Efficient Parametric Model Checking Using Domain-Specific Modelling Patterns

Radu Calinescu
Department of Computer Science
University of York
United Kingdom
radu.calinescu@york.ac.uk

Kenneth Johnson
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
New Zealand
kenneth.johnson@aut.ac.nz

Colin Paterson
Department of Computer Science
University of York
United Kingdom
colin.paterson@york.ac.uk

ABSTRACT

We propose a parametric model checking (PMC) method that enables the efficient analysis of quality-of-service (QoS) properties of component-based systems. Our method builds on recent advances in PMC techniques and tools, and can handle large models by exploiting domain-specific modelling patterns for the software components. We precompute closed-form expressions for key QoS properties of such patterns, and handle system-level PMC by combining these expressions into easy-to-evaluate systems of equations.

ACM Reference Format:
Radu Calinescu, Kenneth Johnson, and Colin Paterson. 2018. Efficient Parametric Model Checking Using Domain-Specific Modelling Patterns. In *Proceedings of 40th International Conference on Software Engineering: New Ideas and Emerging Results Track, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE-NIER'18),* 4 pages.
https://doi.org/10.1145/3183399.3183404

1 INTRODUCTION

Parametric model checking (PMC) [5, 11, 13] is a technique for the analysis of Markov chains with transition probabilities specified as rational functions over a set of continuous variables. These variables may correspond to configurable parameters of a system modelled by the Markov chain, or may represent environment parameters unknown until run time. The properties of Markov chains analysed by PMC are formally expressed in probabilistic computation tree logic (PCTL) [12] extended with rewards [1], and the result of the analysis is an algebraic expression over the model variables.

PMC is supported by the leading model checkers PARAM [10], PRISM [14] and Storm [6]. We introduce a compositional PMC method (Section 2) that complements these state-of-the-art PMC techniques by exploiting domain-specific modelling patterns for the components of software systems. Our application of the new method to the service-based systems domain (Section 3) and preliminary experiments (Section 4) show reductions in PMC time of up to several orders of magnitude compared to PARAM, PRISM and Storm, and extends the range of models analysable by these tools (Section 5). We envisage that many other domains (discussed in Section 6) can similarly benefit from adopting our PMC method.

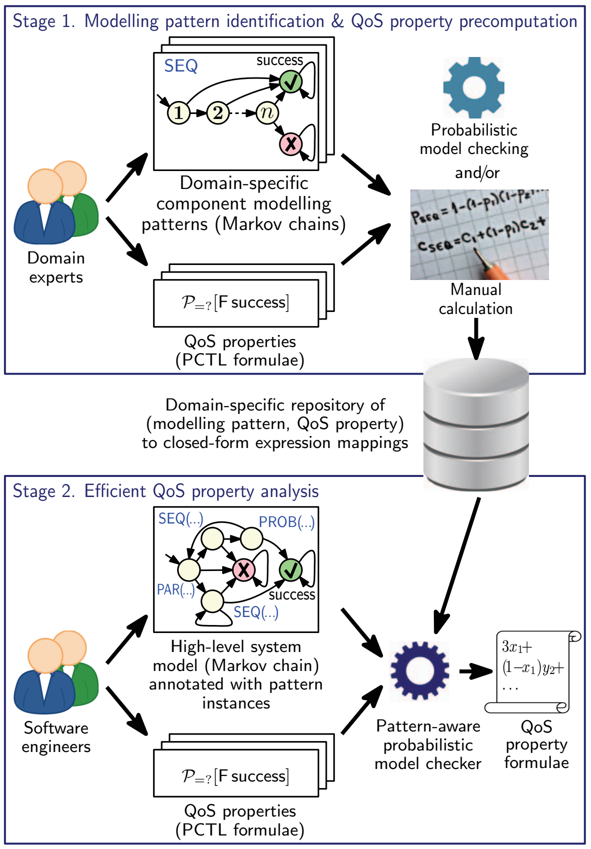
2 COMPOSITIONAL PMC

Our efficient parametric model checking method comprises the two stages depicted in Fig. 1. The first stage is domain specific and is performed only once for each domain (i.e. type of software system) that the method is applied to. This stage uses domain-expert input to identify *domain-specific modelling patterns* for components of systems from the considered domain, and precomputes closed-form expressions for key QoS properties of these patterns.

For example, the modelling patterns for the service-based systems domain (presented in Section 3) correspond to different ways in which $n \geq 1$ functionally-equivalent services can be used to execute an operation of the system. One option is to invoke the n services sequentially, such that service 1 is always invoked, and service i, $i > 1$, is only invoked if the invocations of services 1, 2, …, $i-1$ have all failed. The modelling pattern labelled 'SEQ' at the top of Fig. 1 shows this option. The graphical representation of the pattern shows the invocations of the n services as nodes labelled 1, 2, …, n, and the successful and failed completion of the operation as self-looping nodes labelled with a tick '✓' and a cross '✗', respectively. QoS properties such as the success probability and the cost of the operation for this pattern can be computed as $p_1 + (1-p_1)p_2 + \ldots + \left(\prod_{i=1}^{n-1}(1-p_i)\right)p_n$ and $p_1c_1 + (1-p_1)p_2c_2 + \ldots + \left(\prod_{i=1}^{n-1}(1-p_i)\right)p_nc_n$, respectively, where p_i, c_i are the probability of successful invocation and the cost of service i, $1 \leq i \leq n$.

As shown in Fig. 1, the computation described above can be performed using a probabilistic model checker such as PRISM, PARAM or Storm (for fixed values of n) or can be carried out manually. The resulting closed-form expressions are stored in a domain-specific repository for exploitation in the next stage of our PMC method.

The second stage of the method is performed for each structurally different variant of a system and QoS property under analysis. The stage involves the PMC of a parametric Markov chain that models the interactions between the system components. This high-level model can be provided by engineers with PMC expertise, or can be generated from more general software models, such as annotated UML activity diagrams [7]. The states of this model that correspond to system components are annotated with *pattern instances*, i.e. constructs which specify the modelling pattern used for each component and its parameters. For instance, the pattern instance $\mathsf{SEQ}(p_1, c_1, p_2, c_2)$ specifies that a component is implemented using $n = 2$ services invoked sequentially as described earlier, where the



Stage 1. Modelling pattern identification & QoS property precomputation
SEQ
success
Domain-specific component modelling patterns (Markov chains)
Probabilistic model checking
and/or
Manual calculation
Domain experts
$\mathcal{P}_{=?}$[F success]
QoS properties (PCTL formulae)
Domain-specific repository of (modelling pattern, QoS property) to closed-form expression mappings
Stage 2. Efficient QoS property analysis
SEQ(..)
PROB(..)
PAR(..)
success
SEQ(..)
High-level system model (Markov chain) annotated with pattern instances
Software engineers
$\mathcal{P}_{=?}$[F success]
QoS properties (PCTL formulae)
Pattern-aware probabilistic model checker
$3x_1+$
$(1-x_1)y_2+$
QoS property formulae

Figure 1: The two stages of the efficient PMC paradigm

success probabilities and costs of these two services are given by parameters named p_1, p_2 and c_1, c_2, respectively. The annotated high-level Markov model is analysed by a probabilistic model checker augmented with pattern manipulation capabilities. The result of the analysis is a system of equations (i.e. formulae) comprising:

(1) An equation (closed-form formula) for the system-level QoS property, given as a function of the component-level QoS property values. This equation is obtained by applying standard PMC to the high-level Markov model;
(2) Equations (closed-form formulae) for all the relevant component-level QoS property values. These equations are obtained by instantiating the appropriate formulae taken from the domain-specific repository of QoS property expressions.

These equations are all rational functions that can be very efficiently evaluated for any combinations of parameter values, e.g. using tools such as Matlab and Octave.

3 PMC OF SERVICE-BASED SYSTEMS

Service-based systems (SBSs) enable the effective development of new applications through the integration of third-party and in-house components implemented as services. SBSs are widely used, e.g. in e-commerce, online banking and e-government, and evolve frequently as a result of maintenance or self-adaptation [2, 15]. This evolution requires the QoS analysis of alternative SBS implementations, to select an implementation that meets the QoS requirements of the system. The alternative implementations differ in the way they use the multiple functionally-equivalent services available for each of their operations. Given $n \geq 1$ services that can perform the same SBS operation with probabilities of success $p_1, p_2, \ldots, p_n$, costs $c_1, c_2, \ldots, c_n$, and execution times $t_1, t_2, \ldots, t_n$, the operation can be implemented using one of the patterns described in Table 1. Variants of the first three patterns have been widely used in related research (e.g. in [16]), while—to the best of our knowledge—the remaining patterns from Table 1 have not been considered before.

Our efficient PMC method is well suited for the SBS domain, as the operation implementation patterns from Table 1 correspond to component modelling patterns whose reliability, cost and execution time can be obtained in the first stage of the method. Table 2 shows the repository of (manually derived) expressions for all patterns from Table 1 and three key QoS properties of SBS operations.

4 PRELIMINARY EXPERIMENTS

We carried out experiments to compare the feasibility, scalability and efficiency of our PMC method with those of the model checkers PRISM, Storm and PARAM. All experiments were performed on a Ubuntu-16 server with i7-4770@3.40GHz × 4 processors and 16GB of memory, using the latest versions of the three model checkers and a proof-of-concept tool that implements Stage 2 of our PMC method. The tool outputs the PMC result as a Matlab script that combines the closed-form expressions obtained through the Storm verification of the high-level parametric Markov model of the analysed system with the appropriate expressions from the repository in Table 2. To enable the reproducibility of our results, we made the proofs, models, verified properties and results from our experiments available on the project webpage https://www.cs.york.ac.uk/tasp/ePMC/.

Our experiments used a foreign-exchange (FX) trading system adopted from [4, 8] and shown in Fig. 2. In the "normal" mode a *Fundamental Analysis* component decides whether a transaction should be performed, the analysis should be retried, or the session should be ended. Performing a transactions uses an *Order* component followed by the invocation of a *Notification* component to inform the trader about the outcome of the transaction. In "expert" mode, a *Market Watch* component obtains exchange market information to be processed by a *Technical Analysis* component. Analysis results may satisfy a set of trader-specified objectives (in which case a transaction is performed), may not meet these objectives (so the *Market Watch* is reinvoked for an update) or may be erroneous (in which case an *Alarm* component is used to warn the trader). We assumed that the probabilities annotating the decision points from Fig. 2 were obtained from system logs.

To evaluate our PMC method, we considered multiple ways to implement the six FX components using modelling patterns from Table 1 with between one and four functionally-equivalent services per component. To analyse this large set of alternative designs with our PMC method, we devised an annotated high-level parametric Markov model of FX as described in Section 2. To analyse the same designs with the model checkers PRISM, Storm and PARAM, we obtained a "monolithic" model for each design by using a Markov chain generator that we implemented for this purpose. So as to not bias the comparison, and to the best of our ability, we optimized this generator to generate models that are as small as possible. Three QoS properties of the SBS were analyzed: (P1) the probability

Table 1: Implementation patterns for SBS operations

Pattern	Description
$\mathsf{SEQ}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n)$	n services are invoked in order, stopping after a successful invocation or after the last service.
$\mathsf{PAR}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n)^\dagger$	n services are all invoked at the same time, and the operation uses the first result returned by a successful invocation (if any).
$\mathsf{PROB}(x_1, p_1, c_1, t_1, \ldots, x_n, p_n, c_n, t_n)$	A single service is invoked; x_i gives the probability that this is service i, where $\sum_{i=1}^{n} x_i = 1$.
$\mathsf{SEQ\_R1}(p_1, c_1, t_1, r_1, \ldots, p_n, c_n, t_n, r_n)$	The services are invoked in order. If service i fails, it is reinvoked with probability r_i; with probability $1 - r_i$, the operation is attempted using service $i + 1$ (if $i < n$) or fails (if $i = n$).
$\mathsf{SEQ\_R2}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n, r)$	n services are invoked in order as for the SEQ pattern; if all n invocations fail, the execution of the operation is retried (from service 1) with probability r otherwise the operation fails.
$\mathsf{PAR\_R}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n, r)^\dagger$	All n services are invoked as for the PAR pattern; if all n invocations fail, the execution of the operation is retried with probability r or the operation fails with probability $1 - r$.
$\mathsf{PROB\_R1}(x_1, p_1, c_1, t_1, r_1, \ldots, x_n, p_n, c_n, t_n, r_n)$	Like PROB, a single service i is invoked; however, its invocation is retried after failure(s) with probability r_i or the operation fails with probability $1 - r_i$.
$\mathsf{PROB\_R2}(x_1, p_1, c_1, t_1, \ldots, x_n, p_n, c_n, t_n, r)$	Like PROB, a single service i is invoked; if the invocation fails, the PROB pattern is retried with probability r or the operation fails with probability $1 - r$.

[†]Pattern unsuitable for non-idempotent operations (e.g. credit card payment in an e-commerce SBS)

Table 2: Repository of (modelling pattern, QoS property) to closed-form expression mappings for the SBS domain

Pattern	**Success prob.**, $\mathsf{P}_{=?}[\mathsf{F}\ \mathsf{success}]$	**Cost**, $\mathsf{R}^{\mathsf{cost}}_{=?}[\mathsf{F}\ \mathsf{success} \vee \mathsf{fail}]$	**Execution time**, $\mathsf{R}^{\mathsf{time}}_{=?}[\mathsf{F}\ \mathsf{success} \vee \mathsf{fail}]$
SEQ	$p_{\mathsf{SEQ}} = 1 - \prod_{i=1}^{n}(1 - p_i)$	$c_{\mathsf{SEQ}} = c_1 + \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1}(1 - p_j)\right) c_i$	$t_{\mathsf{SEQ}} = t_1 + \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1}(1 - p_j)\right) t_i$
PAR[†]	$p_{\mathsf{PAR}} = p_{\mathsf{SEQ}}$	$c_{\mathsf{PAR}} = \sum_{i=1}^{n} c_i$	$t_{\mathsf{PAR}} = \widetilde{p}_1 t_1 + \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1}(1 - p_j)\right) \widetilde{p}_i t_i$
PROB	$p_{\mathsf{PROB}} = \sum_{i=1}^{n} x_i p_i$	$c_{\mathsf{PROB}} = \sum_{i=1}^{n} x_i c_i$	$t_{\mathsf{PROB}} = \sum_{i=1}^{n} x_i t_i$
SEQ_R1[‡]	$p_{\mathsf{SEQ\_R1}} = 1 - \prod_{i=1}^{n}(1 - p'_i)$	$c_{\mathsf{SEQ\_R1}} = c'_1 + \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1}(1 - p'_j)\right) c'_i$	$t_{\mathsf{SEQ\_R1}} = t'_1 + \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1}(1 - p'_j)\right) t'_i$
SEQ_R2	$p_{\mathsf{SEQ\_R2}} = \frac{p_{\mathsf{SEQ}}}{1-(1-p_{\mathsf{SEQ}})r}$	$c_{\mathsf{SEQ\_R2}} = \frac{c_{\mathsf{SEQ}}}{1-(1-p_{\mathsf{SEQ}})r}$	$t_{\mathsf{SEQ\_R2}} = \frac{t_{\mathsf{SEQ}}}{1-(1-p_{\mathsf{SEQ}})r}$
PAR_R	$p_{\mathsf{PAR\_R}} = p_{\mathsf{SEQ\_R2}}$	$c_{\mathsf{PAR\_R}} = \frac{c_{\mathsf{PAR}}}{1-(1-p_{\mathsf{PAR}})r}$	$t_{\mathsf{PAR\_R}} = \frac{t_{\mathsf{PAR}}}{1-(1-p_{\mathsf{PAR}})r}$
PROB_R1[‡]	$p_{\mathsf{PROB\_R1}} = \sum_{i=1}^{n} x_i p'_i$	$c_{\mathsf{PROB\_R1}} = \sum_{i=1}^{n} x_i c'_i$	$t_{\mathsf{PROB\_R1}} = \sum_{i=1}^{n} x_i t'_i$
PROB_R2	$p_{\mathsf{PROB\_R2}} = \frac{p_{\mathsf{PROB}}}{1-(1-p_{\mathsf{PROB}})r}$	$c_{\mathsf{PROB\_R2}} = \frac{c_{\mathsf{PROB}}}{1-(1-p_{\mathsf{PROB}})r}$	$t_{\mathsf{PROB\_R2}} = \frac{t_{\mathsf{PROB}}}{1-(1-p_{\mathsf{PROB}})r}$

[†]assuming that the n services are ordered such that $t_1 \leq t_2 \leq \cdots \leq t_n$, with $\widetilde{p}_i = p_i$ for $i < n$ and $\widetilde{p}_n = 1$

[‡] $p'_i = \frac{p_i}{1-(1-p_i)r_i}$, $c'_i = \frac{c_i}{1-(1-p_i)r_i}$ and $t'_i = \frac{t_i}{1-(1-p_i)r_i}$ for all $i = 1, 2, \ldots, n$

of successful completion; (P2) the expected execution time; and (P3) the expected cost.

Table 3 shows that the PMC time required to analyse these properties using our method is always better and typically orders of magnitude smaller than the PMC times of PRISM, Storm and PARAM (except when a single service is used for each FX component, cf. row 1). Moreover, the three tools ran out of memory or timed out when components used four (and sometimes even two or three) services. With the exception of the last row, the results from Table 3 summarise experiments in which every FX component used the same pattern (SEQ, PAR, etc.) and number of services. The last row reports the mean time, standard deviation ('sd') and number of timeouts over 10 experiments in which the pattern and number of services (two or three) used for each component were chosen randomly. Note that for our method we report the combined PMC time for all properties because of the insignificant variation between individual times for each property (and to save space). Our proof-of-concept tool invokes Storm to analyse the high-level SBS model for each property, which takes the times highlighted in the first row of Table 3 (as the high-level model is identical to the monolithic Markov chain for the SEQ pattern with one service), and then only requires a few milliseconds to extract and instantiate the appropriate precomputed formulae from the repository in Table 2.

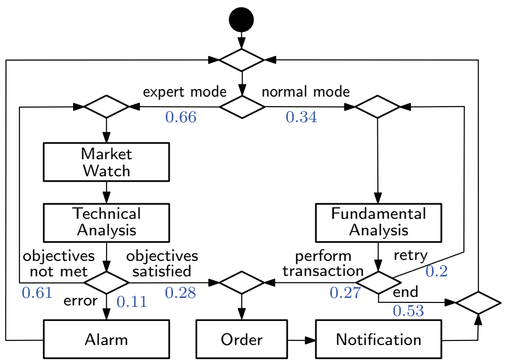


Figure 2: Activity diagram of the FX service-based system

To further evaluate our PMC method, we plotted properties from the systems of equations generated by our tool and the algebraic expressions generated by Storm (the best performing model checker in our experiments). Fig. 3 shows graphs for P1 and P3 generated by Matlab. For an FX instance using the PROB_R2 pattern with

Table 3: Partial summary of PMC verification time, in seconds (see the complete table at https://www.cs.york.ac.uk/tasp/ePMC/)

		Our PMC method	PRISM			Storm			PARAM		
Pattern	# services	P1 + P2 + P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
SEQ	1	0.089	0.114	0.356	0.346	0.025	0.028	0.028	0.008	0.030	0.029
SEQ	2	0.089	1.498	M	M	0.081	0.616	0.611	0.325	20.25	21.58
SEQ	3	0.090	M	-	-	7.102	763.5	783.7	463.3	T	T
SEQ	4	0.090	-	-	-	T	T	T	T	-	-
PAR	2	0.090	0.644	M	M	0.085	0.817	0.404	0.323	19.53	10.29
PAR	3	0.090	M	-	-	118	703.3	107.6	511.3	T	T
PAR	4	0.090	-	-	-	T	T	T	T	-	-
PROB	2	0.090	M	M	M	0.066	0.201	0.200	T	T	T
PROB	3	0.090	-	-	-	0.184	0.950	0.964	-	-	-
PROB	4	0.090	-	-	-	T	T	T	-	-	-
SEQ_R1	2	0.089	M	M	M	T	T	T	T	T	T
SEQ_R1	3	0.092	-	-	-	-	-	-	-	-	-
PROB_R2	2	0.090	M	M	M	12.90	T	T	T	T	T
PROB_R2	3	0.093	-	-	-	567.8	-	-	-	-	-
PROB_R2	4	0.090	-	-	-	T	-	-	-	-	-
10 random	2 or 3	0.090, sd=0.002	M	M	M	4.7, sd=5.9	11.3, sd=10.8	6.8, sd=5.5	T	T	T
		no time out	all out of memory			2 timed out	3 timed out	4 timed out	all timed out		

Key: M = out of memory; T = timeout limit (15 minutes) exceeded; - = skipped experiment, as PMC failed for smaller Markov chain

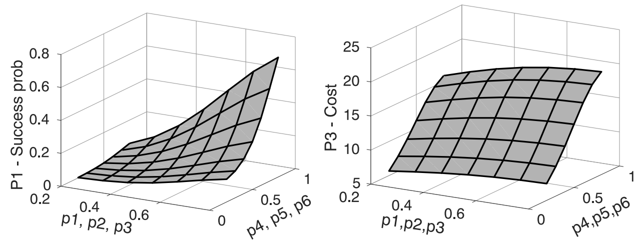


Figure 3: QoS analysis showing increase in FX success probability and (as fewer FX sessions fail halfway) cost when the reliability p_i of (all) services used for the i-th FX component increases (all services are assumed to have a cost of 1)

three services per component our tool generated a 1,396-character system of equations for P1 and was plotted in 2.6 seconds. The 1,130,276-character formula generated by Storm took 8.2 hours to plot. The P3 graph, for an FX instance using the PAR pattern with three services per component, was plotted in 1.3 seconds using our 767-character system of equations, and in 2.5 hours using the 705,648-character formula from Storm.

5 RELATED WORK

Our PMC method is orthogonal to the other research on improving the efficiency of PMC, which does not exploit domain knowledge. For example, the state-of-the-art PMC method introduced in [13] employs a new polynomial factorisation technique and a *domain-agnostic* decomposition of the analysed Markov chain into strongly connected components to accelerate PMC. Our PMC method builds on top of this advance by using Storm [6] (which implements the method from [13]) in both its stages, and significantly accelerates PMC further by leveraging domain knowledge.

6 CONCLUSION

We introduced a method that exploits domain-specific component modelling patterns to considerably speed-up the PMC of component-based systems. As we showed for the service-based systems domain, the new PMC method also generates (systems of) algebraic expressions that are much smaller and can be evaluated much faster than those produced by existing PMC tools. Our ongoing work suggests that similar gains can be achieved for other domains where the QoS properties of component-based systems are analysed using Markov models, including software product lines [9] and multi-tier software architectures [3].

REFERENCES

[1] S Andova, H Hermanns, and J-P Katoen. 2004. Discrete-Time Rewards Model-Checked. In *FORMATS'03*. 88–104.
[2] R Calinescu, K Johnson, and Y Rafiq. 2013. Developing self-verifying service-based systems. In *ASE'13*. 734–737.
[3] R Calinescu, S Kikuchi, and K Johnson. 2012. Compositional Reverification of Probabilistic Safety Properties for Large-Scale Complex IT Systems. In *Monterey'12*. Springer, 303–329.
[4] R Calinescu, D Weyns, S Gerasimou, M U Iftikhar, I Habli, and T Kelly. 2017. Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases. *IEEE Trans. Software Eng.* PP, 99 (2017).
[5] C Daws. 2005. Symbolic and Parametric Model Checking of Discrete-time Markov Chains. In *ICTAC'04*. 280–294.
[6] C Dehnert et al. 2017. A Storm is Coming: A Modern Probabilistic Model Checker. In *CAV'17*. 592–600.
[7] S Gallotti et al. 2008. Quality Prediction of Service Compositions Through Probabilistic Model Checking. In *QoSA'08*. 119–134.
[8] S Gerasimou, G Tamburrelli, and R Calinescu. 2015. Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering. In *ASE'15*. 319–330.
[9] C Ghezzi and A M Sharifloo. 2013. Model-based verification of quantitative non-functional properties for software product lines. *Information & Software Technology* 55, 3 (2013), 508–524.
[10] E M Hahn et al. 2010. PARAM: A Model Checker for Parametric Markov Models. In *CAV'10*. 660–664.
[11] E M Hahn, H Hermanns, and L Zhang. 2011. Probabilistic reachability for parametric Markov models. *International Journal on Software Tools for Technology Transfer* 13, 1 (2011), 3–19.
[12] H Hansson and B Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5 (1994), 512–535.
[13] N Jansen et al. 2014. Accelerating Parametric Probabilistic Verification. In *QEST'14*. 404–420.
[14] M Kwiatkowska, G Norman, and D Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *CAV'11*. 585–591.
[15] D Weyns and R Calinescu. 2015. Tele Assistance: A Self-adaptive Service-based System Examplar. In *SEAMS'15*. 88–92.
[16] L Zeng et al. 2004. QoS-aware middleware for web services composition. *IEEE Trans. Software Eng.* 30, 5 (2004), 311–327.