Softw Syst Model (2014) 13:1395–1415
DOI 10.1007/s10270-012-0277-5

THEME SECTION PAPER

Evaluating probabilistic models with uncertain model parameters

Indika Meedeniya · Irene Moser · Aldeida Aleti · Lars Grunske

Received: 30 January 2012 / Revised: 1 July 2012 / Accepted: 23 July 2012 / Published online: 1 September 2012

Abstract Probabilistic models are commonly used to evaluate quality attributes, such as reliability, availability, safety and performance of software-intensive systems. The accuracy of the evaluation results depends on a number of system properties which have to be estimated, such as environmental factors or system usage. Researchers have tackled this problem by including uncertainties in the probabilistic models and solving them analytically or with simulations. The input parameters are commonly assumed to be normally distributed. Accordingly, reporting the mean and variances of the resulting attributes is usually considered sufficient. However, many of the uncertain factors do not follow normal distributions, and analytical methods to derive objective uncertainties become impractical with increasing complexity of the probabilistic models. In this work, we introduce a simulation-based approach which uses Discrete Time Markov Chains and probabilistic model checking to accommodate a diverse set of parameter range distributions. The number of simulation runs automatically regulates to the desired significance level and reports the desired percentiles of the values which ultimately characterises a specific quality attribute of the system. We include a case study which illustrates the flexibility of this approach using the evaluation of several probabilistic properties.

Keywords Software architecture evaluation · Parameter uncertainty · Probabilistic quality models · Monte-Carlo simulation

Communicated by Prof. Dr. Dorina Petriu and Dr. Jens Happe.

I. Meedeniya (✉) · I. Moser
Faculty of Information and Communication Technologies,
Swinburne University of Technology,
Hawthorn,VIC 3122, Australia
e-mail: imeedeniya@swin.edu.au

I. Moser
e-mail: imoser@swin.edu.au

A. Aleti
Faculty of Information Technology, Monash University,
Clayton, VIC 3800, Australia
e-mail: aldeida.aleti@monash.edu

L. Grunske
Faculty of Computer Science and Center for Mathematical
and Computational Modelling (CM), University of Kaiserslautern,
67653 Kaiserslautern, Germany
e-mail: grunske@cs.uni-kl.de

1 Introduction

Probabilistic quality evaluation models are an important asset during the development of software-intensive embedded systems. The benefit of these evaluation models is especially evident in the architectural design phase, since different design alternatives can be evaluated and software architects are able to make informed choices between these alternatives. To date, a number of evaluation models have been proposed for evaluating specific quality attributes such as performance [7,9], reliability [23] and safety [30]. However, specific parameters of these quality evaluation models are often just estimated. These estimations tend to use field data obtained during testing or operational usage, historical data from products with similar functionality, or reasonable guesses made by domain experts. In practice, however, parameters can rarely be estimated accurately [4,20,28]. In the context of software-intensive systems design and probabilistic quality evaluation models, the sources of uncertainty can be classified into two major categories.

Aleatory uncertainty *is the inherent variation associated with the physical system or environment under consideration*

[43]. This category refers to sources of uncertainty that are inherently stochastic in nature. Physical uncertainties such as noise in electrical conductors, humidity, temperature, material parameters, behavior and instantaneous decisions of operators are examples in the domain of embedded systems. This type of uncertainty cannot be avoided [10].

Epistemic uncertainty *is uncertainty of the outcome due to the lack of knowledge or information in any phase or activity of the modelling process* [43]. This source of uncertainty reflects the lack of knowledge about the exact behaviour of the system. Uncertainties of this type are subjective and depend on factors such as maturity of the design and models, experience of the application domain and the coverage and extent of testing.

Both of these types of uncertainty manifest themselves in model parameters related to software components, inter-component interactions, hardware components, communication links, operational profile and use cases. Thus a solution to handle uncertain parameters in probabilistic quality evaluations is relevant for the decision making in the development phases of software-intensive systems.

Related work In the context of quality evaluation under uncertainty, a considerable amount of research has been conducted in the area of sensitivity analysis with respect to the parameters of probabilistic quality evaluation models. Most of the approaches to date have concentrated on specific quality attributes. In the area of architecture-based reliability evaluation Cheung [14] presented a sensitivity analysis method for his original reliability model with a composite Discrete-Time Markov Chain (DTMC) abstraction. The method is purely analytical and consists of a number of 2nd and 3rd order partial derivatives of system reliabilities which are hard to estimate in real cases. Goševa-Popstojanova and Kamavaram [28] proposed the *method of moments* to calculate the sensitivity of a system's reliability to component reliabilities and transition probabilities analytically. Cortellessa and Grassi [18] discussed the significance of error propagation to other parts of the system. Their sensitivity analysis can help identify the most critical system components. Coit et al. [16,51] have used means and variances of reliability estimates of software components to derive the mean and variance of the reliability of a redundancy allocation. Assuming normal distributions for input, Finodella and Gokhale [20] derived the distribution of system reliability from a multinomial distribution. Coit and Smith [17] presented an analytical approach to obtain the lowerbound percentile of the reliability in series–parallel systems. Bhunia et al. [11] applied a similar method to the evaluation of reliability bounds. All of the above methods are analytical approaches to quantifying sensitivity. One disadvantage of these analytical sensitivity analysis methods is poor generalisability. All methods discussed are based on the assumption of normal parameter distributions which are characterised by the mean and variance alone.

Goševa-Popstojanova et al. [26,27] have shown that analytical methods of uncertainty analysis do not scale well and proposed a method based on Monte Carlo (MC) simulation. Using experimental validation they demonstrated that the MC method scale better than the *method of moments* approach [24]. Similarly, Marseguerra et al. [38] have used mean and variance estimates of component reliabilities to obtain the mean and variance of the system reliability using MC simulation. These approaches have applied simulations to reliability-specific models, assuming normal input distributions. Yin et al. [52] have proposed a DTMC simulation-based approach to derive system reliability from the probability distributions of component reliabilities under the assumption that component and system reliabilities are gamma-distributed. Axelsson [4] has also highlighted the significance of MC-based approaches in cost evaluation with uncertainty.

The existing MC-simulation-based uncertainty analysis approaches are based on the assumption that there is a specific continuous input distribution and that the resulting sample distribution is normal or Weibull. In contrast, practical experience in connection with reliability studies [21,37] show that the actual distributions are hard to determine. Mean-and-variance-based quality evaluations are not sufficient for informed decision making in the case of safety-and-mission-critical systems.

Contribution and overview In previous work [41], we investigated different aspects of parameter uncertainties in architecture-based quality evaluation with a specific focus on reliability and formulated a framework that incorporates specification, evaluation and quantification of probabilistic quality attributes in the presence of uncertainty. This framework uses Monte Carlo simulations to evaluate the reliability of a system based on its architecture specification. As a result, the approach is able to incorporate such heterogeneous uncertainties as they naturally occur in software architecture evaluation models [4,45,46,52]. Figure 1 illustrates the elements of the approach proposed in Meedeniya et al. [41] and their relationships. The leftmost element represents the specification of the uncertainty of parameters in terms of a diverse set of probability distributions. These parameters and the system architecture are used to generate a reliability evaluation model which is analysed to determine the quality of the prospective system. Based on the results of multiple Monte Carlo (MC) simulations, estimates for the reliability attributes of the architectures are computed. A dynamic stopping criterion terminates the MC simulations when sufficient samples have been taken to achieve the desired accuracy.

In this paper, we extend the previous work and show that the approach applies not only to reliability evaluations but

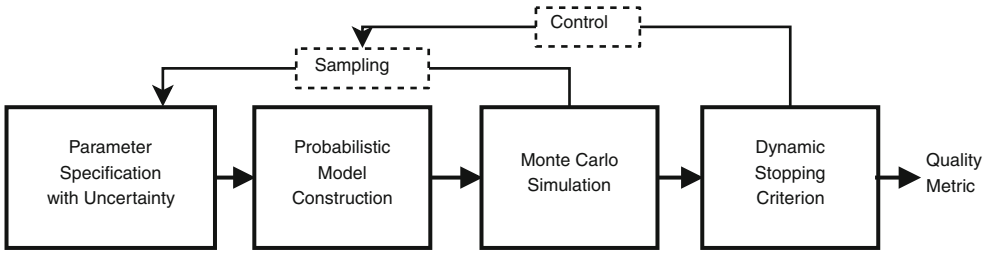

Fig. 1 Architecture evaluation under uncertainty

also to a variety of probabilistic properties. Instead of using a dedicated reliability evaluation model with a specific evaluation technique we utilise probabilistic model checking. As a result, we are able to handle any probabilistic quality requirement that can be formulated in the probabilistic temporal logical formula [5,6,31]. To implement the approach we use the PRISM model-checker [34] as a back-end to evaluate a specific quality evaluation model for a Monte Carlo (MC) simulation run. As a result, our approach can be integrated into any of the probabilistic quality evaluations and evaluation models that are supported by PRISM.

The remainder of this article is organised as follows: In Sect. 2 we shortly describe the main formalisms used throughout the paper, namely probabilistic temporal logics and Markov Models. Section 3 motivates the research and provides the details of an Anti-lock Brake System (ABS), which is used in this paper as a example to explain the concepts. The four elements of the approach depicted in Fig. 1 are defined in detail in Sects. 4–7, while a validation of the proposed approach is presented in Sect. 8. Section 9 concludes the paper and points out directions for future works.

2 Preliminaries

2.1 Formal definition of quality requirements with probabilistic temporal logics

The precise specification of quality requirements is an important aspect for architecture evaluation procedure [22]. Probabilistic temporal logic formulations are a suitable specification formalism [29]. These probabilistic logic models extend real-time temporal logic such as MTL (Metric Temporal Logic) [33] and TCTL (Timed Computational Tree Logic) [1] with operators to reason about the probabilistic behavior of a system. Commonly used probabilistic temporal logic formulations are PCTL (Probabilistic Computation Tree Logic) [31], PCTL* [5], PTCTL (Probabilistic Timed CTL) [36] and CSL (Continuous Stochastic Logic) [6]. The significant benefits of using logic-based requirement specifications include the ability to define these requirements concisely and unambiguously and to analyse architectures using rigorous, mathematics-based tools such as model checkers. In this article we focus on PCTL which is defined as follows [31]:

Definition 1 (*PCTLSyntax*) Let AP be a set of atomic propositions and $a \in AP$, $p \in [0,1]$, $t_{PCTL} \in \mathbb{N}$, and $\bowtie \in \{\geq, >, <, \leq\}$, then a state-formula Φ and a path formula Ψ in PCTL are defined by the following grammar:

$$\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}(\Psi)$$
$$\Psi ::= X\Phi \mid \Phi U \Phi \mid \Phi U^{\leq t_{PCTL}} \Phi$$

Probabilistic temporal logic distinguishes between state and path formulae. The state formulae include the standard logical operators $\wedge$ and $\neg$, which also allow a formulation of other usual logical operators (disjunction ($\vee$), implication ($\Rightarrow$), etc.) and $false$. The main extension of the state formulae, compared with non-probabilistic logics, is to replace the traditional path quantifier $\exists$ and $\forall$ with a probabilistic operator $\mathcal{P}$. This probabilistic operator defines upper or lower bounds on the probability of the system evolution. As an example, the formula $\mathcal{P}_{\geq p}(\Psi)$ is true at a given state if the probability of the future evolution of the system satisfying Ψ is at least p. Similarly, the formula $\mathcal{P}_{\leq p}(\Psi)$ is true if the probability that the system fulfils (Ψ) is less than or equal to p. In addition to the $\mathcal{P}_{\bowtie p}(\Psi)$ probability bound specification in PCTL syntax, the numerical probability value that the system fulfils (Ψ) is denoted by $\mathcal{P}_{=?}(\Psi)$ [35]. The path formulae that can be used with the probabilistic path operator are the "next" formula $X\Phi$, time bounded "until" formula $\Phi_1 U^{\leq t} \Phi_2$ and unbounded "until" formula $\Phi_1 U \Phi_2$. The formula $X\Phi$ holds if Φ is true in the next state of a path. Intuitively, the time bounded "until" formula $\Phi_1 U^{\leq t} \Phi_2$ requires that Φ_1 holds continuously within a time interval $[0, x)$ where $x \in [0, t]$, and Φ_2 becomes true at time instance x. The semantics of the unbounded versions is identical, but the (upper) time bound is set to infinity $t = \infty$. Based on the time-bounded and -unbounded "until" formula further temporal operators ("eventually" $\Diamond$, "always" $\Box$, and "weak until" $\mathcal{W}$) can be expressed as described in Ciesinski and Größer [15] as well as Grunske [29]. For example, the eventually formula $\mathcal{P}_{\bowtie p}(\Diamond\Phi)$ is semantically equivalent to $\mathcal{P}_{\bowtie p}(true U \Phi)$.

Traditionally, the semantics of the PCTL is defined with a satisfaction relation $|=$ over the states S and possible paths $Path^{\mathcal{M}}(s)$ that are possible in a state $s \in S$ of a discrete time probabilistic model $\mathcal{M}$. For details about the formal semantics the reader is referred to Ciesinski and Größer [15] or Hansson and Jonsson [31].

2.2 Quality evaluation models

Several approaches exist in the literature for the model-based quality analysis and prediction, spanning the use of Petri nets, queueing networks, layered queueing networks, stochastic process algebras, Markov processes, fault trees, statistical models and simulation models (see Ardagna et al. [2] for a recent review and classification of models for software quality analysis).

In this article, we focus on Markov models which are a very general evaluation model that can be used to reason about a wide variety of properties including performance and reliability.

Specifically, Markov models are stochastic processes defined as state-transition systems augmented with probabilities. Formally, a stochastic process is a collection of random variables $X(t), t \in T$ all defined on a common sample (probability) space. The $X(t)$ is the state while (time) t is the index that is a member of set T (which can be discrete or continuous). In Markov models [13], states represent possible configurations of the system being modelled. Transitions among states occur at discrete or continuous time-steps and the probability of making transitions is given by exponential probability distributions. The Markov property, which defines a subsequent state as dependent of the immediately preceding state only, characterises these models. In other words, the description of the present state fully captures all the information that could influence the future evolution of the process. The most commonly used Markov models include

- *Discrete Time Markov Chains* (DTMC), which are the simplest Markovian model where transitions between states happen at discrete time steps;
- *Continuous Time Markov Chains* (CTMC) where the value associated with each outgoing transition from a state is intended not as a probability but as a parameter of an exponential probability distribution (transition rate);
- *Markov Decision Processes* (MDP) [44] that are an extension of DTMCs allowing multiple probabilistic behaviours to be specified as output of a state. These behaviours are selected non-deterministically.

Definition 2 A *Discrete Time Markov Chain (DTMC)* is a tuple (S, P) where S is a finite set of states, and $P : S \times S \rightarrow [0, 1]$ is a transition probability matrix.

A DTMC is *absorbing* when at least one of its states has no outgoing transition [49]. Such states are known as absorbing states.

Definition 3 A labelled discrete time *Markov Reward Model (MRM)* is a triple $\mathcal{M} = ((S, P), \rho, \tau)$ where (S, P) is an underlying DTMC, $\rho : S \rightarrow \mathbb{R}_{\geq 0}$ is a state reward structure and $\tau : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is an impulse reward structure satisfying $\forall s \in S : \tau(s, s) = 0$.

A path of an absorbing DTMC is a finite sequence $\sigma = s_0 s_1 s_2 \ldots s_n$ of states, where s_n is an absorbing state. Let $X_\sigma(s)$ denote the number of visits of state s in path σ. Similarly, let $XT_\sigma(s, s')$ represent the number of occurrences of transition (s, s') in σ. Then we can calculate the *accumulated reward of* σ as

$$\mathcal{R}_\sigma = \sum_{s \in S} (X_\sigma(s) \cdot \rho(s)) + \sum_{(s,s') \in (S \times S)} (XT_\sigma(s, s') \cdot \tau(s, s')) \quad (1)$$

The accumulated reward properties can also be specified in conjunction with CSL, when a set of paths is given in CSL formula. For instance, $\mathcal{R}_{=?}(\Diamond\, \Phi)$ denotes the expected reward cumulated before a state satisfying Φ is reached.

The analytical solution techniques for Markov models differ according to the specific model and the underlying assumptions (e.g., transient or non-transient states, continuous vs. discrete time, etc.). For example, the evaluation of the stationary probability π_s of a DTMC model requires the solution of a linear system whose size is given by the cardinality of the state space S. The exact solution of such a system can be obtained only if S is finite or when the matrix of transition probabilities has a specific form.

3 Example application

The example of a deployment architecture which assigns software components to a hardware infrastructure of electronic control units (ECUs) is used to illustrate the concepts introduced in this paper. The software components belong to the *Anti-lock Brake System* (ABS) of a car. ABS is a system that maintains traction during braking manoeuvres to prevent skidding. It is therefore a crucial safety feature found in most contemporary cars. The system and its parameters are briefly described in this section, and further details can be found in Meedeniya et al. [40].

3.1 Software components and interactions

The software components in this example are treated as black boxes [32], i.e. a description of the externally visible parameters while the internal structures are unknown and not modifiable. The components interact to implement a set of services,

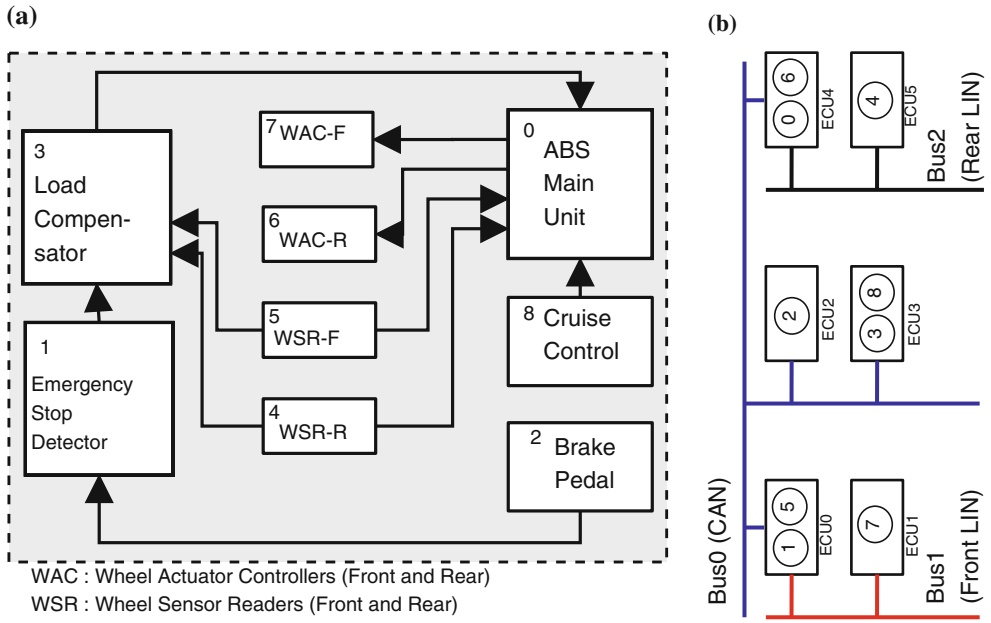

Fig. 2 Software components and their interactions. The deployment shows the allocations of software components, represented by numbers, to the HW topology of the ABS system. **a** SW architecture. **b** Deployment

defining the functional units accessed by the user of the system. The ABS activity is initiated in one software component (with a given probability) which may employ many auxiliary components it is connected to via communication links. The process and logical views of the subsystems are depicted in Fig. 2a. The *ABS Main Unit* is the major decision-making unit regarding the braking levels for individual wheels, while the *Load Compensator* unit assists with computing adjustment factors from the wheel load sensor inputs. Components 4 to 7 represent transceiver software components associated with each wheel and communicate with sensors and brake actuators. *Brake Pedal* is the software component that reads from the paddle sensor and sends the data to the *Emergency Stop Detection* software module.

The software components are characterised by two externally observable parameters:

(a) *Workload* (wl): computational load of a software component in executing a requested task; expressed in MI (million instructions).
(b) *Execution initiation probability* (q_0): the probability of the program execution starting at this component.

The interaction between two components $\mathcal{C}_i$ and $\mathcal{C}_j$ have the following observable characteristics:

(a) *Data size* (ds): the amount of data transmitted from software component $\mathcal{C}_i$ to $\mathcal{C}_j$ during a single communication event; expressed in KB (kilobytes).
(b) *Next-step probability* (p): the probability that a service calls component $\mathcal{C}_j$ after component $\mathcal{C}_i$.

3.2 Hardware topology and deployment

The hardware model used to deploy the software components is comprised of a distributed set of certified ECUs having different capacities of memory, processing power, access to sensors, etc. ECUs communicate through buses. Many types of buses with different data rates and reliability can be present. The ECUs and the bus system that compose the hardware architecture for the system are depicted in Fig. 2b. In this example, we consider one of the feasible deployments of software components to the hardware architecture. The numbers in Fig. 2b refer to the allocated software components with corresponding ids in Fig. 2a.

The ECUs in this system have the following relevant parameters:

(a) *Processing speed* (ps): the instruction-processing capacity of the ECU; expressed in MIPS (million instructions per second). This is used to calculate the execution time, which is a function of processing speed of the ECU and the computation workload of the service.
(b) *Failure rate* (fr): failure rate (of the exponential distribution [3,12]) that characterises the probability of a single ECU failure.
(c) *Energy dissipation rate* (er): the rate of energy dissipation during execution within an ECU; expressed in mW (millijoules per second).

Table 1 Probability distributions and their specification

Distribution	Specification syntax	Range	Example
Normal	NORMAL, μ, σ^2	$(-\infty, \infty)$	$NORMAL, 3.75, 0.05$
Beta	BETA, $\alpha,\ \beta$	[0, 1]	$BETA, 10, 2$
Shifted beta	BETA_SHD, $\underline{x},\ \overline{x},\ \alpha,\ \beta$	$(\underline{x}, \overline{x})$	$BETA\_SHD, 3, 5, 2, 10$
Exponential	EXP, λ	$(0, \infty)$	$EXP, 7.5 \times 10^{-6}$
Uniform	UNIFORM, $\underline{x},\ \overline{x}$	$(\underline{x}, \overline{x})$	$UNIFORM, 3.5, 4.0$
Gamma	GAMMA, λ	$(0, \infty)$	$GAMMA, 1.5$
Weibull	WEIBULL, α	$(0, \infty)$	$WEIBULL, 1.5$
Discrete	DISCRETE, $x_0,\ p_0, x_1,\ p_1, \ldots, x_n,\ p_n$	(x_0, x_n)	$DISCRETE, 2, 0.4, 2.1, 0.5, 2.3, 0.1$

The buses that connect the ECUs can vary in terms of the following observable properties:

(a) *Data rate* (dr): the data transmission rate of the bus, expressed in KBPS (kilobytes per second). This is used to calculate the latency in for data transmission, as it is a function of the data rate of the bus and the amount of data transmitted during the communication.
(b) *Failure rate* (fr): failure rate of the exponential distribution characterizing data communication failure of each bus.
(c) *Energy dissipation rate* (er): the rate of energy dissipation during the use of communication channel, expressed in mW (millijoules per second).

3.3 Objectives

The case study is concerned with the architecture-based evaluation of reliability, response time and energy consumption. As a guiding example throughout the presentation of the approach, we evaluate the reliability of the ABS function in a given deployment architecture. With respect to the deployment, two sources of failure, defined in Meedeniya et al. [40], are considered for the reliability evaluation.
Execution failures Failures may occur in the ECUs during the execution of a software component, which affects the reliability of the software modules running on that ECU. For this illustration, we assume a fixed deterministic scheduling of tasks within the ECU. It is also assumed that the failure that happens in an ECU while a software component is executing or queued leads to a service execution failure.
Communication failures Failure of a data communication bus when a software component communicates with another over the bus directly leads to a failure of the service that depends on this communication.

The annotations, model and evaluation of the reliability are presented in later subsections.

4 Specification of uncertain parameters

Even when there is uncertainty in the parameter space, not all the parameters necessarily have probabilistic values. Often there are considerable dissimilarities between parameters whose values cannot be determined definitively. Since it has been established that the variability of parameter estimates significantly affects the quality metric of the architecture [8,24,48], it is important to capture the variability characteristics as accurately as possible. Given these considerations, we comprehend architecture parameters as a mix of precise and imprecise sets.

4.1 Probability distributions

As a means to capture heterogeneous uncertainties in parameter estimation, we propose to use generalised probability distributions. A parameter in an architecture specification is considered as a random variable, whose variability is characterised by its—continuous or discrete—distribution. For the parameter specifications in the architecture descriptions we propose a generic notation that can cater for any distribution. The specification is given as a parameter list, starting with a unique identifier assigned to the distribution. Some examples for Probability Density Function (PDF) specifications are given at Table 1.

4.2 Mapping uncertainty into PDFs

The proposed approach makes it possible to combine diverse sources that affect the nominal value of the parameter and consider their impact on the quality evaluation in addition to the conventional point estimates. Some guidelines to obtain the PDFs at the design stage can be given as follows:

- *Derive from the source variations* The uncertainty of parameters is often influenced by the origin of components. Information from hardware manufactures, third party software vendors or system experts is useful in

Table 2 Parameter specification of software and hardware elements of the architecture

Comp. ID	$wl(MI)$	q_0	
(a) *Software Components*			
0	1.2	0	
1	0.6	0	
2	0.4	$DISCRETE$, 0.03, 0.2, 0.3, 0.4, 1.5, 0.2, 3, 0.2	
3	1	0	
4	0.4	$NORMAL$, 0.3, 0.075	
5	0.4	$NORMAL$, 0.3, 0.075	
6	0.4	0	
7	0.4	0	
8	0	$DISCRETE$, 0.01, 0.2, 0.1, 0.4, 0.5, 0.2, 1, 0.2	
Trans $c_i \rightarrow c_j$	$p(c_i, c_j)$	ds (KB)	
(b) *Component Interactions*			
$0 \rightarrow 6$	$DISCRETE$, 0.05, 0.2, 0.5, 0.4, 2.5, 0.2, 5, 0.2	2	
$0 \rightarrow 7$	$DISCRETE$, 0.05, 0.2, 0.5, 0.4, 2.5, 0.2, 5, 0.2	2	
$1 \rightarrow 3$	1	2	
$2 \rightarrow 1$	1	2	
$3 \rightarrow 0$	1	2	
$4 \rightarrow 0$	$GAMMA$, 0.7	1	
$4 \rightarrow 3$	$GAMMA$, 0.3	2	
$5 \rightarrow 0$	$GAMMA$, 0.7	1	
$5 \rightarrow 3$	$GAMMA$, 0.3	2	
$8 \rightarrow 0$	1	0	
ECU ID	ps (MIPS)	fr (h^{-1})	ec (mW)
(c) *ECUs*			
0	4	$DISCRETE$, $4 \cdot 10^{-5}$, 0.2, $4 \cdot 10^{-4}$, 0.4, $4 \cdot 10^{-3}$, 0.2, 0.04, 0.2	$GAMMA$, 2
1	2	$DISCRETE$, $4 \cdot 10^{-5}$, 0.2, $4 \cdot 10^{-4}$, 0.4, $4 \cdot 10^{-3}$, 0.2, 0.04, 0.2	$GAMMA$, 1
2	2	$DISCRETE$, $2 \cdot 10^{-6}$, 0.2, $2 \cdot 10^{-5}$, 0.4, $2 \cdot 10^{-4}$, 0.2, $2 \cdot 10^{-3}$, 0.2	$NORMAL$, 2.5, 0.625
3	2	$DISCRETE$, $1 \cdot 10^{-5}$, 0.2, $1 \cdot 10^{-4}$, 0.4, $1 \cdot 10^{-3}$, 0.2, 0.01, 0.2	$NORMAL$, 3, 0.75
4	11	$DISCRETE$, $8 \cdot 10^{-5}$, 0.2, $8 \cdot 10^{-4}$, 0.4, $8 \cdot 10^{-3}$, 0.2, 0.08, 0.2	$NORMAL$, 4, 1
5	11	$DISCRETE$, $2 \cdot 10^{-5}$, 0.2, $2 \cdot 10^{-4}$, 0.4, $2 \cdot 10^{-3}$, 0.2, 0.02, 0.2	$GAMMA$, 2

Table 2 continued

BUS ID	dr (KBPS)	$fr\ (h^{-1})$	ec (mW)
(d) *Buses*			
0	128	$BETA\_SHD, 3 \cdot 10^{-6}, 3 \cdot 10^{-4}, 10, 2$	$NORMAL, 3, 0.75$
1	64	$BETA\_SHD, 1.2 \cdot 10^{-5}, 1.2 \cdot 10^{-3}, 10, 2$	$GAMMA, 2$
2	64	$BETA\_SHD, 4 \cdot 10^{-6}, 4 \cdot 10^{-4}, 10, 2$	$GAMMA, 2$

characterising the uncertainty in specific parameters. In some situations, the distribution of the source variables can be obtained and consequently, the desired parameter's distribution can be approximated from its sources.

Example The failure rate (λ) of an ECU is a function of its ambient temperature (T in Kelvin) [12], such as $\lambda = 4 \cdot 10^{-6} \times T + 100$. Consider an automotive electronic system where the temperature profile around ECU X varies between 300K and 400K, has a 370K mode and is negatively skewed. The PDF of λ of ECU X can be derived and specified as $\lambda_X = BETA\_SHD, 400 \times 4 \cdot 10^{-6}, 500 \times 4 \cdot 10^{-6}, 10, 2$.

- *Histogram approximation* Prior information on the parameters may be available. For certain parameters, large numbers of raw data may be available as a result of testing. In such situations, the PDFs can be approximated from the histograms of the raw data.

Example In functional test executions of the system model, the histogram of the test results indicated that the message transfer probability from component C_i to component C_j is normally distributed. The average of the samples is 0.2 with a variance of 0.04. Therefore, the transfer probability can be given as $p_{i,j} = NORMAL, 0.2, 0.04$.

- *Uniform approximation* It is common to have limited information on the range of the variation without any specification on variation within the range. Uniform distributions can be used to approximate such situations.

Example The system has a need to communicate with a new external service X, of which we only know that its worst case response time is 1.0 s. The communication link takes at least 5ms for the data transfer. $rt = UNIFORM, 5 \cdot 10^{-3}, 1.0$.

- *Specify distinct information as a discrete-sample distribution* : In cases where the a parameter can only vary within a discrete set, discrete-sample distributions can be used to capture it. This is a very powerful feature in our approach as in most of the practical situations it is relatively easy to obtain discrete estimates.

Example Experts have indicated that the request rate (rr) for a service X can be either 200 or 800 per second. In 75 % of the cases it is 200. This will be given as $rr = DISCRETE, 200, 0.75, 800, 0.25$.

4.3 Illustration using the example

Not every parameter pertaining to the current example is subject to uncertainty. For instance, the processing speed (ps) of

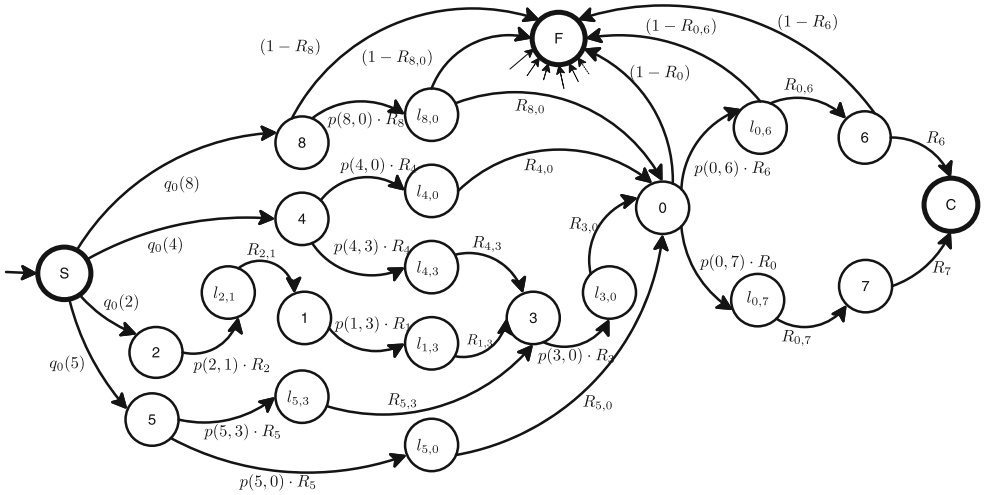

Fig. 3 Annotated DTMC for service reliability evaluation. Note that only a few of the possible failure transitions are included in the diagram as an illustration. Similar arcs should be drawn from each transient state to the state F

an ECU or the computational load (wl) of a software component can realistically be considered fixed and deterministic. However, parameters such as the failure rates of ECUs, failure rates of buses, execution initiation probabilities and transition probabilities are subject to uncertainty and have to be estimated. Table 2 shows an example set of parameters.

The probabilistic specification of parameters in the tables reflects the variability of these parameters in relation to the automotive ABS system. It is realistic to assume that even the same parameter of different architectural elements can have very different uncertain characteristics [3,12,25,40]. For example, execution initialisation probabilities of components 4 and 5 depend on vehicle dynamics in various road conditions, whereas the uncertainty of the execution initialisation probability of component 8 arises from probabilistic user actions from a discrete set of choices. Hence, within the same column, we may have mentioned different PDFs as well as distinct values.

5 Probabilistic model construction

5.1 Propagation of uncertainty in models

The specification of architectural elements as discussed above has to be incorporated into the quality evaluation. Different quality attributes can be evaluated using different modelling approaches as discussed in the introduction. In the case of probabilistic quality attributes, the evaluation models are also probabilistic. The model parameters are often derived from the parameters of the architectural elements. This process results in one-to-one, one-to-many, many-to-one or many-to-many relationships of architecture parameters to the parameters of probabilistic model. As we have incorporated probabilistic specifications for some of the architectural elements's properties, the probabilistic notion is transformed to the evaluation model parameters. Due to the fact that the inputs are probability distributions, the resulting evaluation model parameters become probability distributions or functions of probability distributions.

5.2 Illustration using the example

In order to obtain a quantitative estimation of the reliability of the automotive architecture in focus, a well-established DTMC-based reliability evaluation model [14,23] is used. From the software components and hardware specification, an absorbing DTMC [49] is constructed for each subsystem such that a node represents the execution of a component and arcs denote the transfer from the execution of one component to that of another. Figure 3 shows the DTMC for the example case. The digits in the node labels point to the corresponding nodes in Fig. 2a. Single-digit nodes represent execution of a software component, and $l_{i,j}$ labelled nodes denote the communications among software components. A super-initial node [50] has been added to represent the start of the software execution, and arcs originating at the node have been annotated with relevant execution initialization probabilities (q_0). Two new absorbing states C and F have been added, representing the *correct output* and *failure* states, respectively. Failures during execution have been mapped to arcs from each execution node c_i to the F state with a probability $(1 - R_i)$ and communication failures are mapped to arcs from each communication node l_{ij} to F with a probability $(1 - R_{ij})$, where R_i, R_{ij} represent component reliability and execution link reliability, respectively. Note that only a few of these failure transitions have been added for the clarity of the figure.

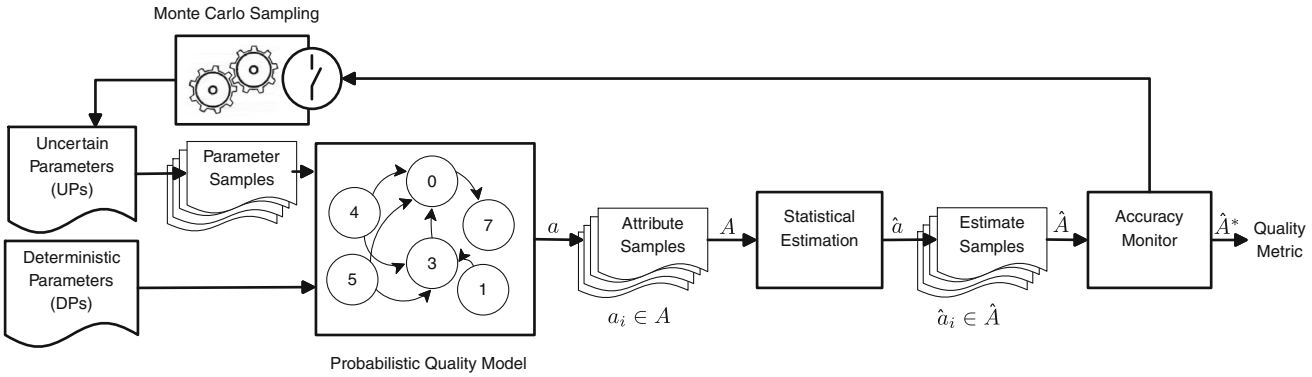

Fig. 4 Monte Carlo simulation

In the resulting DTMC we find only two absorbing states (i.e. C and F) and the reliability of the ABS service can be defined as '*successfully reaching the state* C, *without being absorbed at* F'. In terms of the property specification, a failure of the system is a probabilistic reachability property which can be given as '*eventually reaching the state* F'. Hence, the reliability property can be specified in PCTL notation,

$$R = 1 - \mathcal{P}_{=?}(\Diamond\ F) \tag{2}$$

where $\mathcal{P}_{=?}$ denotes the probability of satisfying the property. Once all the transition probabilities of the DTMC in Fig. 3 have been obtained, the property formula 2 can be computed by solving the DTMC [14,49] manually or with the use of a probabilistic model checking tool like PRISM [34]. However, in order to compute the transition probabilities, the following quantities have to be defined.

The failure rates of the execution elements can be obtained from the ECU parameters. The execution time is defined as a function of the software-component workload and the processing speed of its ECU. Similar to the models used in [3,39], the reliability of the ABS system considers both ECU and communication link failures. The reliability of a component c_i can be computed using Eq. 3,

$$R_i = e^{-fr(d(c_i)) \cdot \frac{wl(c_i)}{ps(d(c_i))}} \tag{3}$$

where $d(c_i)$ denotes the ECU allocation of component c_i. A similar computation can be employed to establish the reliability of the communication elements [39], which, in our model, is characterised by the failure rates of the hardware buses, and the time taken for inter-component communication, defined as a function of the buses' data rates dr and the size of the data exchange ds between the software components. Therefore, the reliability of the communication between component c_i and c_j is defined according to Eq. 4.

$$R_{ij} = e^{-fr(d(c_i), d(c_j)) \cdot \frac{ds(c_i, c_j)}{dr(d(c_i), d(c_j))}} \tag{4}$$

Some of the entries for the parameters in Table 2 (e.g. $p_{i,j}$, fr_i, $fr_{i,j}$, q_{0_i}) are probability distributions. These parameters form part of the transition probabilities of the DTMC given in Fig. 3. Furthermore, R_i and R_{ij} in formulations (3) and (4) are influenced by the probabilistic specifications of fr_i and $fr_{i,j}$ in Table 2. Hence, the transition probability values of the DTMC become functions of probability distributions rather than distinct numerical values. Computing the reliability (i.e. reachability property given in Eq. 2) requires solving the annotated DTMC. However, the analytical reachability calculation [49] or Markov chain solving tools like PRISM cannot be used in this situation, since transition probabilities are mathematical functions with probability distribution variables rather than numerical values.

6 Quality metric estimation

The probabilistic model with partially uncertain parameter space has to be evaluated to obtain the quantitative metric of the quality of the system architecture at hand. It has been emphasised before that these models are often hard to represent as linear mathematical functions. When many parameters are uncertain with diverse distributions, the quantitative metric as a distribution cannot be derived analytically. Consequently, the Monte Carlo(MC)-based approach presented here draws samples from the probability distributions of input parameters.

Figure 4 illustrates the architecture evaluation process using MC simulation. The input of the MC simulation of the probabilistic model (PM) is a set of parameters specified as probability distributions (UPs) as well as deterministic/certain parameters (DPs).

6.1 Monte Carlo simulation

The MC simulation takes samples from input probability distributions of the architectural elements within the probabilistic evaluation model. Any single parameter of an architectural element may contribute to more than one parameter in the evaluation model. Every time a sample is taken from an input distribution, all model parameters dependent on this parameter have to be updated. The resulting strategy for a single run of the MC simulation is explained in the following:

1. *Sample* A sample is taken from the Cumulative Distribution Function (CDF) of each parameter. The *inverse transformation method* [47] can be used for this process.
2. *Update* Using the samples drawn from the input distributions, the numerical values for the evaluation model parameters are updated. Since more than one parameter of the probabilistic model may be dependent on a parameter in the architecture, a Publisher–Subscriber paradigm can be used. Whenever a sample is taken for a specific architectural parameter, all the subscribing model parameters are updated and recomputed.
3. *Resolve dependencies* The model-specific parameter dependencies are solved in this phase. The numerical values for the outgoing transition probabilities are normalised to comply with the model assumptions.
4. *Compute* The model is analytically solved/simulated and the quantitative metric of the system quality is obtained.

A single run of this MC simulation results in one numerical value for the quality attribute (a). Due to the probabilistic inputs (UPs), the values obtained from different runs ($\{a_1, a_2, a_3, \ldots, a_N\} = A$) are most often not identical. From the software architect's point of view, a single statistical index of a quality metric ($\hat{a}$) is desirable despite the uncertainty. The level of tolerance in the statistical estimation depends on the application domain and the quality attribute. Depending on the character of the system to be designed, the expected value, variance, quartiles, confidence intervals and worst-case values have been used to describe the quality of a system.

One important challenge regarding this estimation is that the actual distribution of the quality metric (A) is unknown. The existing uncertainty analysis techniques in software architecture evaluation have a prior assumption regarding the distribution of the metric (A). With some exceptions [17,52], most studies assume a normal distribution [16,28]. Due to the heterogeneity of the uncertainty in the input parameters, and the non-linearity and complexity of the model evaluation techniques, the resulting quality distribution after the MC simulation is unpredictable. For our approach, we introduce a generalised estimation of ($\hat{a}$), using the flexible percentiles while supporting the expected/worst case measures.

6.2 Distribution estimation

The statistical data ($A = \{a_1, a_2, a_3, \ldots, a_N\}$) obtained from the MC runs can be processed using statistical methods to identify parameters of a candidate distribution. Possible approaches are the method of maximum likelihood and the method of moments, as well as Bayesian estimation [42]. These methods can be applied when prior information about the distribution of the resulting quality metric (A) is available. Due to the diverse nature of the input parameter distributions and the complexity of the quality evaluation models, estimating the prior distribution is hard and computationally expensive, since it would have to be repeated for each architecture evaluation.

6.3 Non-parametric estimation

Non-parametric estimation has the major advantage of not making any assumptions about the probability distribution of the population (A). Non-parametric methods lend themselves to providing a generic estimation for flexible percentile estimates ($\hat{A}$).

Instantaneous objective values for each MC run ($A = a_1, a_2, a_3, \ldots, a_N$) are stored and sorted in ascending or descending order. Percentile estimates can be obtained from retrieving the appropriate position in the array.

Example Assume the quantitative predictions reliability of an architecture X for each MC run (A) have been captured in an array $S = s_1, s_2, s_3, \ldots, s_N$ and sorted in ascending order. The 95th percentile of reliability for architecture X is easily obtained calculating the index $i = N * 95/100$ of the required entry.

7 Dynamic stopping criterion

All of the estimation techniques discussed above sample from appropriate distributions and obtain a desired statistical index of a quality attribute $\hat{a}$. However, the accuracy of the estimate $\hat{a}$ strongly depends on the sample size, i.e. on the number of MC trials carried out. One important characteristic of the problem is that the actual value of the estimate ($\hat{a}$) or the distribution of A is not known. Large numbers of MC runs cannot be conducted because given the large number of candidate architectures produced in stochastic optimization, the computational expense is prohibitive.

7.1 Sequential statistical significance test

To solve this issue, we propose a dynamic stopping criterion based on accuracy monitoring. In this approach, the assumptions on the monitored distribution (A) are relaxed

by transforming the monitoring phase to the estimate $\hat{A}$. A statistical significance test is carried out on the samples of the statistical index ($\hat{A}$).

- A minimum of k MC executions (a_1, .., a_k) are conducted before estimating the desired index $\hat{A}$. After k repeats, one of the methods discussed in Sect. 6.3 can be used to obtain each $\hat{a}$.
- The variation of the estimate $\hat{A} = \{\hat{a}_1, \hat{a}_2, \hat{a}_3, \ldots, \hat{a}_k\}$ is monitored for a sliding window of size k. Only the last k samples of the estimate $\hat{A}$ are monitored, as the accuracy of the estimation is a changing property. The objective is to detect when sufficient accuracy has been obtained.
- The statistical significance is calculated for the last k estimates [47]:

$$w_r = \frac{2z_{(1-\alpha/2)}}{\sqrt{k}} \frac{\sqrt{\overline{\hat{a}^2} - \left(\overline{\hat{a}}\right)^2}}{\overline{\hat{a}}} \tag{5}$$

where w_r is the relative error, $\overline{\hat{a}}$ is the average of last k estimates, $\overline{\hat{a}^2}$ is the mean-square of the last k estimates, α is the desired significance of the test and z refers to the inverse cumulative density value of the standard normal distribution. The relative error w_r of the estimate $\hat{A}$ is checked against a tolerance level, e.g. 0.05. The complete process is explained in Algorithm 1.

Algorithm 1: Monte Carlo simulation with dynamic stopping criterion

```
1  i = 1, j = 1;
2  while w_r > tolerable w_r do
3      a_i := conduct one MC execution();
4      if i ≥ k then
5          â_j := compute appropriate statistical index using
           (a_1, a_2, a_3, ..., a_i);
6          if j ≥ k then
7              â_bar = (Σ_{p=j-k}^{j} â_p) / k;
8              â²_bar = (Σ_{p=j-k}^{N} â²_p) / k;
9              w_r = (2 z_(1-α/2)) / √k · √(â²_bar − (â_bar)²) / â_bar;
10         end
11         j++;
12     end
13     i++;
14 end
15 â* = â_j;
```

It should be noted that in Algorithm 1, the parameters epoch size (k) and significance (α) can be set independently of the architecture evaluation problem.

7.2 Illustration using the example

The above algorithm can be applied to the example case as follows:

1. The DTMC reliability model given in Fig. 3 is constructed from the architecture specification. The parameters are sampled according to the specifications in Table 2.
2. Using the instantaneous samples of the input parameters, the transition probabilities of the DTMC are computed. From the sampled failure rates of ECUs and buses, the reliabilities can be calculated using formulae (3) and (4).
3. Model-specific parameter dependencies are resolved. In the DTMC-based reliability model, model parameters are normalised to satisfy the following properties.
   - The sum of all outgoing transitions from any non-absorbing node must be equal to 1.
   - The sum of execution initialization probabilities (q_0's) must be equal to 1.
4. Now that numerical values have been calculated for the transition probabilities of the DTMC, the reachability property (Formula 2) can be computed. This process represents a single MC run, which yields a reliability value $a_i \in A$.
5. The steps (1) to (4) are repeated $k = 10$ times, after which the accuracy estimation process starts. Assuming the goal of 90 % significance ($\alpha = 0.1$), for k samples, the initial estimate of reliability ($\hat{a_1}$) is computed using a non-parametric percentile estimation of the tolerance level (5th percentile) described in Sect. 6.3.
   All existing samples are used to estimate $\hat{a}$ after each MC run. When k estimates $\hat{a}$ are available, the dynamic stopping criterion in Eq. 5 is applied.
6. If w_r is less than a threshold, the last $\hat{a}$ is considered as the 5th percentile reliability of the architecture. Otherwise, the process repeats from step 1. When the stopping criterion is reached, the final estimate of $\hat{a}$ is taken as the quality metric $\hat{a}^* \in \hat{A}^*$.

8 Experiments

8.1 Experiments on the example

Reliability The first set of experiments validating the MC simulation process has been conducted on reliability evaluation using the properties of the case study. In each MC run, samples were taken from the input parameter specification, and a DTMC-based reliability model was created as described. The DTMC was expressed as a PRISM [34] model (.pm) and reachability formulae (2) were evaluated by executing the PRISM model checker. The results of 3000 MC trials are

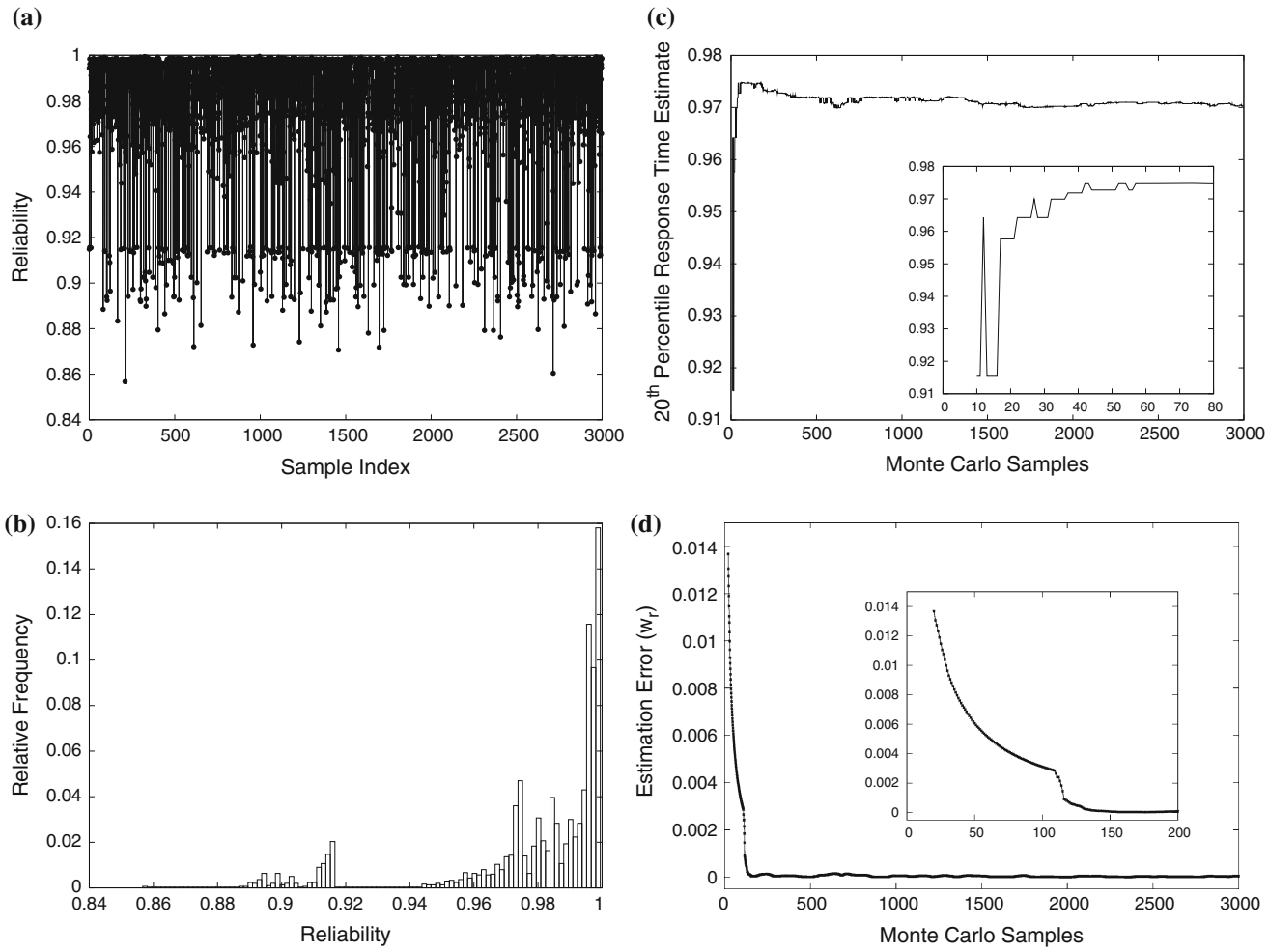

Fig. 5 Results of the experiments with ABS system example. **a** Instantaneous samples of reliability in MC trials. **b** Histogram of reliability samples. **c** Variation of the estimate over the MC runs. **d** Relative error of sequential accuracy monitoring

presented in Fig. 5. The samples for each MC run, taken as described in step 4 in Sect. 7.2, are distributed as depicted in Fig. 5a. It can be seen that the values for the reliability vary from 0.85 to 0.999, which is a significant variation with respect to the notion of reliability. The histogram of the reliability obtained from 3000 samples in Fig. 5b shows that the conventional assumption of a normal or Weibull distribution of the system reliability is not applicable to the actual characteristics of the sample distribution obtained from the MC simulation.

The goal of the MC run was set to the 20th percentile, i.e. the reliability of the system will be greater than the estimate for 80 % of the cases. The epoch size k was set to 10. The values of each estimation is presented in Fig. 5c (note that the graph starts at sample size of 10). Considerable variations can be observed at the early estimations, while an observable stabilisation is achieved after around 100 MC runs. These variations are reflected in the error values (Fig. 5d). The stopping criterion uses a significance test configuration of $\alpha = 0.05$ and $w_r = 0.0005$, which leads to the MC simulation achieving the required accuracy at 156 runs, with the estimate for the 20th percentile reliability of 0.974527.

The reliability evaluation of the case study which is an instance of the reachability property of the absorbing Markov chain. The attribute evaluation method has also been applied to the evaluation of response time and energy consumption, which represent the nature of path properties of the Markov chain.

Response time The response time metric of the ABS system is defined as the '*when a break service is triggered, the time taken to complete the breaking activity*'. To assess this property it is necessary to consider the execution times of software components, the scheduling delays as well as the communication among the components in executing the function. The time taken to process a request in a software component c_i is denoted by t_i and can be considered as a function of the execution load (wl) of the component and the processing speed (ps) of the ECU in which the component is deployed.

Using the same notation scheme described in Sect. 5.2 where $d(c_i)$ denotes the ECU allocation of component c_i, $wl(c_i)$ to denote the workload of component c_i and $ps(d(c_i))$ representing the processing speed of the ECU of component c_i, t_i can be given as follows:

$$t_i = \frac{wl(c_i)}{ps(d(c_i))} \tag{6}$$

Similarly, the communication time between software components c_i and c_j is denoted by t_{ij} and computed using bus speed (dr) and communication data size (ds).

$$t_{ij} = \frac{ds(c_i, c_j)}{dr(d(c_i), d(c_j))} \tag{7}$$

In formula 7, $ds(c_i, c_j)$ represents the data size of the communication between c_i to c_j, and $dr(d(c_i), d(c_j))$ denotes the data rate of the communication channel between the ECUs where c_i and c_j are deployed.

In modelling the response time considering the aforementioned two factors, the system model is considered as a Markov Reward Model where execution and communication delays are treated as state rewards. The execution states of the DTMC given in Fig. 3 are annotated with the state reward structure of execution time (t_i) values and communication nodes in the DTMC are annotated with communication delay (t_{ij}) rewards. Consequently, the response time property can be specified as a PCTL formula

$$RT = \mathcal{R}_{=?\,(\text{“time rewards”})}(\Diamond C) \tag{8}$$

where $\mathcal{R}_{=?\,(\text{“time rewards”})}$ denotes the expected time rewards accumulated in all the paths that lead to the satisfying states. Note that not only the transition probabilities but also the state rewards of the model become variables (functions of uncertain parameters) which prevent the application of conventional model evaluation techniques to the computation of the property.

The results of the evaluation of the system response time in the presence of uncertain parameters are given in Table 2. As in the experiments on reliability evaluation, the PRISM model checker was used to evaluate the reward property for models generated in each MC sample. The results of 3000 MC trials are presented in Fig. 6. The histogram of the property obtained from the samples is depicted in Fig. 6b, whose shape is also not indicative of a normal distribution. Considering the nature of the property, the goal of the estimation was set to the 75th percentile, i.e. the response time of the system will be lower than the estimate for more than 75 % of the uncertain cases. The epoch size k was set to 10. The values of each estimation is presented in Fig. 6c. Considerable variations can be observed at the early estimations, while an observable stabilization is achieved after around 100 MC runs. These variations are reflected in the error values (Fig. 7d). The stopping criterion uses a significance test configuration of $\alpha = 0.05$ and $w_r = 0.0005$, which leads to the MC simulation achieving the required accuracy at 166 runs, with the estimate for 75th percentile response time of 7.2506 ms.

Energy consumption With emerging trends towards energy-efficient embedded software design, energy consumed in electronics is becoming an important property. Model-based evaluation of energy consumption helps the software architects make better design decisions. In the automotive ABS example, the energy consumed in a single execution of the service is an important quantity, especially with regard to the current trend towards electric cars.

The evaluation of this property requires the consideration of energy consumed during the execution of software and the energy spent for communication among the components in executing the service. The energy consumed in processing a request in a software component c_i is denoted by e_i, and can be considered as a function of execution load (wl) of the component, and the processing speed (ps) and energy dissipation rating (er) of the ECU in which the component is deployed.

$$e_i = \frac{wl(c_i)}{ps(d(c_i))} \cdot er(d(c_i)) \tag{9}$$

The same notation scheme from the previous section is used here where $d(c_i)$ denotes the ECU allocation of component c_i, $wl(c_i)$ to denote the workload of component c_i and $er(d(c_i))$ representing the energy dissipation rating of the ECU of component c_i.

Similarly, the energy consumed in a single communication between software components c_i and c_j is denoted by e_{ij} and computed using the energy dissipation rating of the bus (er) and communication time.

$$e_{ij} = \frac{ds(c_i, c_j)}{dr(d(c_i), d(c_j))} \cdot er(d(c_i), d(c_j)) \tag{10}$$

In the above formula, $ds(c_i, c_j)$ represents the data size of the communication between c_i to c_j, and $dr(d(c_i), d(c_j))$ and $er(d(c_i), d(c_j))$ denote the data rate and energy dissipation rating of the communication channel between the ECUs where c_i and c_j are deployed, respectively.

In order to model the energy consumed in electronics during one execution of the ABS service, the system model is considered as a Markov Reward Model where execution and communication energy consumptions are treated as state rewards. The execution states of the DTMC given in Fig. 3 are annotated with the state reward structure of processing energy consumption (e_i)s and the communication nodes in the DTMC are annotated with communication energy consumption (e_{ij}) rewards. Accordingly, the response time property can be specified as a PCTL formula

$$EC = \mathcal{R}_{=?(\text{“energy consumption rewards”})}(\Diamond C) \tag{11}$$

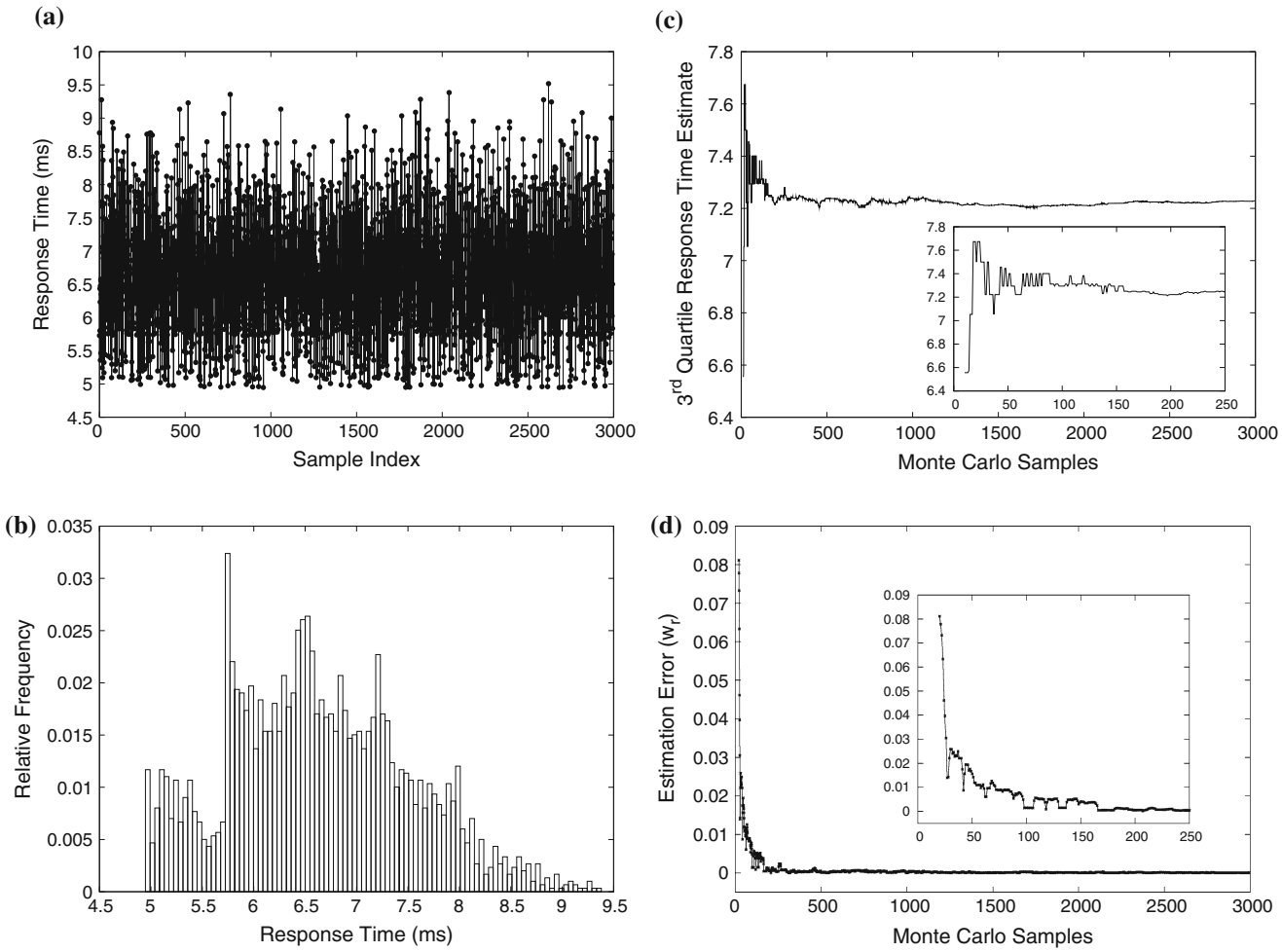

Fig. 6 Results of the response time estimation of the ABS system example. **a** Instantaneous samples of response time in MC trials. **b** Histogram of response time samples. **c** Variation of the estimate over the MC runs. The graph starts at a sample size of 10. **d** Relative error of sequential accuracy monitoring

which captures the expected energy rewards cumulated over all paths to execution completion (state C).

The energy consumption of the architecture at hand is evaluated considering the uncertain parameters given in Table 2. The results of 3000 MC trials obtained using the PRISM model checker are presented in Fig. 7. The histogram of the property obtained from the samples is depicted in Fig. 7b. The goal of the estimation was set to quantify the mean energy consumption under the uncertain input parameters. The epoch size k was set to 10. The values of each estimation is presented in Fig. 7c. As with the other two probabilistic property estimations, the variations observed during the early stages stabilise as the sampling process continues. These variations are reflected in the error values (Fig. 7d). The stopping criterion uses a significance test configuration of $\alpha = 0.05$ and $w_r = 0.0005$, which leads to the MC simulation achieving the required accuracy at 235 runs, with the estimate for average energy consumption of 18.0126 mJ.

8.2 Experiments on generated problems

Experimental setup In addition to the three probabilistic property evaluations of the automotive case study, a series of experiments have been conducted to further investigate the presented architecture evaluation approach under uncertainty. The scalability of the approach is explored by generating instances from a range of problem sizes. Without loss of generality, the objective of each problem is estimation of reliability.

– To represent the uncertainties in the component reliability figures, each component's reliability is specified as a random distribution in the range 0.99–0.9999. For a single

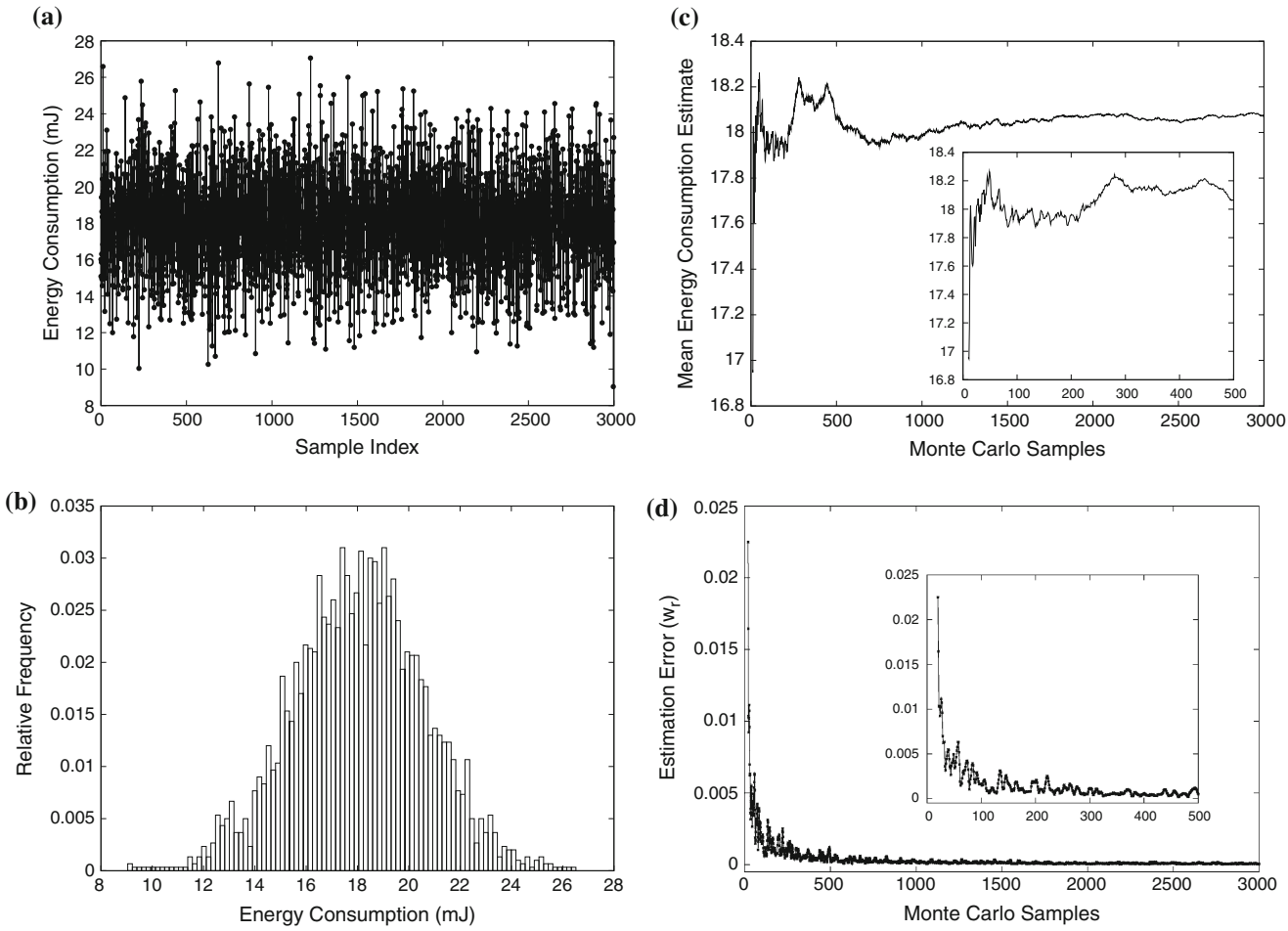

Fig. 7 Results of the energy consumption estimation of the ABS system example. **a** Instantaneous samples of energy consumption in MC trials. **b** Histogram of energy consumption samples. **c** Variation of the estimate over the MC runs. **d** Relative error of sequential accuracy monitoring

problem, the distributions remain unchanged throughout the experiments to maintain the comparability of the results.

– Additional distributions are introduced to represent uncertainties associated with other parameters. The number of additional uncertainties is varied from 0 to 10 for each problem size in order to investigate the level of uncertainty in different instances. Parameters with uncertainty draw their values from Normal, Uniform, Beta, Shifted Beta, Gamma, Exponential, Weibull and Discrete distributions.
– In order to represent diverse architecture design problems, the DTMC is constructed using random relationships between components. Therefore, a parameter may have an effect on randomly selected transition probabilities in the generated DTMC.
– The support for different levels of compromise in the estimation process is captured by optimizing each problem instance for median, 25th percentile (75 % pessimistic), 5th percentile (95 % pessimistic) of the reliability. The dynamic stopping criterion is applied with a significance level of $\alpha = 0.05$, an error margin of $w_r = 0.005$ and an epoch size of $k = 10$.

The configurations for the problem instances are given in Table 3.

Results Table 4 lists the results for the expected value, 25th percentile (75 % pessimistic), 5th percentile (95 % pessimistic) of the reliability using the 16 problem instances and 3 classes of tolerance described in Table 3. In the table, N refers to the number of MC evaluations needed to satisfy the stopping criterion. The estimation of the quality at the stopping condition is listed in the columns $\hat{a}^*$.

The MC simulations were carried out for a large number of evaluations (10,000), even after the stopping criterion had been met. The final estimation a_f obtained from 10,000 runs was compared with the estimation achieved at the stopping condition. The column labelled d_r indicates the relative

Table 3 Experiment configurations

Case ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DTMC nodes	10	10	10	10	20	20	20	20	50	50	50	50	100	100	100	100
Additional uncertainties	0	2	5	10	0	2	5	10	0	2	5	10	0	2	5	10

Table 4 Results of the randomly generated experiments against 16 problem instances and 3 classes of tolerance

Case ID	Median (50th percentile)				25th percentile (75 % pessimistic)				5th percentile (95 % pessimistic)			
	N	$\hat{a}^*$	a_f	d_r	N	$\hat{a}^*$	a_f	d_r	N	$\hat{a}^*$	a_f	d_r
1	19	0.952560	0.954734	0.002277	19	0.945523	0.948031	0.002646	19	0.935858	0.002277	0.003700
2	19	0.965903	0.962922	0.003097	19	0.954604	0.957312	0.002828	19	0.949370	0.003097	0.000109
3	19	0.966705	0.968018	0.001356	19	0.962980	0.961934	0.001088	19	0.934353	0.001356	0.018801
4	19	0.942744	0.932468	0.011020	19	0.918319	0.918192	0.000138	23	0.903614	0.011020	0.013062
5	19	0.897277	0.902449	0.005732	19	0.890973	0.892493	0.001703	19	0.880296	0.005732	0.002291
6	19	0.913447	0.907576	0.006469	19	0.900099	0.899102	0.001109	26	0.877142	0.006469	0.011810
7	19	0.912154	0.916944	0.005224	19	0.908060	0.908293	0.000256	19	0.888432	0.005224	0.007256
8	19	0.966131	0.965325	0.000834	19	0.961862	0.960805	0.001101	19	0.948812	0.000834	0.000323
9	19	0.784776	0.785679	0.001149	30	0.767856	0.773189	0.006897	19	0.762634	0.001149	0.007971
10	28	0.739862	0.736462	0.004617	21	0.721135	0.721837	0.000972	19	0.698968	0.004617	0.003520
11	19	0.783287	0.778369	0.006317	19	0.774306	0.762080	0.016042	133	0.725934	0.006317	0.012663
12	19	0.771074	0.748191	0.030584	125	0.727915	0.722666	0.007264	257	0.691504	0.030584	0.013951
13	19	0.592317	0.594764	0.004113	19	0.579529	0.580552	0.001761	48	0.562311	0.004113	0.003324
14	64	0.593832	0.593420	0.000693	34	0.572371	0.579086	0.011595	19	0.545807	0.000693	0.021654
15	33	0.584036	0.589625	0.009480	19	0.576236	0.573726	0.004375	21	0.563660	0.009480	0.020882
16	269	0.536075	0.530330	0.010832	241	0.481548	0.483096	0.003204	19	0.438971	0.010832	0.042167

N number of MC evaluations at reaching stopping criterion, $\hat{a}^*$ estimation at reaching stopping criterion, a_f estimation at 1000 MC runs, d_r relative difference between a_f and $\hat{a}^*$

difference between $\hat{a}^*$ and a_f calculated as

$$d_r = \left(\frac{\hat{a}^* - a_f}{a_f}\right)^2 \tag{12}$$

In all cases, the relative difference d_r is smaller than the relative error w_r calculated by Eq. 5. The results show that the approach is applicable to different sizes of the problem as well as diverse levels of uncertainty. The accuracy of the MC simulations complies with the specified tolerance levels. It should be noted that the novel stopping criterion controls the process with the effect of saving large number of computational resources. For example in 5th percentile estimations, many cases have achieved sufficient accuracy with a small number of MC runs, while cases like 12 are automatically continued for longer to obtain an accurate estimation.

Figure 8 illustrates a comparison of number of MC runs when the dynamic stopping criterion is reached with different problem configurations. Even for the same number of uncertain parameters, large problem instances exhibit higher number of MC runs to converge. This observation supports the uncertain parameter generation scheme used in the problem generation. Since we have let a model parameters to take an arbitrary relationship to uncertain input parameters, larger problems use the uncertain ones more often than the small problem instances. Therefore, the large problem instances would require higher number of MC samples to obtain a stable statistical index of the quality attribute.

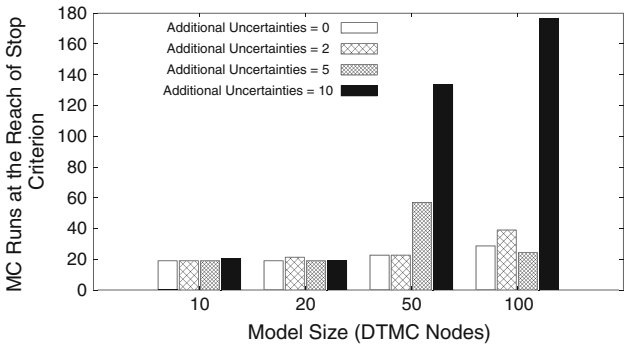

Fig. 8 Average dynamic stopping size for different problem configurations

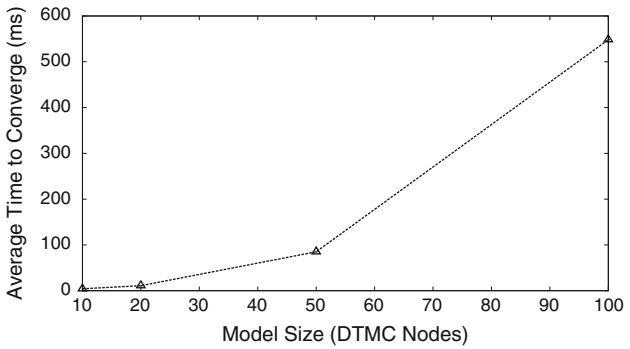

Fig. 9 Scalability of the approach

In order to investigate the scalability of the approach, the average time taken to reach the dynamic stopping criterion in each problem instance has been recorded. Figure 9 depicts the time taken in milliseconds averaged over the three types of experiments (i.e. median, 25th percentile and 5th percentile estimations). The results indicate that the time needed for reaching the stopping criterion grows exponentially with respect to the problem size. It should be noted that this growth has strong relationship with the quality evaluation functions as well. When the problem gets bigger, complex quality evaluation functions like Markov Chain-based evaluations require exponentially growing computation time [19].

8.3 Discussion of results

The experiments on the case study illustrated that the new DTMC-based evaluation model is able to include heterogeneous uncertainties associated with parameters into architecture specification and use them in probabilistic quality evaluation. The input parameter specification shows the capability to cater for different combinations of probability distributions to specify various characteristics of the parameters. The evaluation of three probabilistic properties using the models for uncertain input parameters has illustrated the applicability of the approach to probabilistic models.

The experiments also indicate that the resulting distributions of the probabilistic properties are not necessarily normal (e.g. Figs. 5b, 6b), further justifying the inapplicability of methods which depend on the assumption of normality. As a result, the conventional assumptions on input parameter distributions have been relaxed and extended support has been introduced for heterogeneous software architecture parameters. Furthermore, assumptions on distributions of the quality metric have been relaxed, and distribution-free statistics has been adopted to support different levels of tolerance. Conventional limitations on mean, variance of the desired quality analysis have been mitigated with the further support of quartile and percentiles. This extends the applicability of the approach to domains like safety–critical embedded systems, where more pessimistic estimates are necessary for certain quality attributes.

The goal settings of 20th percentile, third quartile and mean for the three probabilistic properties also demonstrate the ability of the approach to use various statistical indices as measures of the quality attributes under uncertainty. The contribution of the new dynamic stopping criterion is evident in the evaluation of all three properties, where the estimation and error variation graphs shows the stabilisation of the estimation process. As a result, the approach has provided quantitative metrics for the probabilistic properties including the aspect of robustness with respect to the uncertainty associated with the input parameters. The architect is given the flexibility to specify uncertain input parameters and the level of required tolerance against the uncertainty. The quantitative metric produced from the new approach includes additional decision support with a confidence in the robustness of the solutions, which has not been possible with the conventional point-estimate-based architecture evaluation methods.

The experiments that use generated problem instances have been designed with the intention of validating the method's ability to capture heterogeneous and diverse characteristics of uncertainty as well as its applicability to probabilistic quality evaluation. These experiments also aimed at demonstrating the accurate performance of the dynamic stopping criterion. The new framework's capability of handling a diverse range of probability distributions has been validated with the experiments in which random distributions were used. It has been shown that the new approach can successfully evaluate a number of very diverse problem instances, indicating versatile applicability. The percentiles estimated by the MC-simulator have been chosen to cover moderate and more pessimistic requirements with the quality attributes in practice. The novel dynamic stopping criterion has been tested for the 16 random cases and for three different percentile estimations, and accuracy of the tests has been validated under the specified tolerance levels.

The framework presented in this paper is deliberately generic and treats the probabilistic evaluation model as a *black box*. We suggest that the evaluation model contributed here can be applied to any architecture-based evaluation model.

9 Conclusions

In this paper, we have addressed the problem of probabilistic quality evaluations in the presence of uncertainty. The evaluation framework introduced in this work provides support for heterogeneous uncertainties. Probabilistic parameter values and their evaluation have been accommodated through the use of probabilistic model checking and MC simulations. A non-parametric significance test as a stopping criterion has

significantly reduced the number of trial runs and function evaluations necessary to achieve the desired confidence level. The functionality of the framework has been illustrated using a practical case study.

In all of our experiments, the window size of the dynamic stopping criterion was set to 10. This was an arbitrary decision which is still to be substantiated. The current results indicates that this window size was appropriate for all cases considered. However, further research is required to determine the ideal window length. Ideally, the process of finding an ideal or near-optimal window size ought to be automated. In future work, we aim to apply sequential hypothesis testing principles to analyze the accuracy of the evaluation results. We also intend to investigate the applicability of the approach to multi-variate sensitivity and bottleneck analysis in architecture-based quality evaluation. We are currently working on the integration of uncertainty analysis with design space exploration with the intention of providing a method for a more robust architecture optimisation.

Acknowledgments This original research was proudly supported by the Commonwealth of Australia, through the Cooperative Research Center for Advanced Automotive Technology (projects C4-501: Safe and Reliable Integration and Deployment Architectures for Automotive Software Systems). Furthermore, the research was supported by the Center for Mathematical and Computational Modelling $(CM)^2$ at the University of Kaiserslautern.

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)
2. Ardagna, D., Ghezzi, C., Mirandola, R.: Rethinking the use of models in software architecture. In: Quality of Software Architectures. Models and Architectures, pp. 1–27. Springer, Berlin (2008)
3. Assayad, I., Girault, A., Kalla, H.: A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In: Dependable Systems and Networks, pp. 347–356. IEEE (2004)
4. Axelsson, J.: Cost models with explicit uncertainties for electronic architecture trade-off and risk analysis. In: Current Practice (2006)
5. Aziz, A., Singhal, V., Balarin, F.: It usually works: The temporal logic of stochastic systems. In: Wolper, P. (ed.) Proceedings of 7th International Conference on Computer Aided Verification, CAV 95. LNCS, vol 939, pp. 155–165. Springer, Berlin (1995)
6. Baier, C., Katoen, J.-P., Hermanns, H.: Approximate symbolic model checking of continuous-time markov chains. In: Baeten, J.C.M., Mauw, S. (eds.) Proceedings of 10th International Conference on Concurrency Theory, CONCUR 99. LNCS, vol. 1664, pp. 146–161. Springer, Berlin (1999)
7. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. IEEE Trans. Softw. Eng. **30**(5), 295–310 (2004)
8. Basseur, M., Zitzler, E.: A preliminary study on handling uncertainty in indicator-based multiobjective optimization. In: Appl. of Evol. Computing, pp. 727–739. Springer, Berlin (2006)
9. Becker, S., Grunske, L., Mirandola, R., Overhage, S.: Performance prediction of component-based systems—a survey from an engineering perspective. In: Architecting Systems with Trustworthy Components. LNCS, vol. 3938, pp. 169–192. Springer, Berlin (2006)
10. Beyer, H., Sendhoff, B.: Robust optimization: a comprehensive survey. Comput. Methods Appl. Mech. Eng. **196**(33–34), 3190–3218 (2007)
11. Bhunia, A., Sahoo, L., Roy, D.: Reliability stochastic optimization for a series system with interval component reliability via genetic algorithm. Appl. Math. Comput. **216**(3), 929–939 (2010)
12. Birolini, A.: Reliability Engineering: Theory and Practice. Springer, Berlin (2010)
13. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queuing Network and Markov Chains. Wiley, New York (1998)
14. Cheung, R.: A user-oriented software reliability model. IEEE Trans. Softw. Eng. **6**(2), 118–125 (1980)
15. Ciesinski, F., Größer, M.: On probabilistic computation tree logic. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems: A Guide to Current Research. LNCS, vol. 2925, pp. 147–188. Springer, Berlin (2004)
16. Coit, D., Jin, T., Wattanapongsakorn, N.: System optimization with component reliability estimation uncertainty: a multi-criteria approach. IEEE Trans. Reliab. **53**(3), 369–380 (2004)
17. Coit, D.W., Smith, A.E.: Genetic algorithm to maximize a lower-bound for system time-to-failure with uncertain component Weibull parameters. Comput. Ind. Eng. **41** (2002)
18. Cortellessa, V., Grassi, V.: A modeling approach to analyze the impact of error propagation on reliability of component-based systems. In: Component-Based Software Engineering, pp. 140–156. Springer, Berlin (2007)
19. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21–28, 2011, pp. 341–350. ACM, New York (2011)
20. Fiondella, L., Gokhale, S.S.: Software reliability with architectural uncertainties. In: Parallel and Distributed Processing, pp. 1–5. IEEE (2008)
21. Förster, M., Trapp, M.: Fault tree analysis of software-controlled component systems based on second-order probabilities. In: International Symposium on Software Reliability Engineering, pp. 146–154. IEEE, Nov 2009
22. Frolund, S., Koistinen, J.: Quality-of-service specification in distributed object systems. Distrib. Syst. Eng. J. **5**(4), 179–202 (1998)
23. Goeva-Popstojanova, K., Trivedi, K.: Architecture-based approach to reliability assessment of software systems. Perform. Eval. **45**(2–3), 179–204 (2001)
24. Goseva-Popstojanova, K., Hamill, M.: Architecture-based software reliability: why only a few parameters matter?. In: Computer Software and Applications Conference, 2007, vol. 1, pp. 423–430. IEEE (2007)
25. Goseva-Popstojanova, K., Hamill, M., Perugupalli, R.: Large empirical case study of architecture-based software reliability. In: International Symposium on Software Reliability Engineering, vol. 54, pp. 10–52. IEEE (2005)
26. Goseva-Popstojanova, K., Hamill, M., Wang, X.: Adequacy, accuracy, scalability, and uncertainty of architecture-based software reliability: lessons learned from large empirical case studies. In: International Symposium on Software Reliability Engineering, pp. 197–203. IEEE (2006)
27. Goseva-Popstojanova, K., Kamavaram, S.: Assessing uncertainty in reliability of component-based software systems. In: ISSRE 2003, pp. 307–320. IEEE (2003)
28. Goseva-Popstojanova, K., Kamavaram, S.: Software reliability estimation under uncertainty:generalization of the method of moments. High Assur. Syst. Eng. **2004**, 209–218 (2004)

29. Grunske, L.: Specification patterns for probabilistic quality properties. In: Proceedings of the 13th International Conference on Software Engineering, ICSE '08 61(0), 31 (2008)
30. Grunske, L., Han, J.: A comparative study into architecture-based safety evaluation methodologies using AADL's error annex and failure propagation models. High Assurance Systems Engineering, Symposium, pp. 283–292 (2008)
31. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994)
32. Jhumka, A., Hiller, M., Suri, N.: Component-based synthesis of dependable embedded software. Lect. Notes Comput. Sci. **2469**, 111–128 (2002)
33. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. **2**(4), 255–299 (1990)
34. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: a hybrid approach. Int. J. Softw. Tools Technol. Transfer **6**(2), 128–142 (2004)
35. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking in practice: case studies with PRISM. ACM SIGMETRICS Perform. Eval. Rev. **32**(4), 16–21 (2005)
36. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods Syst. Des. **29**(1), 33–78 (2006)
37. Limbourg, P.: Multi-objective optimization of generalized reliability design problems using feature models: a concept for early design stages. Reliab. Eng. Syst. Saf. **93**(6), 815–828 (2008)
38. Marseguerra, M., Zio, E., Podofillini, L., Coit, D.: Optimal design of reliable network systems in presence of uncertainty. IEEE Trans. Reliab. **54**(2), 243–253 (2005)
39. Meedeniya, I., Buhnova, B., Aleti, A., Grunske, L.: Architecture-driven reliability and energy optimization for complex embedded systems. In: Quality of Software Architectures, QoSA 2010. LNCS, vol. 6093, pp. 52–67. Springer, Berlin (2010)
40. Meedeniya, I., Bühnova, B., Aleti, A., Grunske, L.: Reliability-driven deployment optimization for embedded systems. J. Syst. Softw. **84**(5), 835–846 (2011)
41. Meedeniya, I., Moser, I., Aleti, A., Grunske, L.: Architecture-based reliability evaluation under uncertainty. In: 7th International Conference on the Quality of Software Architectures, QoSA 2011 and 2nd International Symposium on Architecting Critical Systems, ISARCS 2011, pp. 85–94 (2011)
42. Montgomery, D., Runger, G.: Applied Statistics and Probability for Engineers. Wiley, India (2007)
43. Oberkampf, W., Helton, J., Joslyn, C., Wojtkiewicz, S., Ferson, S.: Challenge problems: uncertainty in system response given uncertain parameters. Reliab. Eng. Syst. Saf. **85**(1–3), 11–19 (2004)
44. Puterman, M.L.: Markov Decision Processes. Wiley, New York (1994)
45. Roshandel, R., Banerjee, S., Cheung, L., Medvidovic, N., Golubchik, L.: Estimating software component reliability by leveraging architectural models. In: International Conference on Software Engineering, p. 853. ACM, New York (2006)
46. Roshandel, R., Medvidovic, N., Golubchik, L.: A Bayesian model for predicting reliability of software systems at the architectural level. LNCS **4880**, 108–126 (2007)
47. Rubinstein, R., Kroese, D.: Simulation and the Monte Carlo Method. Wiley-Interscience, New York (2008)
48. Sanchez, A., Carlos, S., Martorell, S., Villanueva, J.: Addressing imperfect maintenance modelling uncertainty in unavailability and cost based optimization. Reliab. Eng. Syst. Saf. **94**(1), 22–32 (2009)
49. Trivedi, K.: Probability & Statistics with Reliability. Queuing and Computer Science Applications. Wiley, India (2009)
50. Wang, W., Wu, Y., Chen, M.: An architecture-based software reliability model. In: Proceedings of 1999 Pacific Rim International Symposium on Dependable Computing, pp. 143–150. IEEE (2002)
51. Wattanapongskorn, N., Coit, D.W.: Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty. Rel. Eng. **92**, 395–407 (2007)
52. Yin, L., Smith, M., Trivedi, K.: Uncertainty analysis in reliability modeling. In: Symposium on Reliability and Maintainability, pp. 229–234 (2001)

Author Biographies

Indika Meedeniya is a research fellow at Swinburne University of Technology, Melbourne, Australia, where he received his Ph.D. in 2012. He graduated with a BSc in Electronic and Telecommunication Engineering from the University of Moratuwa, Sri Lanka in 2005, and he worked as a Tech-Lead in developing performance- and reliability-critical software at Millennium IT until 2008. His research interests include modeling and architecture-based evaluation of probabilistic quality attributes, software architecture optimisation and methods to deal with uncertainty in design-time estimates.

Irene Moser is a Lecturer in the Faculty of ICT at Swinburne University of Technology, where she received her Ph.D. Her research interests include combinatorial, multi- and manyobjective optimisation using stochastic (evolutionary) and deterministic methods. Her specialisations encompass the use of statistical methods for optimisation and search space diagnostics as well as the modelling of practical problems.

Aldeida Aleti is a lecturer at Monash University. She received her Ph.D. degree from Swinburne University of Technology, Melbourne, in 2012. She studied BEng (Hons) in Computer Engineering at Yildiz University in Istanbul, 2005, and MEng in Computer Engineering at Polytechnic University of Tirana, 2008. Since 2008, she has held a Ph.D. scholarship from the Cooperative Research Centre for Advanced Automotive Technology in Australia. Her research interests include optimization of software architectures at design time.

Lars Grunske is Jun. Professor at the University of Kaiserslautern, Germany. He received his Ph.D. degree in computer science from the University of Potsdam, Germany, (Hasso-Plattner-Institute for Software Systems Engineering), in 2004. He was Boeing Postdoctoral Research Fellow at the University of Queensland, Australia, and Lecturer at the Swinburne University of Technology, Australia. He has active research interests in the areas of modeling and verification of systems and software. His main focus is on architecture optimization and model-based dependability evaluation of complex software intensive systems.