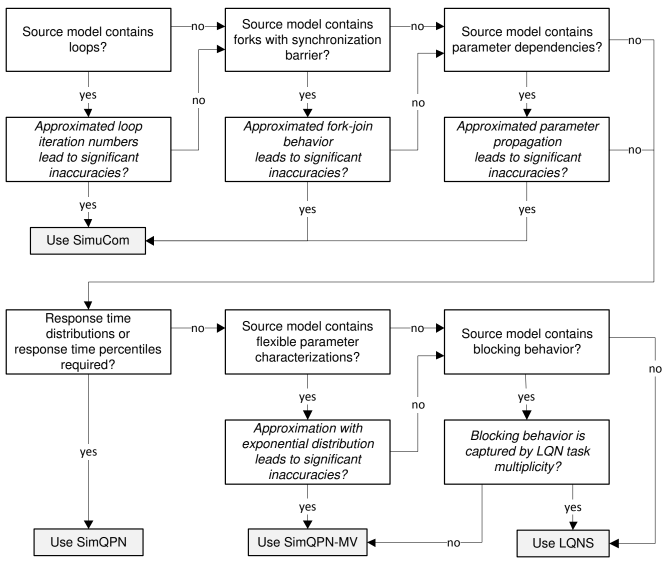
Source model contains loops?
Source model contains forks with synchronization barrier?
Source model contains parameter dependencies?
Approximated loop iteration numbers lead to significant inaccuracies?
Approximated fork-join behavior leads to significant inaccuracies?
Approximated parameter propagation leads to significant inaccuracies?
Use SimuCom
Response time distributions or response time percentiles required?
Source model contains flexible parameter characterizations?
Source model contains blocking behavior?
Approximation with exponential distribution leads to significant inaccuracies?
Blocking behavior is captured by LQN task multiplicity?
Use SimQPN
Use SimQPN-MV
Use LQNS
yes
no

Fig. 19. Decision tree for selecting suitable transformation and analysis tool.

prediction accuracy and overhead. LQNS and SimQPN-MV are most efficient in terms of prediction overhead, however, they only provide mean value metrics.

Fig. 19 depicts a decision tree that is based on the gaps between the presented prediction approaches as they are shown in Tab. 1. In scenarios where the prediction overhead is important, this decision tree guides the selection of a specific transformation and respective solver. The less complex modeling features are used, the faster prediction approaches can be selected without losing prediction accuracy. There are prediction approaches that support some modeling features only partially, i.e., the approaches approximate the modeling features. If the approximation leads to acceptable prediction inaccuracies or not, differs from scenario to scenario. In Fig. 19, we mark such decisions using *italic* font. For instance, the decision if the approximation of loop iteration numbers (as it is presented in Section 3) leads to significant inaccuracies cannot be taken without knowing (i) the loop iteration number distribution, (ii) the performance-relevance of the loop body, (iii) the actual performance metrics that need to be predicted, and (iv) the accuracy a given scenario requires. However, the explanation of the semantic gaps in Section 3 and their individual evaluation in Section 4 provide assistance to answer such questions.

The selected transformations considered in this paper as well as the respective model solution techniques serve as representative examples of the typical model types and solution techniques used for performance prediction in practical studies. The considered case studies were carefully chosen to cover different modeling features and to represent systems of different size and complexity. Thus, even though the results were presented in the context of the Palladio Component Model (PCM), the gained insights from the evaluation are easily generalizable to other systems, both in terms of the expressiveness and limitations of the various transformation approaches, as well as in terms of the accuracy and efficiency of the various solution techniques. The presented results and insights help software architects and performance engineers to select an appropriate model transformation and solution approach for a given context.

6 Related Work

The main contribution of this paper is related to other quantitative comparisons between transformation approaches and analysis tools. The compared transformations themselves are related to a variety of approaches in software performance engineering.

6.1 Quantitative Comparisons

There are only a few works quantitatively comparing different model transformations and analysis tools. Balsamo et al. [9] transformed an annotated UML model of a naval communications system both into Aemilia, a notation based on stochastic process algebras, and a simulation model. Although they executed both solvers and discussed their different scalabilities, the authors did not provide the deviations between the numerical solutions and the simulations results nor did they compare the respective solver execution times.

Tribastone [10] compared the solutions of LQNS and the numerical solution of the Markov chain underlying an PEPA model for a distributed system. He found a prediction error below 15 percent in most of the analyzed cases and showed that the LQN solver was orders of magnitude faster in cases with low concurrency levels. In another paper, Tribastone et al. [11] demonstrated that the solution of the PEPA model of a three tier distributed system was up to four orders of magnitude faster based on ordinary differential equations than based on a continuous time Markov chain analysis. However these approaches did not involve model transformations as both variants were modeled by hand.

Other papers involving multiple model transformations into different performance models [6], [44] do not provide quantitative prediction results, but rather focus on the proof-of-concept feasibility of the transformation approach itself.

6.2 SPE Approaches

Numerous authors have proposed model transformations from higher-level software models to performance models [2]. Such approaches enable developers to reuse existing design models and keep the mathematical intrinsics of the performance models hidden and may thus lower the barrier to conduct model-based performance analyses in practice.

One common approach is to employ UML with extensions to support performance analysis. Early approaches use proprietary UML extensions [2], later the OMG standardized the UML SPT profile [45] and UML MARTE profile [46]. They have been applied for transformations into many different target languages ranging from layered queueing models to stochastic process algebra and stochastic Petri nets [4] [7], [8], [47], [48], [49]. However, UML lacks concepts to specify parameterizable, reusable models, which make an application in a component-based development scenario difficult. Thus, these approaches should only be applied if the system under analysis does not follow the component-paradigm.

To specifically support the performance analysis of component-based systems and exploit features of component-based software engineering (CBSE) such as reusability, division of work, and hierarchical composition, several research groups proposed special modeling languages [3]. However, Hissam et al. [50] specifically targets embedded systems, while the authors in [51] provide a restricted resource model. Bertolino et al. [52] provides limited support for parameterized component performance models and restricts the class of analyzable systems because of hard assumptions in the analysis model. The approach presented in [40] lacks the capability to change the control flow within a component depending on the parameter input and has limited tool support. Other approaches remain at a conceptual stage and have not been supported by tools, which complicates applying them in practice [53], [54], [55], [56].

The potentially large number of model transformations in software performance engineering from multiple high-level modeling notations into multiple low-level performance models have led to the creation of intermediate or kernel modeling languages [57]. However, Smith et al. [58], [59] tailor their approach towards a data interchange format making it less applicable in a performance analysis scenario. The KLAPER approach [6] lacks the concept of parameter dependencies and transformations have been proposed but not yet implemented. The prototypical model transformation from the Core Scenario Model (CSM) to LQNs described in [5] was not yet available for third-party use.

7 Conclusions

This paper presents a comparison of different transformations of performance-annotated software architecture models into stochastic performance models which apply both analytical and simulation-based solution strategies. It briefly introduces each transformation and describes the semantic gaps between typical source model abstractions and the different analysis techniques. We present a detailed evaluation of the effects of the identified semantic gaps on the prediction accuracy as well as several case studies to provide end-to-end evaluations of the different model transformations. We discuss the results in detail and give recommendations to select the right transformation and analysis tool for different analysis contexts. Even though the results involve the proprietary PCM, the gained insights from the evaluation are generalizable to other models and systems, both in terms of the expressiveness and limitations of the various transformation approaches, as well as in terms of the accuracy and efficiency of the various solution techniques.

As such, the results presented in this paper help performance analysts to select the right transformation depending on the analysis context or the degree of maturity of the model of the system under study. Analytical models can help to quickly guide the design process with early but more coarse-grained performance metrics while simulation helps for detailed performance analyses. Also the results presented in this paper help performance analysts to estimate the loss in accuracy implied by the use of analytic solvers.

One direction to extend the results presented in this paper is to derive an automatic selection method which decides on the transformation to use depending on the analysis context, i.e., automating taking the decision for a transformation according to the decision tree presented in this paper. The automated decision can include the level of details of the model of the system under study, available time until the performance prediction results are required, or the level of detail of modeling infrastructure aspects in the models.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center ”On-The-Fly Computing” (SFB 901) and by DFG grant No ”KO 3445/6-1”.

References

[1] C. U. Smith, *Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: A survey,” *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 295–310, 2004.

[3] H. Koziolek, “Performance Evaluation of Component-based Software Systems: A Survey,” *Performance Evaluation*, vol. 67, no. 8, pp. 634–658, August 2010.

[4] G. Gu and D. Petriu, “XSLT transformation from UML models to LQN performance models,” in *Proc. 3rd Int. Workshop on Software and Performance (WOSP’02)*. ACM, 2002, pp. 227–234.

[5] D. B. Petriu and M. Woodside, “An intermediate metamodel with scenarios and resources for generating performance models from UML designs,” *Journal of Software and Systems Modeling*, vol. 6, no. 2, pp. 163–184, June 2006.

[6] V. Grassi, R. Mirandola, and A. Sabetta, “Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach,” *Journal on Systems and Software*, vol. 80, no. 4, pp. 528–558, 2007.

[7] S. Bernardi and J. Merseguer, “Performance evaluation of UML design with Stochastic Well-formed Nets,” *Journal of Systems and Software*, vol. 80, no. 11, pp. 1843–1865, 2007.

[8] M. Tribastone and S. Gilmore, “Automatic extraction of pepa performance models from uml activity diagrams annotated with the marte profile,” in *Proc. 7th Int. Workshop on Software and Performance (WOSP’08)*, ser. WOSP ’08. New York, NY, USA: ACM, 2008, pp. 67–78.

[9] S. Balsamo, M. Marzolla, A. D. Marco, and P. Inverardi, "Experimenting different software architectures performance techniques: a case study," in *Proc. 4th International Workshop on Software and Performance (WOSP'04)*. New York, NY, USA: ACM, 2004, pp. 115–119.

[10] M. Tribastone, "Relating layered queueing networks and process algebra models," in *Proc. 1st Joint WOSP/SIPEW Int. Conf. on Performance engineering*, ser. WOSP/SIPEW '10. New York, NY, USA: ACM, 2010, pp. 183–194.

[11] M. Tribastone, S. Gilmore, and J. Hillston, "Scalable differential analysis of process algebra models," *IEEE Trans. on Softw. Eng.*, vol. TBD, 2012.

[12] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, January 2009.

[13] F. Bause, "Queueing Petri Nets˜- A formalism for the combined qualitative and quantitative analysis of systems," in *Proceedings of the 5th International Workshop on Petri Nets and Performance Models, Toulouse, France, October 19-22*, 1993.

[14] P. Meier, S. Kounev, and H. Koziolek, "Automated Transformation of Palladio Component Models to Queueing Petri Nets," in *In 19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011)*, July 25-27 2011.

[15] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, "Performance analysis of distributed server systems," in *Proc. 6th International Conference on Software Quality (ICSQ'96)*, 1996, pp. 15–26.

[16] H. Koziolek, "Parameter dependencies for reusable performance specifications of software components," Ph.D. dissertation, University of Oldenburg, Germany, March 2008.

[17] H. Koziolek and R. Reussner, "A model-transformation from the palladio component model to layered queueing networks," in *Proc. of the SPEC International Workshop on Performance Engineering (SIPEW'08)*, ser. LNCS, vol. 5119. Springer, June 2008, pp. 58–78.

[18] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," vol. 82, pp. 3–22, 2009.

[19] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. Reussner, "Performance Modeling in Industry: A Case Study on Storage Virtualization," in *ACM/IEEE 32nd International Conference on Software Engineering, Software Engineering in Practice Track, Capetown, South Africa*. ACM, 2010, pp. 1–10.

[20] H. Koziolek, B. Schlich, C. Bilich, R. Weiss, S. Becker, K. Krogmann, M. Trifu, R. Mirandola, and A. Martens, "An industrial case study on quality impact prediction for evolving service-oriented software," in *Proc. 33rd ACM/IEEE Int. Conf. on Software Engineering (ICSE'11) Software Engineering in Practice Track*. ACM, May 2011, pp. 776–785.

[21] J. Happe, H. Koziolek, and R. Reussner, "Facilitating performance predictions using software components," *IEEE Software*, vol. 28, no. 3, pp. 27–33, May 2011.

[22] F. Brosig, N. Huber, and S. Kounev, "Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems," in *26th IEEE/ACM Intl. Conference On Automated Software Engineering (ASE)*, November 2011.

[23] C. Rathfelder, S. Kounev, and D. Evans, "Capacity Planning for Event-based Systems using Automated Performance Predictions," in *26th IEEE/ACM Intl. Conference On Automated Software Engineering (ASE)*, November 2011.

[24] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek, "An industrial case study of performance and cost design space exploration," in *Proceedings of the third joint ACM/SPEC international conference on Performance Engineering (ICPE 2012)*, L. Kurian John and D. Krishnamurthy, Eds., New York, NY, USA, 2012.

[25] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-Based Reliability Prediction with the Palladio Component Model," *IEEE Trans. on Softw. Eng.*, 2012.

[26] S. Kounev, "Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets," *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 486–502, July 2006.

[27] S. Kounev, K. Bender, F. Brosig, N. Huber, and R. Okamoto, "Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics," in *4th Intl. ICST Conference on Simulation Tools and Techniques (SIMUTools)*, Mar. 2011.

[28] S. Kounev, K. Sachs, J. Bacon, and A. Buchmann, "A Methodology for Performance Modeling of Distributed Event-Based Systems," in *11th IEEE Intl. Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 13–22.

[29] R. Nou, S. Kounev, F. Julia, and J. Torres, "Autonomic QoS control in enterprise Grid environments using online simulation," *Journal of Systems and Software*, vol. 82, no. 3, pp. 486–502, Mar. 2009.

[30] S. Spinner, S. Kounev, and P. Meier, "Stochastic Modeling and Analysis using QPME: Queueing Petri Net Modeling Environment v2.0," in *Proceedings of the 33rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2012)*, ser. Lecture Notes in Computer Science (LNCS), S. Haddad and L. Pomello, Eds., vol. 7347. Berlin, Heidelberg: Springer-Verlag, June 2012, pp. 388–397. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31131-4_21

[31] P. Meier, "Automated Transformation of Palladio Component Models to Queueing Petri Nets," Master's thesis, Karlsruhe Institute of Technology (KIT), 2010.

[32] G. Bolch, Ed., *Queueing networks and Markov chains : Modeling and Performance Evaluation with Computer Science Applications*, 2nd ed. Hoboken, NJ: Wiley, 2006.

[33] G. Franks, "Performance analysis of distributed server systems," Ph.D. dissertation, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, December 1999.

[34] G. Franks, T. Omari, C. M. Woodside, O. Das, and S. Derisavi, "Enhanced modeling and solution of layered queueing networks," *IEEE Trans. on Software Engineering*, vol. 35, no. 2, pp. 148–161, 2009.

[35] J. A. Rolia and K. C. Sevcik, "The method of layers," *IEEE Trans. Softw. Eng.*, vol. 21, no. 8, pp. 689–700, 1995.

[36] G. Franks, "Simulating layered queueing networks with passive resources," in *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, ser. TMS-DEVS '11. San Diego, CA, USA: Society for Computer Simulation International, 2011, pp. 8–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=2048476.2048477

[37] A. Martens, H. Koziolek, L. Prechelt, and R. Reussner, "From monolithic to component-based performance evaluation of software architectures," *Empirical Software Engineering*, vol. 16, no. 5, pp. 587–622, 2011.

[38] D. Menasce, V. Almeida, and L. Dowdy, *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*. New Jersey: Prentice-Hall, Mar. 1994, $44.

[39] H. Koziolek, S. Becker, and J. Happe, "Predicting the Performance of Component-based Software Architectures with different Usage Profiles," in *Proc. 3rd International Conference on the Quality of Software Architectures (QoSA'07)*, ser. LNCS, vol. 4880. Springer, Juli 2007, pp. 145–163.

[40] X. Wu and M. Woodside, "Performance Modeling from Software Components," in *Proc. 4th International Workshop on Software and Performance (WOSP'04)*, vol. 29, no. 1. New York, NY, USA: ACM Press, 2004, pp. 290–301.

[41] S. Kounev, S. Spinner, and P. Meier, "QPME 2.0 - A Tool for Stochastic Modeling and Analysis Using Queueing Petri Nets," in *From Active Data Management to Event-Based Systems and More*, ser. LNCS, K. Sachs, I. Petrov, and P. Guerrero, Eds. Springer, 2010, vol. 6462, pp. 293–311.

[42] S. Kounev and A. Buchmann, "SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation," *Performance Evaluation*, vol. 63, no. 4-5, pp. 364–394, May 2006.

[43] K. Sachs, S. Kounev, and A. Buchmann, "Performance Modeling and Analysis of Message-oriented Event-driven Systems," *Journal of Software and Systems Modeling (SoSyM)*, October 2011.

[44] D. C. Petriu, C. M. Woodside, D. B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J. M. Bieman, S. H. Houmb, and J. Jürjens, "Performance analysis of security aspects in uml models," in *WOSP '07: Proceedings of the 6th international workshop on Software and performance*. New York, NY, USA: ACM, 2007, pp. 91–102.

[45] Object Management Group (OMG), "UML Profile for Schedulability, Performance and Time," http://www.omg.org/cgi-bin/doc?formal/2005-01-02, 2005, last retrieved 2008-01-13. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2005-01-02

[46] ——, "UML Profile for MARTE, Beta 1," http://www.omg.org/cgi-bin/doc?ptc/2007-08-04, August 2007, last retrieved 2008-01-13. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/2007-08-04

[47] A. Di Marco and P. Inverardi, "Compositional generation of software architecture performance QN models," in *Proc. 4th Working IEEE/IFIP Int. Conference on Software Architecture (WICSA'04)*. IEEE, 2004, pp. 37–46.

[48] S. Balsamo, M. Marzolla, and R. Mirandola, "Efficient performance models in component-based software engineering," in *EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 64–71.

[49] S. Distefano, M. Scarpa, and A. Puliafito, "From uml to petri nets: The pcm-based methodology," *IEEE Trans. Softw. Eng.*, vol. 37, pp. 65–79, January 2011.

[50] S. A. Hissam, G. A. Moreno, J. A. Stafford, and K. C. Wallnau, "Packaging Predictable Assembly," in *Proc. IFIP/ACM Working Conference on Component Deployment (CD'02)*. London, UK: Springer-Verlag, 2002, pp. 108–124.

[51] E. Bondarev, J. Muskens, P. de With, M. Chaudron, and J. Lukkien, "Predicting Real-Time Properties of Component Assemblies: A Scenario-Simulation Approach," in *Proc. 30th EUROMICRO Conf. (EUROMICRO'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 40–47.

[52] A. Bertolino and R. Mirandola, "Cb-spe tool: Putting component-based performance engineering into practice," in *Proc. 7th Int. Symposium on Component-based Software Engineering (CBSE'04)*, ser. LNCS, no. 3054. Springer, May 2004, pp. 233–248.

[53] M. Sitaraman, G. Kuczycki, J. Krone, W. F. Ogden, and A. Reddy, "Performance specification of software components," in *Proc. of SSR '01*, 2001.

[54] Y. Liu, A. Fekete, and I. Gorton, "Design-level performance prediction of component-based applications," *IEEE Trans. Softw. Eng.*, vol. 31, no. 11, pp. 928–941, November 2005, member-Yan Liu and Member-Alan Fekete and Member-Ian Gorton.

[55] D. Hamlet, "Tools and experiments supporting a testing-based theory of component composition," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, pp. 12:1–12:41, June 2009.

[56] S. Röttger and S. Zschaler, "Tool support for refinement of non-functional specifications," *Journal on Software and Systems Modelling (SoSyM)*, vol. 6, no. 2, Jun. 2007.

[57] A. DiMarco and R. Mirandola, "Model transformations in software performance engineering," in *Quality of Software Architectures, 2nd International Conference, QoSA 2006, Västerås, Sweden, June 27 - 29, 2006, Proceedings*, ser. Lecture Notes in Computer Science, C. Hofmeister, I. Crnkovic, R. Reussner, and S. Becker, Eds., vol. 4214, 2006, pp. 95–110.

[58] C. U. Smith, C. M. Llado, V. Cortellessa, A. D. Marco, and L. G. Williams, "From UML models to software performance results: an SPE process based on XML interchange formats," in *Proc. 5th international workshop on Software and performance (WOSP'05)*. New York, NY, USA: ACM Press, 2005, pp. 87–98.

[59] C. Smith, C. Lladó, and R. Puigjaner, "Performance Model Interchange Format (PMIF 2): A comprehensive approach to Queueing Network Model interoperability," *Performance Evaluation*, vol. 67, no. 7, pp. 548–568, 2010.