Family-Based Performance Analysis of Variant-Rich Software Systems

Matthias Kowal[1], Ina Schaefer[1], and Mirco Tribastone[2]

[1] Technische Universität Braunschweig, Germany
[2] University of Southampton, United Kingdom

Abstract. We study models of software systems with variants that stem from a specific choice of configuration parameters with a direct impact on performance properties. Using UML activity diagrams with quantitative annotations, we model such systems as a product line. The efficiency of a *product-based* evaluation is typically low because each product must be analyzed in isolation, making difficult the re-use of computations across variants. Here, we propose a *family-based* approach based on symbolic computation. A numerical assessment on large activity diagrams shows that this approach can be up to three orders of magnitude faster than product-based analysis in large models, thus enabling computationally efficient explorations of large parameter spaces.

1 Introduction

User-configurable parameters of software systems often have a critical impact on non-functional properties such as performance and energy consumption. For example, an elastic cloud-based application may dynamically change the number of active virtual machines depending on the current workload conditions, by setting appropriate threshold-based rules in the virtualization framework. In embedded systems, such as those employed in automation, there may be a trade-off between energy consumption due to increasing speeds and the capability to process jobs with given time constraints (e.g., [1]).

Run-time analysis can be conveniently employed to find the optimal configuration of parameters to automatically adapt to changing conditions. For instance, workloads on a web server may be subjected to day/night or weekday/weekend patterns. In this case, the availability of a software model can be particularly beneficial: When a new operating condition is detected in the system, a mathematical problem can be defined, whose solution gives the values of the user-tunable parameters that satisfy certain given criteria of optimality with respect to the current situation, for instance a performace/cost trade off. Typical solution methods essentially involve repeated analyses using different feasible configurations. This makes the analysis difficult when the evaluation of a single configuration is expensive and/or when the parameter space is large. In particular, applicability of the approach at run-time may be severely hindered if the time constraints for re-configurations and adaptations are stringent enough.

S. Gnesi and A. Rensink (Eds.): FASE 2014, LNCS 8411, pp. 94–108, 2014.

This paper proposes a framework for the efficient evaluation of software designs with large parameter spaces. We consider a class of UML activity diagrams (ADs) to model systems which can be reasonably described as workflow processes, such as real-world data-centers [2], service-oriented architectures (e.g., [3]), and automation systems [4]. To capture non-functional properties, ADs are augmented with performance-related annotations (such as the duration to execute an activity of an activity node) and interpreted as continuous-time Markov chains, along the lines of established routes in model-driven software performance engineering; see, for instance, [5] for a review and [6], [7], and [8], for related work about the automatic extraction of performance models from ADs. Our *performance-annotated ADs* (PAADs) are integrated with software product-line techniques to precisely capture variability aspects. More specifically, we consider a *delta-oriented* approach, where possibly many variants can be generated as a result of applying changes (i.e., *deltas*) to a *core* PAAD [9]. This is a novel application of delta modeling, which has so far been used to represent static variability of software architectures [10] and Java programs [11].

A straightforward solution technique requires a separate analysis of each variant—the so-called *product-based* (PB) evaluation. In this paper, we consider a *family-based* (FB) approach [12]. The configurable parameters of the model under study, inferred by the kinds of delta operations defined on the AD, are used for obtaining a solution in *symbolic* form. In this way, performance indices can be simply obtained by evaluating a polynomial expression that explicitly depends on the configurable parameters. The evaluation may become faster than the PB analysis, which is based on the numerical inversion of a matrix of size equal to the number of nodes in the PAAD. By numerical experimentation we show that our FB approach is up to three orders of magnitude faster than PB analysis, with a tendency to become increasingly more convenient as the model size grows. Although family-based product line analyses have been introduced for type checking [13–15] and model checking [16–18], for the first time this approach is considered for the efficient performance modeling of product lines.

Related work. This paper is most closely related to [19], where an approach based on parametric probabilistic model checking of software product lines models as annotated UML sequence diagrams is proposed. This leads to a symbolic expression that encodes the dependence of certain properties of interest from variables, in a manner which is analogous to ours. However, our work is different in that we consider properties of performance as opposed to reliability/energy. While, in principle, the model checking algorithm of [19] is also applicable to performance-related properties, it may not be efficient. This is due to the potentially massive state space size involved in typical performance models, which generally consider contention for resources by many users, unlike the single-user model in [19]. Under these conditions, the state space size grows (at worst) exponentially with the number of users, which can make symbolic computation infeasible. In our work, instead, this problem is basically circumvented by observing that the classes of models of our interest admit an efficient solution technique that does not necessitate state-space enumeration altogether.

In [20, 21], UML-annotated software product line designs are translated into layered queueing networks [22] and solved with a PB analysis. While layered networks are more expressive than our model, as, e.g. they capture simultaneous resource possession, they do not admit an efficient FB analysis.

Paper outline. Section 2 introduces Performance Annotated Activity Diagrams (PAADs). In Section 3, PAADs are integrated with the delta modeling approach to handle variability. In Section 4, we present our FB analysis. The experimental comparison between the FB and PB evaluation is reported in Section 5, followed by concluding remarks in Section 6.

2 Foundations

Graphically, a UML AD is visualized as a multipartite directed graph. The elements of interest in our modeling framework are categorized into three different groups: *action nodes* essentially describe the smallest possible event in an activity; *activity edges* connect the nodes, expressing possible paths of execution; and *control flow nodes* are used to model, for instance, conditional behavior. Throughout the paper, we shall use the neutral name of *job* to indicate an element (i.e., a *token*) that circulates through the nodes of an activity diagram. This is to be interpreted, e.g., as a user, service request, or a physical item, depending on the context of the specific model under consideration.

The following provides a formal definition of performance-annotated activity diagrams in a manner that is independent from the actual annotation mechanism that may be used in an implementation. A concrete realization is feasible, for instance with the MARTE profile (see [23]) and its PaStep stereotype (e.g., [24]).

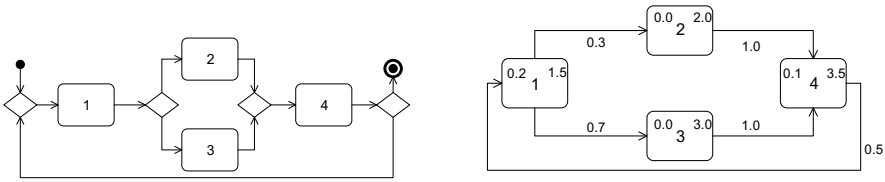
Definition 1 (Performance-Annotated Activity Diagram). *Let $\mathcal{V}$ be the set of all nodes. A* performance-annotated activity diagram *(PAAD) is a tuple*

$$PAAD = (V, E, \lambda, \mu),$$

where $V \subseteq \mathcal{V}$, $E \subseteq V \times \mathbb{R}_{\geq 0} \times V$, $\lambda : V \to \mathbb{R}_{\geq 0}$, and $\mu : V \to \mathbb{R}_{>0}$.

This definition specifies a directed graph annotated with three distinct pieces of information. Each edge $e \in E$ has a non-negative real, giving the probability with which that path is taken by a job in the source node. Each node $v \in V$ is associated with a *rate*, $\mu(v)$, denoting the average speed at which a job is processed in v; $\lambda(v)$, instead, denotes the *workload*, the speed at which jobs arrive from the external world. This may model, for instance, users that issue invocations to the service described by the AD.

We wish to point out that Definition 1 does not explicitly consider initial, final, and merge nodes. Similarly to previous work [24], we argue that these are not necessary when an AD is to be interpreted as a performance model. For instance, Figure 1 shows a sample AD (left diagram), and its representation in our annotated PAAD format (right diagram), removing the nodes that are not supported and redirecting the edges appropriately. For instance, the initial



(a) Sample Activity Diagram **(b)** Annotated PAAD

Fig. 1. Running example

node, which models the start of the activity, is replaced by a nonzero workload into node 1. This PAAD will be used as a running example throughout this paper. For each action node v, the top-left label gives $\lambda(v)$, the top-right label is $\mu(v)$, whereas the label in the middle indicates the node itself. The edges are instead labeled with their associated probabilities. For example, after an element is processed in node 4, it will return to node 1 with probability 0.5. Thus, with probability 0.5 it will leave the system. This captures the intended design in the AD diagram, where one outgoing edge of the decision node leads to a final state.

Of all the elements of UML ADs that are not used in Definition 1, we remark the absence of fork/join nodes. Unfortunately, their performance interpretation leads to models that do not enjoy efficient FB solutions. Although we leave further investigation of this matter to future work, we observe that Definition 1 allows us to capture models considered in the literature such as service-oriented systems (e.g., [25–27]) and manufacturing job-shops [1, 28].

We now provide conditions to be enjoyed by a PAAD to yield a meaningful performance model.

Definition 2 (Well-formedness). *A PAAD is well-formed if and only if the following conditions hold:*

i) There exists at least one $v \in V$ such that $\lambda(v) > 0$;
ii) For all $v \in V$ it holds that $\sum_{(v,p,v') \in E} p \leq 1$;
iii) For all $v, v' \in V$, for any $(v, p, v'), (v, q, v') \in E$ it holds that $p = q$;
iv) There exists at least one $v \in V$ such that $\sum_{(v,p,v') \in E} p < 1$.

Assumption i) is required to ensure that the model receives requests starting at least from one node. Assumption ii) corresponds to the natural interpretation of edge labels as probabilities. Assumption iii) requires that there is at most one directed edge between any two nodes, so that the probability with which node v' is visited after v is not ambiguously defined. Finally, iv) requires that, eventually, jobs leave the workflow. This is a necessary condition for a steady-state behaviour. Otherwise, the system would keep accumulating jobs.

The following definition permits the analysis of a (well-formed) PAAD. We call this *product-based evaluation* as it concerns a given, concrete PAAD, unlike *family-based evaluation*, which will be introduced later in the paper.

Definition 3 (Product-based evaluation). *The product-based evaluation of a PAAD is given by the following system of linear equations*

$$(I - P^T)\gamma = \lambda, \tag{1}$$

where I is the identity matrix, λ is the (column) vector with elements $\lambda(v)$. This is ordered in the same way as nodes appear in the matrix P, which is defined as $P = (p_{v,v'})$, for all $v, v' \in V$ with

$$p_{v,v'} = \begin{cases} p & \text{if } (v, p, v') \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, γ is the vector of unknowns, with elements denoted by $\gamma(v)$.

In essence, we are interpreting a PAAD as a continuous-time Markov chain that underlies a Jackson-type queueing network [28], by giving the following semantics. $\lambda(v)$ is the rate of arrival of jobs at the node v, which is assumed to be a Poisson process. Thus, for $\lambda(v) > 0$, $1/\lambda(v)$ is the inter-arrival time between two jobs, which is exponentially distributed. If $\lambda(v) = 0$, node v does not have exogenous arrivals and may only process jobs arriving from other nodes, according to the topology of the workflow. Jobs at action node v are served by a processing unit with rate $\mu(v) > 0$. Therefore, $1/\mu(v)$ is the average service demand of a job at node v. When the node is busy serving a job, the other jobs accumulate in an unbounded queue and are scheduled according to a first-in first-out discipline. P is the *routing probability matrix*, defining with which probability a job in node v, after being serviced, moves to any other node v'. Finally, γ gives the *effective* arrival rates, which take into account the actual traffic incoming at node v due to the exogenous arrival as well as to the feedback from other nodes. With obvious ordering, in our running example we have

$$P = \begin{bmatrix} 0.0 & 0.3 & 0.7 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.5 & 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad \lambda = \begin{bmatrix} 0.2 \\ 0.0 \\ 0.0 \\ 0.1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 1.5 \\ 2.0 \\ 3.0 \\ 3.5 \end{bmatrix} \tag{2}$$

Once the system (1) is solved for γ, the steady-state behavior of the network is completely characterized (e.g., [29]). Specifically, the following indices can be computed for any $v \in V$.

- $\gamma(v)$ is the *throughput*, i.e., the rate at which jobs are served at node v.
- The *utilization* of node v, denoted by $\rho(v)$, i.e., the probability that the node is busy serving jobs, is given by:

$$\rho(v) = \gamma(v)/\mu(v).$$

- The *queue length* at node v, denoted by $L(v)$, i.e., the number of jobs at node v including those in service, is given by

$$L(v) = \rho(v)/(1 - \rho(v)). \tag{3}$$

– Using Little's law, the *average response time* of jobs at node v, denoted by $W(v)$, is given by

$$W(v) = L(v)/\gamma(v).$$

Response times along paths of the PAAD can be computed similarly using the same law.

In our numerical evaluation in Section 5, we will consider the average queue length in the network as the metric of interest. This is simply obtained by computing $\sum_{v \in V} L(v)/|v|$. For instance, solving (2) yields

$$\gamma = \begin{bmatrix} 0.50 & 0.15 & 0.35 & 0.60 \end{bmatrix}^T$$

which leads to an average queue length of 0.23 customers in the steady state.

3 Variability of PAADs

In this section, we discuss how to model variability aspects of PAADs using delta modeling. Delta modeling is a modular, yet flexible variability modeling approach on the artifact level (in contrast to feature models which live on the requirements level) and allows capturing closed as well as open variant spaces. We consider a core PAAD and an associated set of *deltas* [9]. Each delta contains a set of basic operations to be performed on a PAAD, such as the addition and the removal of a node, or the modification of parameters such as the probabilities of an edge. Thus, applying a delta to the core yields a new PAAD *variant*, which has performance characteristics that can be numerically analyzed using the product-based evaluation in Definition 3.

In addition, from a core model with an associated set of deltas we generate a *150%-model*. This is an over-saturated variant representing the whole product line which, in general, does not correspond to a concrete variant of interest to the modeler. However, we define a solution method based on symbolic computation yielding an expression that directly relates a performance index (such as the average queue length) to all the model parameters that are at least altered once by any of the deltas. When such a symbolic expression is evaluated for the parameters of a specific variant, it returns the actual performance index for that variant. This allows the re-use of the same symbolic expression *for all variants*, unlike the numerical solution with product-based evaluation.

We start with defining all possible atomic delta operations on a PAAD.

Definition 4 (PAAD deltas). *A PAAD delta is a set of delta operations $\delta \subseteq Op$, where*

$$\begin{aligned} Op = {} & \{\mathsf{add}\ (v_i, \lambda_i, \mu_i) \mid v_i \in \mathcal{V}, \lambda_i \geq 0, \mu_i \geq 0\} \\ & \cup \{\mathsf{add}\ (v_i, p_{ij}, v_j) \mid v_i, v_j \in \mathcal{V}, p_{ij} > 0\} \cup \{\mathsf{rem}\ v \mid v \in \mathcal{V}\} \cup \{\mathsf{rem}\ e \mid e \in \mathcal{E}\} \\ & \cup \{\mathsf{mod}\ \lambda(v_i)\ \mathsf{by}\ \lambda_j \mid v_i \in \mathcal{V}, \lambda_j \geq 0\} \cup \{\mathsf{mod}\ \mu(v_i)\ \mathsf{by}\ \mu_j \mid v_i \in \mathcal{V}, \mu_j > 0\} \\ & \cup \{\mathsf{mod}\ (v_i, p_{ij}, v_j)\ \mathsf{by}\ q_{ij} \mid (v_i, p_{ij}, v_j) \in \mathcal{E}, q_{ij} > 0\}. \end{aligned}$$

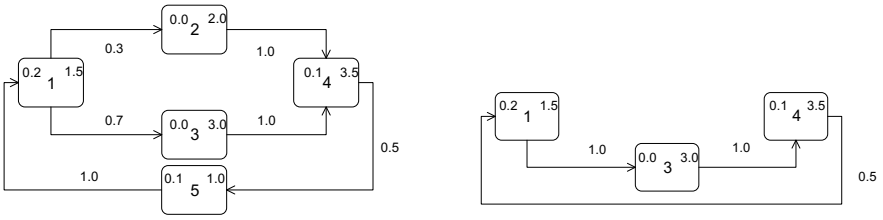
When an edge is added, its associated probability must be strictly positive. This is without loss of generality because an edge with probability zero essentially corresponds to the case where no edge at all is connecting two nodes. For the same reason, when a service rate or a probability are modified we require strictly positive values. This is not the case for arrival rates, so long as there exists at least a node with a positive arrival rate. The modification of values is a simplification since it can also be encoded by removing node or edge and adding it with the desired rate or probability, respectively.

For simplicity, in this paper we only consider a single delta to generate a PAAD variant. This is without loss of generality, since the effect of a set of deltas can be combined into a single composite delta by defining an appropriate delta composition operation. In order to ensure that the application of such a delta leads to a well-formed PAAD variant, we require that the delta is applicable and consistent. A delta is *applicable* to a PAAD if all elements (node, edge, rate or probability) which should be removed or modified exist and if all elements which are added do not exist. A delta is *consistent* if it adds, removes or modifies each element at most once [9]. Furthermore, a consistent delta ensures that are no dangling edges in the resulting PAAD. This means that removing a node also causes the removal of all resulting edges. Edges are never added between nodes that are removed in the delta. If a node of an added edge does not exist in the core PAAD, the necessary source and/or target edges are also added in the delta. However, there may be unreachable nodes in the resulting PAAD variant. This is not an issue for the well-formedness of the result.

As an example of delta modeling on PAADs, we consider deltas δ_1 and δ_2, defined as follows:

$$\delta_1 = \{\mathsf{rem}\ (4, 0.5, 1), \mathsf{add}\ (5, 0.1, 1.0), \mathsf{add}\ (4, 0.5, 5), \mathsf{add}\ (5, 1.0, 1)\},$$
$$\delta_2 = \{\mathsf{rem}\ (1, 0.3, 2), \mathsf{rem}\ (2, 1.0, 4), \mathsf{rem}\ 2, \mathsf{mod}\ (1, 0.7, 3)\ \mathsf{by}\ 1.0\}.$$

The following definition formalizes, in a straightforward way, how to obtain a variant by applying a delta to a PAAD. Figure 2 illustrates the application of δ_1 and δ_2 to the core model in Fig. 1.



(a) δ_1 application **(b)** δ_2 application

Fig. 2. Deltas applied to the core model in Figure 1

Definition 5 (PAAD delta application). *The application of an applicable and consistent delta $\delta \subseteq Op$ to a $PAAD = (V, E, \lambda, \mu)$ is defined by the function $PAAD' = apply(PAAD, \delta)$, where $PAAD' = (V', E', \lambda', \mu')$. It is recursively defined as follows.*

1. *Case $\delta = \emptyset$: $PAAD' = PAAD$.*
2. *Case: $\delta = \delta' \cup \delta'' \wedge \delta', \delta'' \in Op$: $PAAD' = apply(apply(PAAD, \delta'), \delta'')$.*
3. *Case: $\delta = \textsf{add}\ (v_i, \lambda_i, \mu_i)$:*

$$V' = V \cup \{v_i\} \quad \lambda'(v) = \begin{cases} \lambda(v) & \text{if } v \neq v_i, \\ \lambda_i & \text{if } v = v_i, \end{cases} \quad \mu'(v) = \begin{cases} \mu(v) & \text{if } v \neq v_i, \\ \mu_i & \text{if } v = v_i. \end{cases}$$

4. *Case: $\delta = \textsf{add}\ (v_i, p_{ij}, v_j)$: $E' = E \cup \{(v_i, p_{ij}, v_j)\}$.*
5. *Case: $\delta = \textsf{rem}\ v$: $V' = V \setminus \{v\}$.*
6. *Case: $\delta = \textsf{rem}\ e$: $E' = E \setminus \{e\}$.*
7. *Case: $\delta = \textsf{mod}\ \lambda(v_i)\ \textsf{by}\ \lambda_j$: $\lambda'(v) = \begin{cases} \lambda(v) & \text{if } v \neq v_i, \\ \lambda_j & \text{if } v = v_i, \end{cases}$*
8. *Case: $\delta = \textsf{mod}\ \mu(v_i)\ \textsf{by}\ \mu_j$: $\mu'(v) = \begin{cases} \mu(v) & \text{if } v \neq v_i, \\ \mu_j & \text{if } v = v_i, \end{cases}$*
9. *Case: $\delta = \textsf{mod}\ (v_i, p_{ij}, v_j)\ \textsf{by}\ q_{ij}$: $E' = \big(E \setminus \{(v_i, p_{ij}, v_j)\}\big) \cup \{(v_i, q_{ij}, v_j)\}$.*

We now consider a *core* PAAD and a set of deltas Δ. We define the 150%-model as a special kind of PAAD which has all nodes and transitions that are introduced or modified by each $\delta \in \Delta$. As discussed, in general the 150%-model is not a valid PAAD variant, but it contains all the information to retrieve a variant resulting from the application of any $\delta \in \Delta$. The origin of a node or transition from the core model or a specific delta, where it is added, modified or removed, is traced by means of a labeling function $\mathcal{L}$, defined as follows.

Definition 6 (150%-model). *Let $PAAD_c = (V_c, E_c, \lambda_c, \mu_c)$ be the* core model *and Δ be a set of consistent and applicable deltas. Let $L = \{C\} \cup \{\underline{\delta}, \underline{!\delta} \mid \delta \in \Delta\}$, with $C \notin \Delta$, be the set of labels. The* 150%-model *is $PAAD_{150} = (V_{150}, E_{150}, \lambda_{150}, \mu_{150}, \mathcal{L})$, where:*

$V_{150} = V_c \cup \{v \mid \exists \delta \in \Delta : \textsf{add}\ (v, \lambda_i, \mu_i) \in \delta\}$,

$E_{150} = E_c \cup \{(v_i, p_{ij}, v_j) \mid \exists \delta : \textsf{add}\ (v_i, p_{ij}, v_j) \in \delta \vee \textsf{mod}\ (v_i, q_{ij}, v_j)\ \textsf{by}\ p_{ij} \in \delta\}$,

λ_{150} and μ_{150} are partial functions of $V_{150} \times L$ defined as

$$\lambda_{150} : V_{150} \times L \to \mathbb{R}_{\geq 0},\ \lambda_{150}(v, l) = \begin{cases} \lambda_c(v) & \text{if } l = C \wedge v \in V_c, \\ \lambda_i & \text{if } l = \underline{\delta} \wedge \textsf{add}\ (v, \lambda_i, \mu_i) \in \delta, \\ \lambda_j & \text{if } l = \underline{\delta} \wedge \textsf{mod}\ \lambda(v_i)\ \textsf{by}\ \lambda_j \in \delta, \\ 0 & \text{if } l = \underline{!\delta} \wedge \textsf{rem}\ v \in \delta, \end{cases}$$

$$\mu_{150} : V_{150} \times L \to \mathbb{R}_{\geq 0},\ \mu_{150}(v, l) = \begin{cases} \mu_c(v) & \text{if } l = C \wedge v \in V_c, \\ \mu_i & \text{if } l = \underline{\delta} \wedge \textsf{add}\ (v, \lambda_i, \mu_i) \in \delta, \\ \mu_j & \text{if } l = \underline{\delta} \wedge \textsf{mod}\ \mu(v_i)\ \textsf{by}\ \mu_j \in \delta, \\ 0 & \text{if } l = \underline{!\delta} \wedge \textsf{rem}\ v \in \delta, \end{cases}$$

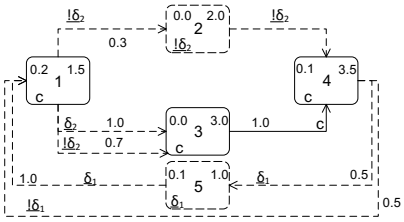


Fig. 3. 150%-model of the running example

and $\mathcal{L}$ *is the* labeling function *defined as*

$$\mathcal{L} : V_{150} \cup E_{150} \to 2^L,$$

$$\mathcal{L}(v) = \begin{cases} C & \text{if } v \in V_c, \\ \emptyset & \text{otherwise,} \end{cases} \cup \{\underline{\delta} \mid \mathit{add}\ (v, \lambda_i, \mu_i) \in \delta\} \cup \{\underline{!\delta} \mid \mathit{rem}\ v \in \delta\}.$$

$$\mathcal{L}(e) = \begin{cases} C & \text{if } e \in E_c, \\ \emptyset & \text{otherwise,} \end{cases} \cup \{\underline{\delta} \mid \mathit{add}\ e \in \delta\} \cup \{\underline{!\delta} \mid \mathit{rem}\ e \in \delta\}$$
$$\cup \{\underline{\delta} \mid \mathit{mod}\ (v_i, q_{ij}, v_j)\ \mathit{by}\ p_{ij} \in \delta \wedge e = (v_i, p_{ij}, v_j)\}$$
$$\cup \{\underline{!\delta} \mid \mathit{mod}\ (v_i, q_{ij}, v_j)\ \mathit{by}\ p_{ij} \in \delta \wedge e = (v_i, q_{ij}, v_j)\}.$$

In order to construct the 150%-model, we consider all nodes V_{150} which are either part of the core PAAD or are added in a delta. The set of edges E_{150} contains all edges from the core and all edges added by a delta. Since the probability of an existing edge can be modified by a delta, we add an edge with the new probability to the 150%-model. As a result, we have an edge with the previous probability and an edge with the modified one in the 150%-model. The domain of the functions λ_{150} and μ_{150} are pairs, where the first element indicates the node or edge that is labelled and the second pair specifies a delta label. The functions map onto the concrete value of the rate that the element has in that specific delta. Finally, the labeling function $\mathcal{L}$ is necessary in order to identify the core and the original PAAD variants in order to map the results of the FB analysis to the PAAD variants. Nodes have three possible labels: C means that the node is part of the core PAAD. Since deltas add or remove nodes, we use $\underline{\delta}$ to denote addition, and $\underline{!\delta}$ for removal. The labeling of edges is done in a similar way. The 150%-model of the running example is shown in Fig. 3. Nodes and edges occur only in the core model, e.g., $\mathcal{L}(v) = \{C\}$, are marked with solid lines; otherwise, dashed lines are used. The labels of the nodes are shown within the nodes in the bottom-left part.

4 Family-Based Evaluation

We now discuss the symbolic evaluation of the 150% model. This is accomplished, in essence, by taking a 150% model and associating a symbolic variable to each element that is varied in at least one delta. We use the following convention. We let $\mathcal{S}$ denote the set of all symbolic variables, whose elements are indicated by a superscript '$*$'.

Definition 7 (Family-based evaluation). *Let $PAAD_{150}$ be a 150% model. The* FB evaluation *is given by the solution of*

$$(I - P_s^T)\gamma_s = \lambda_s, \tag{4}$$

where:

$$\lambda_s : V_{150} \to \mathbb{R} \cup \mathcal{S},\ \lambda_s(v) = \begin{cases} \lambda_{150}(v, C) & \text{if } \nexists l \in L \setminus \{C\} : \lambda_{150}(v, l) \text{ is defined.} \\ \lambda_v^* & \text{otherwise,} \end{cases}$$

$$\mu_s : V_{150} \to \mathbb{R} \cup \mathcal{S},\ \mu_s(v) = \begin{cases} \mu_{150}(v, C) & \text{if } \nexists l \in L \setminus \{C\} : \mu_{150}(v, l) \text{ is defined.} \\ \mu_v^* & \text{otherwise,} \end{cases}$$

$$P_s = (p^s_{v,v'})_{v,v' \in V_{150}},\ p^s_{v,v'} = \begin{cases} q & \text{if } \exists e = (v, q, v') \in E_{150} \wedge \mathcal{L}(e) = \{C\}, \\ 0 & \text{if } \nexists e = (v, q, v') \in E_{150}, \\ p^*_{v,v'} & \text{otherwise.} \end{cases}$$

Informally, $\lambda_s(v)$ (and similarly, $\mu_s(v)$) treats as symbolic all the parameters that are changed by at least one delta operation. Else, the parameter is simply the concrete value assigned in the core model. Concrete probabilities $p_{v,v'}$ are assigned when two nodes are associated *only in the core model*, or when they are not associated at all. Otherwise, the symbolic variable $p^*_{v,v'}$ is used.

For an illustrative explanation, let us consider again our running example in Fig. 1 as core model and let us take $\Delta = \{\delta_a, \delta_b, \delta_c\}$, with

$$\delta_a = \{\mathsf{mod}\ (1, 0.3, 2)\ \mathsf{by}\ 0.4,\ \mathsf{mod}\ (1, 0.7, 3)\ \mathsf{by}\ 0.6\},$$
$$\delta_b = \{\mathsf{mod}\ \lambda(1)\ \mathsf{by}\ 0.1\}, \quad \delta_c = \{\mathsf{mod}\ \mu(4)\ \mathsf{by}\ 4.0\}.$$

For conciseness, we do not show the actual variants obtained through these deltas (which will have the same topology, but different concrete labels). By Definition 7, the FB evaluation is

$$P_s = \begin{bmatrix} 0.0 & p^*_{1,2} & p^*_{1,3} & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.5 & 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad \lambda_s = \begin{bmatrix} \lambda_1^* \\ 0.0 \\ 0.0 \\ 0.1 \end{bmatrix} \qquad \mu_s = \begin{bmatrix} 1.5 \\ 2.0 \\ 3.0 \\ \mu_4^* \end{bmatrix} \tag{5}$$

where $\mathcal{S} = \{p^*_{1,2}, p^*_{1,3}, \lambda^*_1, \mu^*_4\}$. As discussed in Section 2, P_s, λ_s, and μ_s characterize the performance of a PAAD. For instance, by using the formula (3), the

average queue length (AQL) will be given by the following symbolic expression:

$$\begin{aligned}\mathrm{AQL} = &-\frac{20\lambda_1^*+1}{80\lambda_1^*+60p_{1,2}^*+60p_{1,3}^*-116} - \frac{p_{1,2}^*(20\lambda_1^*+1)}{4(21p_{1,2}^*+20p_{1,3}^*+20\lambda_1^*p_{1,2}^*-40)} \\ &- \frac{10\lambda_1^*p_{1,2}^*+10\lambda_1^*p_{1,3}^*+1}{20\mu_4^*p_{1,2}^*-40\mu_4^*+20\mu_4^*p_{1,3}^*+40\lambda_1^*p_{1,2}^*+40\lambda_1^*p_{1,3}^*+4} \\ &- \frac{p_{1,3}^*(20\lambda_1^*+1)}{4(30p_{1,2}^*+31p_{1,3}^*+20\lambda_1^*p_{1,2}^*-60)}. \quad (6)\end{aligned}$$

Let us observe that assigning $p_{1,2}^* = 0.3$, $p_{1,3}^* = 0.7$, $\lambda_1^* = 0.2$, and $\mu_4^* = 3.5$ in the 150% model (5) corresponds to the model in (2), which is the product-based evaluation of a specific variant. Now, it holds that evaluating the symbolic expression (6) with such specific values yields 0.23, consistently with the product-based evaluation. The average queue lengths for the variants obtained by each of the deltas δ_a, δ_b, and δ_c, can be computed by evaluating *the same symbolic expression*, plugging in the appropriate concrete values related to each variant. Instead, as discussed, product-based evaluation does not allow the re-use of any computation because the solution based on matrix inversion is done numerically.

We are now left with showing that the symbolic evaluation with the appropriate concrete parameters of a variant always corresponds to the PB evaluation, i.e., the non-symbolic numerical analysis of a single variant in isolation. The following definition *concretizes* a 150% model with respect to a delta δ, i.e., it isolates the elements of the 150% model that are relevant for δ.

Definition 8 (Concretization). *Let $PAAD_{150}$ be a 150% model from a core $PAAD_c$ with a set of deltas Δ and with symbolic FB evaluation (4). A* concretization of $PAAD_{150}$ for $\delta \in \Delta$ *is given by* $(I - P_k^T)\gamma_k = \lambda_k$, *where*

$$\lambda_k = (\lambda_k(v))_{v\in V_{150}},\ \lambda_k(v) = \begin{cases}\lambda_{150}(v,\underline{\delta}) & \text{if defined,}\\ \lambda_{150}(v,C) & \text{if defined and } \lambda_{150}(v,\underline{\delta}) \text{ is not defined,}\\ 0 & \text{otherwise,}\end{cases}$$

$$P_k = (p_{v,v'}^k)_{v,v'\in V_{150}},\ p_{v,v'}^k = \begin{cases}p_{v,v'}^s & \text{if } p_{v,v'}^s \notin \mathcal{S},\\ p & \text{if } \exists e = (v,p,v') \in E_{150} \wedge \underline{\delta} \in \mathcal{L}(e),\\ 0 & \text{otherwise.}\end{cases}$$

The concretization yields a system of equations of the size of the 150% model. The next theorem is the desired, crucial result of consistency of this paper. It states that the FB symbolic solution, *restricted* to those nodes that are in the variant given by $apply(\mathrm{PAAD}_c, \delta)$, corresponds to the PB evaluation of $apply(\mathrm{PAAD}_c, \delta)$ itself.

Theorem 1 (Consistency). *Let $(I - P_a^T)\gamma_a = \lambda_a$ denote the PB evaluation of $(V_a, E_a, \lambda_a, \mu_a) = apply(PAAD_c, \delta)$, for $\delta \in \Delta$, and let $(I - P_k^T)\gamma_k = \lambda_k$ be the concretization of the 150% model $PAAD_{150}$ for δ. Furthermore, we define*

$$V^\delta = \left\{ v \in V_{150} : \left(C \in \mathcal{L}(v) \wedge !\underline{\delta} \notin \mathcal{L}(v)\right) \vee \underline{\delta} \in \mathcal{L}(v) \right\}.$$

It holds that i) $V^\delta = V^a$ and ii) $\gamma_a(v) = \gamma_k(v)$ and $\mu_a(v) = \mu_k(v)$, for all $v \in V^\delta$.

5 Numerical Experiments

We compared the FB symbolic analysis against the PB approach, where each configuration is solved numerically for a given concrete set of parameter values.

Our experimental set-up was as follows. We considered randomly generated 150% PAAD models with different numbers of nodes and different number of variables (i.e., the set of elements from $\mathcal{S}$) for their symbolic evaluation. This is motivated by the fact that, while PB evaluation has a cost which is at best quadratic with the number of nodes [29], the FB approach requires the evaluation of a polynomial expression. Thus, we wish to test the hypothesis that FB analysis is increasingly more convenient with larger networks, and to assess the impact of the number of variables on the length of such polynomials and thus on their evaluation time. Clearly, a trade-off must be struck between the degrees of freedom and the effectiveness of the FB approach, as the cost of such form of symbolic computation is clearly dependent on the number of variables [30]. Notice, however, that each symbol represents a parameter that takes values in the reals. Thus, even a *single variable* may represent *infinitely many variants*.

Let n denote the total number of nodes in the FB evaluation, i.e., $n = |V_{150}|$, p be the number of variables in the routing probability matrix (i.e., the number of symbols in P_s), m the number of variables in the service rates (the symbols in the range of the function in μ_s), and g the number of variables in the exogenous arrival rates (the symbols in the range of λ_s). For any given choice of (n, p, m, g), all the other parameters were randomly generated, with the service rates drawn uniformly at random in $[1.0; 20.0]$, and the arrival rates in the range $[0.0; 3.0]$. For instance, the symbolic evaluation (5) corresponds to a configuration $(n, p, m, g) = (4, 2, 1, 1)$, where the remaining concrete values shown in P_s, λ_s and μ_s would be generated randomly. In particular, the routing probability matrices generated in this way led to network topologies with densely connected nodes.

For each tuple (n, p, m, g), we considered 200 randomly generated variants which we analyzed with both the FB and the PB approach. We did so by randomly generating 200 tuples, each of length $p+m+g$, corresponding to a specific instantiation of the symbolic parameters. For the FB approach, each tuple was used to evaluate a symbolic expression of the average queue length such as (6); for the PB approach instead, the parameters were used to numerically solve the system of equations (1) for each variant.

We measured the wall-clock execution times for both FB and PB evaluation, repeated 5 times in order to reduce the noise in the measurements. The tests were implemented in Matlab version 7.9.0 (R2009b) using the *Symbolic Math Toolbox* for the FB evaluation, and the built-in functions for the solutions of systems of linear equations for the PB evaluation. The measurements were conducted on a machine with an Intel Core i7 2.66 GHz with 8 GB RAM.

Each line in Table 1 shows the overall execution times, averaged over the 5 tests, of both FB and PB across the 200 random variants, which represent the whole family for each configuration (n, p, m, g). We report the average speedup

Table 1. Numerical results

Variables				*Runtimes (s)*			
n	p	m	g	*FB*	*PB*	*PB/FB*	*PC*
4	1	0	0	0.011	0.049	4.47	0.545
4	0	1	0	0.004	0.043	10.78	0.185
4	0	0	1	0.009	0.045	4.92	0.190
4	1	1	1	0.011	0.046	4.13	0.214
4	2	2	2	0.014	0.049	3.60	0.319
4	4	4	4	0.020	0.049	2.43	0.310
24	1	0	0	0.011	0.066	5.96	1.141
24	0	1	0	0.004	0.063	14.60	1.124
24	0	0	1	0.011	0.067	6.29	1.152
24	1	1	1	0.013	0.068	5.04	1.244
24	2	2	2	0.016	0.068	4.31	1.100
24	0	6	0	0.007	0.065	9.94	1.004
24	4	4	4	0.099	0.070	0.70	1.904
142	1	0	0	0.016	6.540	397.34	5.425
142	0	1	0	0.005	8.107	1664.28	4.908
142	0	0	1	0.015	10.808	726.41	4.836
142	1	1	1	0.017	10.069	601.51	5.113
142	2	2	2	0.019	10.052	526.50	4.936
142	0	6	0	0.007	7.802	1191.79	5.137
142	4	4	4	0.026	10.985	429.83	4.942
302	1	0	0	0.019	19.186	1007.95	11.026
302	0	1	0	0.006	13.680	2292.46	11.192
302	0	0	1	0.018	13.399	728.73	11.050
302	1	1	1	0.021	19.520	909.12	11.591
302	2	2	2	0.024	19.459	820.30	11.214
302	0	6	0	0.006	13.896	2258.21	11.089
302	4	4	4	0.030	14.879	495.06	11.718

PB/FB. The last column shows the pre-computation (PC) time taken to symbolically solve (4). These results allow us to make the following observations.

- We confirm that for any fixed choice of p, m, and g that we considered, larger values of n make FB increasingly more convenient, with speedups up to over 2000; see, for instance, the configurations $(4, 0, 1, 0)$, $(24, 0, 1, 0)$, $(142, 0, 1, 0)$, $(4, 1, 0, 0)$, and $(302, 1, 0, 0)$. This is because of the increasing cost of the solution of the system of linear equations for PB, while FB solves it symbolically only once and off-line.
- For fixed n, varying p, m, and g has an impact on speedup, since the higher the number of variables the larger the closed-form polynomial expression derived by the FB approach.
- For fixed n, not all other parameters affect the speedup in the same manner. In particular, compare the two cases $p = m = g = 2$ and $p = g = 0$, $m = 6$, for every given n. Both have the same number of variables (i.e., six), but the latter case consistently enjoys a better speedup. This is because the m variables do not appear in (1), thus for $m = 6$ the symbolic expressions of the solution (1) can be greatly simplified because it consists of only scalars. (The m variables will appear in the calculation for the queue lengths L_v.)
- FB evaluation is not always more effective than PB evaluation. In our study, this has occurred in smaller models (i.e., $n = 24$) with relatively high number of variables. In these cases, the polynomial expression turned out to be more difficult to analyze than the linear system of equations (for which Matlab is well-known to be optimized).
- The pre-computation time behaves well with the number of variables, in particular with respect to the p variables that are used in the solution of (1).

6 Conclusion

We have presented an efficient technique for model-based performance analysis of software product lines using a class of UML activity diagrams annotated with quantitative information. Our approach enables a family-based evaluation by means of symbolic computation, which has been shown to be up to three orders of magnitude faster than product-based analysis in large models.

Regarding future work, this paper can be extended in two directions. From a theoretical viewpoint, we will study extensions to other kinds of performance models that are amenable to analogous closed-form symbolic solutions. From a practical viewpoint, we plan an implementation integrated with UML CASE tools and an experimentation with run-time optimization of automation systems.

Acknowledgment. This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future — Managed Software Evolution, and by the EU project QUANTICOL, 600708. The authors would like to thank Christian Prehofer for fruitful discussions.

References

1. Govil, M.K., Fu, M.C.: Queueing theory in manufacturing: A survey. Journal of Manufacturing Systems 18(3), 214–240 (1999)
2. Singh, R., Shenoy, P., Natu, M., Sadaphal, V., Vin, H.: Predico: A System for What-if Analysis in Complex Data Center Applications. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 123–142. Springer, Heidelberg (2011)
3. Huhns, M., Singh, M.: Service-oriented computing: key concepts and principles. IEEE Internet Computing 9(1), 75–81 (2005)
4. Vogel-Heuser, B., Witsch, D., Katzke, U.: Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. In: ICCA, pp. 1034–1039 (2005)
5. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Trans. Software Eng. 30(5), 295–310 (2004)
6. Petriu, D.C., Shen, H.: Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 159–177. Springer, Heidelberg (2002)
7. Balsamo, S., Marzolla, M.: Performance evaluation of UML software architectures with multiclass queueing network models. In: WOSP, pp. 37–42 (2005)
8. López-Grao, J.P., Merseguer, J., Campos, J.: From UML activity diagrams to stochastic Petri nets: application to software performance engineering. SIGSOFT Softw. Eng. Notes 29(1), 25–36 (2004)
9. Schaefer, I.: Variability modelling for model-driven development of software product lines. In: VaMoS, pp. 85–92 (2010)

10. Haber, A., Kutz, T., Rendel, H., Rumpe, B., Schaefer, I.: Delta-oriented architectural variability using monticore. In: ECSA, pp. 6:1–6:10 (2011)
11. Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N.: Delta-oriented programming of software product lines. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 77–91. Springer, Heidelberg (2010)
12. von Rhein, A., Apel, S., Kästner, C., Thüm, T., Schaefer, I.: The PLA model: on the combination of product-line analyses. In: VaMoS, pp. 14:1–14:8 (2013)
13. Apel, S., Kästner, C., Grösslinger, A., Lengauer, C.: Type safety for feature-oriented product lines. ASE 17(3), 251–300 (2010)
14. Delaware, B., Cook, W., Batory, D.: A Machine-Checked Model of Safe Composition. In: FOAL, pp. 31–35. ACM (2009)
15. Damiani, F., Schaefer, I.: Family-based analysis of type safety for delta-oriented software product lines. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 193–207. Springer, Heidelberg (2012)
16. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A., Raskin, J.F.: Model checking lots of systems: Efficient verification of temporal properties in software product lines. In: ICSE. IEEE (2010)
17. Lauenroth, K., Pohl, K., Toehning, S.: Model checking of domain artifacts in product line engineering. In: ASE, 269–280 (2009)
18. Asirelli, P., ter Beek, M.H., Gnesi, S., Fantechi, A.: Deontic logics for modeling behavioural variability. In: VaMoS, Essen, Germany, pp. 71–76 (January 2009)
19. Ghezzi, C., Sharifloo, A.M.: Verifying non-functional properties of software product lines: Towards an efficient approach using parametric model checking. In: SPLC, pp. 170–174 (2011)
20. Tawhid, R., Petriu, D.C.: Towards automatic derivation of a product performance model from a UML software product line model. In: WOSP, pp. 91–102 (2008)
21. Tawhid, R., Petriu, D.C.: Automatic derivation of a product performance model from a software product line model. In: SPLC, pp. 80–89 (2011)
22. Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE Trans. Software Eng. 35(2), 148–161 (2009)
23. Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). Beta 1. OMG, OMG document number ptc/07-08-04 (2007)
24. Tribastone, M., Gilmore, S.: Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In: WOSP, pp. 67–78 (2008)
25. D'Ambrogio, A., Bocciarelli, P.: A model-driven approach to describe and predict the performance of composite services. In: WOSP, pp. 78–89 (2007)
26. Menascé, D., Dubey, V.: Utility-based QoS brokering in service oriented architectures. In: ICWS, pp. 422–430 (July 2007)
27. Marzolla, M., Mirandola, R.: Performance prediction of web service workflows. In: Overhage, S., Ren, X.-M., Reussner, R., Stafford, J.A. (eds.) QoSA 2007. LNCS, vol. 4880, pp. 127–144. Springer, Heidelberg (2008)
28. Jackson, J.R.: Jobshop-like queueing systems. Management Science 10(1), 131–142 (1963)
29. Stewart, W.J.: Probability, Markov Chains, Queues, and Simulation. Princeton University Press (2009)
30. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: ICSE, pp. 341–350 (2011)