Evaluation of resilience in self-adaptive systems using probabilistic model-checking

ARTICLE · JUNE 2012

DOI: 10.1109/SEAMS.2012.6224391

2 AUTHORS, INCLUDING:

Javier Cámara
Carnegie Mellon University
53 PUBLICATIONS **353** CITATIONS

Evaluation of Resilience in Self-Adaptive Systems Using Probabilistic Model-Checking

Javier Cámara
University of Coimbra
Portugal
jcmoreno@dei.uc.pt

Rogério de Lemos
University of Kent
United Kingdom
r.delemos@kent.ac.uk

***Abstract*—The provision of assurances for self-adaptive systems presents its challenges since uncertainties associated with its operating environment often hamper the provision of absolute guarantees that system properties can be satisfied. In this paper, we define an approach for the verification of self-adaptive systems that relies on stimulation and probabilistic model-checking to provide levels of confidence regarding service delivery. In particular, we focus on resilience properties that enable us to assess whether the system is able to maintain trustworthy service delivery in spite of changes in its environment. The feasibility of our proposed approach for the provision of assurances is evaluated in the context of the Znn.com case study.**

***Keywords*-resilience; assurances; evaluation; probabilistic model checking; stimulation; self-adaptation**

I. Introduction

Despite recent advances in self-adaptive systems, a key aspect that still remains a challenge is the provision of assurances, that is, the collection, structuring and combination of evidence that a system satisfies a set of stated functional and non-functional properties during its operation. The main reason for this is the high degree of uncertainty associated with changes that may occur to the system itself, its environment or its goals [5]. In particular, since the behavior of the environment cannot be predicted nor controlled by the system (*e.g.*, load, network conditions, etc.), this prevents obtaining absolute guarantees that system properties can be satisfied. Moreover, unlike in consolidated model-based verification and validation techniques, models in self-adaptive systems cannot be assumed to be fixed since changes that might affect the system, its environment or its goals might also affect the models.

This intrinsic uncertainty associated with self-adaptive systems has established the need to devise new approaches for assessing whether a set of stated properties are satisfied by the system during operation while changes occur. A major requirement for these approaches is that they need to be able to provide levels of confidence, instead of establishing absolute guarantees about property satisfaction. But for that, there is the need to determine what are the limits of the system when facing changes while still providing a service that can justifiably be trusted (*i.e.*, the avoidance of failures that are unacceptably frequent or severe) [17].

In this paper, we propose a new approach, based on stimulation and probabilistic model-checking, for the verification of self-adaptive systems, whose environment exhibits stochastic behavior [12]. In a nutshell, the underlying idea is to stimulate the environment of the system in order to exercise its adaptive capabilities, and to collect data about how the system reacts to those environmental changes. The collected data is aggregated into a probabilistic model of the system's behavior that will be used as input into a model checker for evaluating whether system properties are satisfied within certain confidence levels. The proposed approach consists of four basic steps (see Figure 1):

1. Stimulus generation. Determine how the system's environment should be stimulated to collect information relevant for the properties of interest. Stimulation is based on system adaptation alternatives, domain knowledge, *levers* (*i.e.*, effectors on the environment), and the properties to be verified;
2. Experimental data collection. Stimulate the system's environment through *levers* to trigger adaptation mechanisms and collect information about system behavior undergoing adaptation as a set of execution traces;
3. Model generation. Aggregate execution traces into a probabilistic model of the system's behavior;
4. Property verification. Check system properties against the obtained probabilistic models.

Although the concept of updating probabilistic model parameters with run-time system information has been explored [9], our approach enables the construction from scratch of a model of the system's response to changes in its environment. The contribution of this paper is twofold:

- A method to obtain levels of confidence regarding the satisfaction of a set of stated non-functional (resilience) properties within a particular time bound;
- An analysis of how different parameters (*e.g.*, resolution) in model construction impact the accuracy of probability estimations.

The rest of this paper is organized as follows. Section II provides an overview of the case study used to illustrate

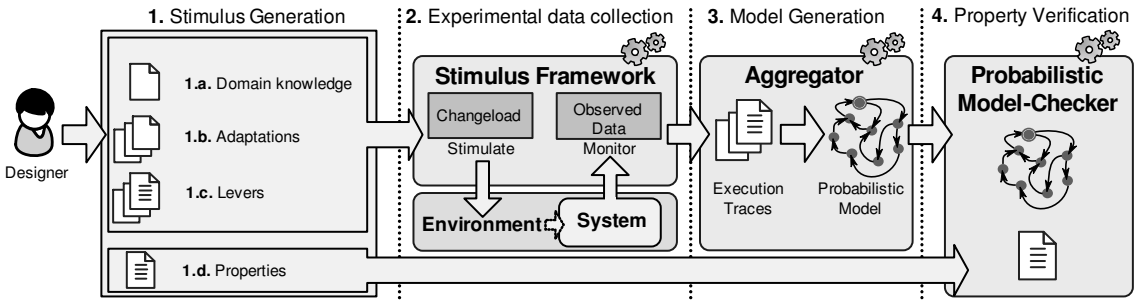

Figure 1. Overview of the approach

the proposed approach. Section III introduces the formal model, as well as language for expressing system properties. Section IV details the steps comprised by our approach. Section V describes the validation and experimental results. Section VI describes related work, comparing it with the proposed approach. Finally, Section VII concludes the paper and indicates future research directions.

II. Case Study

To illustrate our approach, we use the Znn.com case study [7], which is able to reproduce the typical infrastructure for a news website. It has a three-tier architecture consisting of a set of servers that provide contents from backend databases to clients via frontend presentation logic. Architecturally, it is a web-based client-server system that satisfies an N-tier style, as illustrated in Figure 2. The system uses a load balancer to balance requests across a pool of replicated servers, the size of which can be adjusted to balance server utilization against service response time. A set of client processes makes stateless requests, and the servers deliver the requested contents (*i.e.,* text, images and videos).

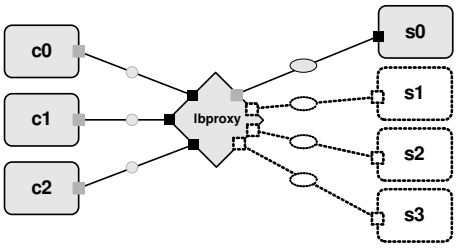

Figure 2. Znn.com system architecture

The main objective for Znn.com is to provide content to customers within a reasonable response time, while keeping the cost of the server pool within a certain operating budget. It is considered that from time to time, due to highly popular events, Znn.com experiences spikes in requests that it cannot serve adequately, even at maximum pool size. To prevent losing customers, the system can provide minimal textual contents during such peak times instead of not providing service to a part of the customers. In particular, we identify two main quality objectives for the self-adaptation of the system: (i) performance, which depends on request response time, server load, and available network bandwidth; and (ii) cost, which is associated to the number of active servers.

Self-adaptive features in Znn.com are implemented using Rainbow [6], [10], a platform for architecture-based self-adaptation. Rainbow is capable of analyzing trade-offs among the different objectives, and execute different adaptations according to the particular run-time conditions of the system. For instance, in Znn.com, when response time becomes too high, the system should increment server pool size if it is within budget to improve its performance; otherwise, servers should be switched to textual mode (start serving minimal text content) if cost is near budget limit.

III. System Model

In this section, we present Discrete-Time Markov Chains (DTMCs) [15], the probabilistic model used as the basis to represent the behavior of the system. Moreover, we introduce the kind of properties that we check in the system as well as Probabilistic Computation Tree Logic (PCTL) [2], the language used for expressing them. This section is concluded by describing how the aforementioned elements are combined in order to obtain system models.

A. Discrete-Time Markov Chains

In order to model the behaviour of our system, we employ DTMCs, defined as state-transition systems augmented with probabilities. States represent possible configurations of the system, and transitions occur at discrete time and have an associated probability. DTMCs are discrete stochastic processes where the probability distribution of future states depend only upon the current state. Let AP be a fixed, finite set of atomic propositions used to label states with properties of interest.

Definition 1 (DTMC): A (labelled) DTMC is a tuple (S, S_0, P, L) where:

- S is a finite set of states;
- $S_0 \subseteq S$ is a set of initial states;

- $P : S \times S \rightarrow [0, 1]$ is a transition probability matrix where $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$;
- $L : S \rightarrow 2^{AP}$ is a labelling function that assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are true in the state.

Each element $P(s, s')$ in the transition probability matrix represents the probability that the next state of the process will be s', given that the current state is s. Terminating states, *i.e.*, those from which the system cannot move to another state, are modeled by adding a self-loop (a single transition going back to the same state with probability one).

B. Resilience Properties

In order to express resilience properties, we use PCTL [2], which is a logic language inspired by CTL [2]. Instead of the universal and existential quantification of CTL, PCTL provides the probabilistic operator $\mathcal{P}_{\bowtie p}(.)$, where $p \in [0, 1]$ is a probability bound and $\bowtie \in \{\leq, <, \geq, >\}$. PCTL is defined by the following syntax:

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

Formulae Φ are named state formulae, and can be evaluated over a Boolean domain ($true$, $false$) in each state. Formulae ψ are named path formulae, and describe a pattern over the set of all possible paths originating in the state where they are evaluated. In state formulae, other Boolean operators, such as disjunction ($\vee$), implication ($\Rightarrow$), etc. can be specified based on the primary Boolean operators. Moreover, we employ the abbreviations: (bounded) finally ($F^{(\leq t)}\Phi = true\ U^{(\leq t)}\Phi$) and globally ($G\Phi = \neg F \neg \Phi$).

The satisfaction relation for a state s in PCTL is:

$s \models true$
$s \models a$ iff $a \in L(s)$
$s \models \neg\Phi$ iff $s \not\models \Phi$
$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ *and* $s \models \Phi_2$
$s \models \mathcal{P}_{\bowtie p}(\psi)$ iff $Pr(s \models \psi) \bowtie p$

A formal definition of how to compute $Pr(s \models \psi)$ is presented in [2]. The intuition is that, its value corresponds to the fraction of paths originating in s and satisfying ψ over the entire set of paths originating in s. The satisfaction relation of a path formula with respect to a path π that originates in state s (*i.e.*, $\pi[0] = s$) is:

$\pi \models X\Phi$ iff $\pi[1] \models \Phi$
$\pi \models \Phi U \Psi$ iff $\exists j \geq 0.(\pi[j] \models \Psi \wedge (\forall 0 \leq k < j.\pi[k] \models \Phi))$
$\pi \models \Phi U^{\leq t} \Psi$ iff $\exists 0 \leq j \leq t.(\pi[j] \models \Psi \wedge (\forall 0 \leq k < j.\pi[k] \models \Phi))$

With the aid of PCTL, a designer can express properties about the system that are typically domain-dependent. Furthermore, in order to ease the formulation of probabilistic properties, we make use of property specification patterns [8] that describe generalized recurring properties in probabilistic temporal logics. In our case, we are interested in how the system responds to changes in the environment, therefore we restrict ourselves to properties that can be instanced by using probabilistic response patterns [11], adapted to PCTL syntax (see Table I). These patterns include a premise Φ_1 that represents in our case a change in the operating environment of the system, and a subformula enclosed by the probabilisic operator $\mathcal{P}_{\bowtie p}(.)$ that represents the response to that change that we are expecting from the system (with a probability bound p and a time bound t).

Table I
PROBABILISTIC RESPONSE SPECIFICATION PATTERNS

PCTL Formulation	Description
$\mathcal{P}_{\geq 1}[G(\Phi_1 \Rightarrow \mathcal{P}_{\bowtie p}(F^{\leq t}\Phi_2))]$	**Probabilistic Response.** After state formula Φ_1 holds, state formula Φ_2 must become *true* within time bound t, with a probability bound $\bowtie p$.
$\mathcal{P}_{\geq 1}[G(\Phi_1 \Rightarrow \mathcal{P}_{\bowtie p}(\neg\Phi_2 U^{\leq t}\Phi_3))]$	**Probabilistic Constrained Response.** After state formula Φ_1 holds, state formula Φ_3 must become *true*, without Φ_2 ever holding, within time bound t, with a probability bound $\bowtie p$.

Example 1: In Znn.com, we are interested in assessing how the system reacts to request response time going above a particular threshold. Let `expRspTime` be the variable associated with experienced request response time, and `totalCost` be the variable associated with operating cost. We can define the following predicates:

$$\texttt{cViolation} = \texttt{expRspTime} > \texttt{MAX\_RSPTIME}$$
$$\texttt{hiCost} = \texttt{totalCost} \geq \texttt{THRESHOLD\_COST}$$

Where `MAX_RSPTIME` is a threshold that establishes the maximum acceptable response time, and `THRESHOLD_COST` determines the maximum operating budget expected for the system. Based on these predicates, we may instantiate the following PCTL property, making use of the probabilistic constrained response pattern included in Table I:

$$\mathcal{P}_{\geq 1}[G(\texttt{cViolation} \Rightarrow \mathcal{P}_{\geq 0.85}(\neg\ \texttt{hiCost}\ U^{\leq 120} \neg\ \texttt{cViolation}))]$$

This property reads as: "When response time goes above threshold `MAX_RSPTIME` , the probability of lowering response time below `MAX_RSPTIME` without exceeding the expected budget `THRESHOLD_COST` in 120 seconds is greater than 0.85".

C. Operational Profiles and Adaptations

Within a self-adaptive system we may distinguish between a *conventional operational profile* in which the system is operating without experiencing any anomalies, and *non-conventional operational profiles* associated with changes in the environment that induce anomalies in the system (typically triggering *adaptations*).

Definition 2 (Conventional Operational Profile): The conventional operational profile C of a system is the region of the state space where no anomalies hold:

$$C = \{s \in S \mid \forall \alpha \in A,\ s \not\models \alpha\}$$

where A is the set of possible anomalies that the system can experience expressed as PCTL state formulae.

Definition 3 (Non-conventional Operational Profile): A non-conventional operational profile N associated with an anomaly $\alpha \in A$ is the region of the state space where the anomaly holds:

$$N = \{s \in S \mid s \models \alpha\}$$

where α is a PCTL state formula built over the set of atomic propositions AP.

An adaptation is triggered as a response to a system anomaly caused by a change in the environment of the system. When the system experiences an anomaly, it goes from its conventional operational profile into a non-conventional operational profile. As a general rule, the intended purpose of the adaptation is steering the system back to its conventional operational profile.

Definition 4 (Adaptation): An adaptation is a tuple (c, g, d) where:

- c is the applicability condition (*e.g.,* system anomaly) expressed as a PCTL state formula built over the set of atomic propositions AP;
- g is an adaptation goal expressed as a PCTL formula built over the set of atomic propositions AP;
- $d \in \mathbb{R}^+$ is a deadline for the satisfaction of the goal of the adaptation in the system.

It is worth observing that we define adaptations in terms of applicability conditions and intended effects on the system, abstracting away from the specific actions that it carries out. Moreover, the system may comprise several alternative adaptations as a response to the same anomaly, and choose among them at run-time according to the particular system state. In any case, an adaptation (c, g, d) is always related to a non-conventional operational profile N through its applicability condition, that is, if we let α be the anomaly associated with N, then $c = \alpha$.

Example 2: Let us consider the predicates introduced in Example 1 as the set of possible anomalies in the system A = { `cViolation` , `hiCost` }. We may then define the conventional operational profile of the system as:

$$C = \{s \in S \mid s \models \neg(\ \texttt{cViolation} \vee \texttt{hiCost}\ )\}$$

That is, the set of states in which the system is operating on budget and according to an acceptable request response time. Furthermore, associated with each anomaly, we identify a non-conventional operational profile:

$$N_{cViolation} = \{s \in S \mid s \models \texttt{cViolation}\}$$
$$N_{hiCost} = \{s \in S \mid s \models \texttt{hiCost}\}$$

Rainbow uses a language called Stitch [6] to represent high-level adaptation concepts, embodied in adaptation strategies. Let us consider now the code presented at the end of this section of a sample strategy intended to reduce response time in Znn.com. In line 2, we can observe the definition of `cViolation` , which is also the applicability condition for the strategy `ReduceResponseTime` (line 5, between brackets). In the case of Stitch, adaptation goals and deadlines are not explicitly specified, therefore in this case, we can complement the specification of the adaptation using as adaptation goal the property described in Example 1:

`ReduceResponseTime` = (`cViolation` ,

$$\mathcal{P}_{\geq 0.85}(\neg\ \texttt{hiCost}\ U^{\leq 120} \neg\ \texttt{cViolation}\ ),\ 120\ )$$

Let us remark that although the deadline in this case coincides with the time bound of the adaptation goal, this may not always be the case (e.g., when the adaptation goal is a conjunction of several PCTL formulae with different time bounds).

```
1
2  define boolean cViolation=exists c:T.ClientT in
3                M.components|c.expRspTime > M.
                       MAX_RSPTIME;
4
5  strategy ReduceResponseTime [cViolation]{
6    define boolean hiLatency =
7       exists k :T.HttpConnT in M.connectors | k.latency
              > M.MAX_LATENCY;
8    define boolean hiLoad =
9      exists s :T.ServerT in M.components | s.load > M.
            MAX_UTIL;
10
11   t1:(#[Pr{t1}] hiLatency)->switchToTextualMode()@
          [1000/*ms*/]{
12     t1a: (success)->done;
13   }
14   t2:(#[Pr{t2}] hiLoad)->enlistServer(1)@[2000/*ms*/]{
15     t2a:(!hiLoad)->done;
16     t2b:(!success)->do[1]t1;
17   }
18   t3:(default)->fail;
19 }
```

D. Modeling Non-conventional Operational Profiles

The high degree of variability in the operating environment of self-adaptive systems hampers the construction of system accurate models. However, since our interest is to assess system ability to provide trustworthy service when environmental changes occur, the analysis can be restricted to non-conventional operational profiles of the system.

For our approach, we assume that a system model consists of a set of non-conventional operational profiles models N_i, $i \in \{1 \ldots p\}$, each of which is associated with a set of adaptations $Ad_i = \{(c_1, g_1, d_1), \ldots, (c_q, g_q, d_q)\}$, $j \in \{1, \ldots, q\}$ through their applicability condition (let α_i be the anomaly associated to N_i, then $\forall c_j,\ c_j = \alpha_i$).

In the following, we formally define what is a non-conventional operational profile model, but before that we formulate some basic concepts associated with that definition. In those definitions, for the sake of clarity, we abstract from the probabilities in DTMCs. Therefore, we assume that there is a transition between states s and s' in a DTMC (denoted as $s \longrightarrow s'$) iff $P(s, s') \neq 0$.

Definition 5 (Boundary): The *boundary* of a non-conventional operational profile N is defined as:

$$boundary(N) = \{n \in N \mid \exists s \longrightarrow n :\ s \in S \setminus N\}.$$

Definition 6 (Trajectory): A *trajectory* is a finite sequence of transitions $\sigma = s_1 \longrightarrow \ldots \longrightarrow s_n$.

We assume that for all transitions $s \longrightarrow s'$, the time needed by the system to move from state s to state s' is given by $\tau \in \mathbb{R}^+$. Then, for any trajectory of the system σ, we define its *duration* as $\Delta(\sigma) = n\tau$.

A state $s' \in S$ is *reachable* from a state s in time t (denoted as $s \xrightarrow{t} s'$) if there exists a trajectory $\sigma = s \longrightarrow \ldots \longrightarrow s'$, such that $\Delta(\sigma) \leq t$.

Definition 7 (Non-conventional Operational Profile Model): A model for a non-conventional operational profile N is a DTMC such that:

- Set of initial states $S_0^N = boundary(N)$;
- Set of states $S^N = \{s \in S \mid \exists s_0 \in S_0^N : s_0 \xrightarrow{max(d_j)} s\}$;

where $Ad = \{(c_1, g_1, d_1), \ldots, (c_q, g_q, d_q)\}, \quad j \in \{1, \ldots, q\}$ is the set of adaptations associated to N. Hence, initial states correspond to those where the system enters the non-conventional operational profile, and the state space of the model includes all the states reachable before the maximum of the deadlines for adaptations expires.

IV. Approach

In this section, we present an overview of the different steps comprised by our approach (Figure 1): (i) stimulus generation, where we determine how the environment must be stimulated to trigger system adaptation mechanisms; (ii) experimental data collection, where the system's environment is stimulated for collecting execution traces of system undergoing adaptation; (iii) model generation, where DTMC based models are obtained from the aggregated execution traces; and (iv) property verification, where system properties are verified against the obtained probabilistic model.

A. Stimulus Generation

To obtain a representative operational model of the system, we need to subject its environment to equally representative changes. A *changeload* is a collection of scenarios comprising representative changes of the environment that stimulate the system's adaptive capabilities, enabling us to observe its response. Obtaining a representative changeload is a challenging task that typically involves domain knowledge, either coming from experts in the application domain, or from real data available from similar existing systems. However, accessing such data is not possible in many situations, since the type of experienced changes or their probability of occurrence are not usually recorded in already deployed systems. Moreover, the wide range and complexity of changes that a self-adaptive system can experience [1] makes even more difficult to automatically generate a representative changeload. However, in the case of the evaluation of the kind of resilience properties described in Section III-B, the test designer can be aided by a partial automation of the process, as we discuss further in this section. Concretely, we provide a formal model of scenarios used in our approach to stimulate the system environment, followed by a description of a method to help identify a changeload.

1) Scenario Model: A scenario defines a postulated sequence of events, which should be able to capture, during a given time frame, changes in the environment, in the system, or at the interface between them (*i.e.*, system goals). However, in this work we focus on self-adaptive systems where system goals do not change at run-time, therefore these are not considered in our definition of scenario. In particular, scenarios consist of a set of traces that correspond to a set of monitored variables associated with the system or its environment. Each trace consists of a sequence of variable values collected at discrete time points according to a rate τ that coincides with the transition time in our model (introduced in Section III-D). Moreover, collected values are discretized according to a particular resolution in order to bound the growth of the state-space of our models.

Definition 8 (Trace): A trace of a variable is a tuple $(\eta, [\alpha, \beta], V)$ where:

- $\eta \in \mathbb{R}^+$ is a quantization parameter that determines the resolution of the monitored variable;
- $[\alpha, \beta]$, $\alpha, \beta \in \mathbb{R}$ is the range of values associated with the variable;
- $V = \langle v_1, \ldots, v_n \rangle$ is a vector, where $v_{i \in \{1,\ldots,n\}} \in [\mathbb{R}]_\eta$ corresponds to the value of the monitored variable at time instant i.

The values of a variable at different time instants are based on a quantization of the state-space $\mathbb{R}$, approximated by the set:

$$[\mathbb{R}]_\eta = \{r \in \mathbb{R} \mid r = k\eta,\ k \in \mathbb{Z},\ \alpha \leq r \leq \beta\}.$$

It is worth observing that v_i is the quantized value of the monitored variable. Let $v_{\mathbb{R}i}$ be the actual value of the variable observed at instant i. The quantized value v_i is obtained as:

$$quant(v_{\mathbb{R}i}) = \arg\min_{r \in [\mathbb{R}]_\eta}(|v_{\mathbb{R}i} - r|).$$

Definition 9 (Scenario): A scenario defined over a set of traces is a tuple (δ, τ, T):

- $\delta \in \mathbb{R}^+$ determines the duration of the time frame associated with the scenario $[0, \delta]$;
- $\tau \in \mathbb{R}^+$ is a time sampling parameter that indicates the rate at which the variables included in the scenario are sampled. Let us remark that the number of samples included in the scenario is $n = \delta/\tau$. We assume that δ is always a multiple of τ;
- $T = \langle tr_1, \ldots, tr_m \rangle$ is a vector of traces $tr_{i \in \{1,\ldots,m\}} \in T_{env} \cup T_{sys}$, where $m \in \mathbb{N}$ corresponds to the number of variables included in the scenario, and T_{env}, T_{sys} correspond respectively to the sets of traces of environment and system variables.

Moreover, in order to be able to determine the overall state of the system and its environment at a given time point, we introduce the concept of *snapshot*.

Definition 10 (Snapshot): A snapshot is a vector $\langle v_1, \ldots, v_m \rangle$, $v_{j \in \{1,\ldots,m\}} \in [\mathbb{R}]_{\eta j}$ that contains the values of a set of variables in a particular time instant.

We refer to a snapshot as *environment snapshot* or *system snapshot* if it includes only variables from T_{env} or T_{sys}, respectively.

Definition 11 (Snapshot Trace): A snapshot trace is a vector $\langle s_1, \ldots, s_n \rangle$ *s.t.* $s_{i \in \{1,\ldots,n\}}$ is a snapshot for time $i\tau$.

Therefore, we can characterize a run of the system as an ordered collection of snapshots that captures state changes from system initialization to termination.

Example 3: Figure 3 outlines a sample scenario of Znn.com that comprises two clients `c1` and `c2` , a server `s` , a load balancer, and the connectors among them `conn1 – 3` . The duration of the scenario is one hour, and the state of the system and the environment is sampled periodically every 10 seconds. Traces of environment variables T_{env} include connector latency and server load, whereas traces for system variables T_{sys} comprise experienced response time for each of the clients and operation cost. Quantization parameters used are $\eta_{latency} = 50$, $\eta_{load} = 5$, $\eta_{expRspTime} = 10$, and $\eta_{totalCost} = 1$. Let us remark that all variable values are multiples of their respective quantization parameters.

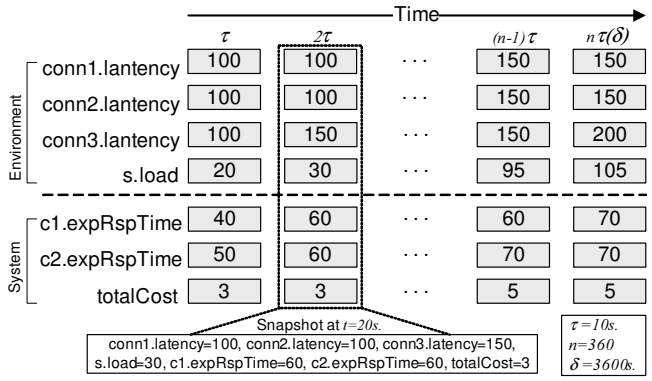

Figure 3. Znn.com sample scenario

2) Identifying a Changeload: The first step to generating a representative changeload for the system is determining how changes in its environment can steer the system towards non-conventional operational profiles. To achieve this goal, we assume that the stimulus framework provides a number of effectors or *levers* that can be used to modify the values of environment variables (*i.e.*, acting as a closed-loop controller, similarly to frameworks such as Rainbow, that use effectors through *operators* to steer the system according to a set of objectives [10]).

Secondly, we need to define a metric to estimate the distance between any state of the system to the objective, that is, a particular non-conventional operational profile. For this purpose, we define the distance from a snapshot s to a non-conventional operational profile N as:

$$Dst_N(s, N) = \min_{s_N \in N} dst(s, s_N); \quad \text{with } dst(s, s') = \max_{j=1}^{m} \frac{|v_j - v'_j|}{\eta_j};$$

where function dst corresponds to the distance between any two snapshots $s = \langle v_1, \ldots, v_m \rangle$ and $s' = \langle v'_1, \ldots, v'_m \rangle$.

Algorithm 1 Environment Stimulation

Input: Initial state of the system s_0, non-conventional operational profile N, anomaly α associated with N, observation time parameter O_τ, settling threshold O_λ, settling period k.
Output: Environment snapshot trace t to steer the system towards N.

```
1:  t := ⟨⟩; current := s0; d := Dst_N(current, N);
2:  while current ⊭ α ∧ ¬cond do
3:      ng := generate_neighbor(Env(current));
4:      apply(ng);
5:      wait(O_τ);
6:      next := observe;
7:      if Dst_N(next, N) < d then
8:          append(t, next);
9:          d := Dst_N(next, N);
10:         current := next;
11:     else
12:         apply(Env(current));
13:         clock := 0;
14:         while clock ≤ k ∧ ¬cond do
15:             wait(τ);
16:             next := observe;
17:             if dst(next, current) < O_λ then
18:                 clock := clock + 1;
19:             else
20:                 clock := 0;
21:             end if
22:         end while
23:     end if
24: end while
25: if cond then
26:     return ⟨⟩;
27: else
28:     return t;
29: end if
```

Using this metric, Algorithm 1 stimulates the environment for steering the system towards a non-conventional operational profile N, starting from an initial condition s_0. It is important to emphasize that this algorithm is intended as an aid for the stimulus designer, rather than as the only way to determine the changeload, since the controllability of the system from the initial condition s_0 ($\exists t \in \mathbb{R}^+ : s_0 \xrightarrow{t} s_N, s_N \in N$) cannot be guaranteed due to the uncertainties regarding how environmental changes affect the system. Each iteration of the algorithm determines a neighbor *ng* (line 3) of the *current* system state by modifying the value of one or more environment variables. Let us remark that the concrete criteria for the generation of a neighbor state and its application on the environment is abstracted away by two functions: *generate_neighbor* (line 3), which has to be customized for every case by using domain knowledge (*e.g.*, in Znn.com, an increase in the latency of a network link will probably increase response time, thus reducing the distance to $N_{cViolation}$); and *apply* (line 4), which maps changes in environment variables to levers in the environment (*e.g.*, induction of an artificial delay in a network link to increase its associated latency). Function

Env (line 3) removes system variables from a snapshot, returning an environment snapshot only with variables from T_{env}. Function *observe* (line 6) returns a snapshot with the current state of the system. In particular, the algorithm waits to observe the effect of a change by using an explicit timing delay O_τ (line 5) to capture the settling time of the system after a change is applied. If the distance to N has been reduced *w.r.t.* the previous state, the algorithm goes to its next iteration (line 2). Otherwise, the algorithm ensures that the system has settled, controlling that the output is bounded and does not exceed a particular threshold O_λ (line 17) for a particular period of time (tracked by the *clock* variable, line 14). If system does not settle by a given deadline or a maximum number of iterations (stop condition abstracted by *cond*, line 14), the algorithm returns an empty trace, meaning that a sequence of changes in the environment for steering the system towards N has not been found. Function *append* (line 8) adds a new element at the end of a vector $V = \langle e_1, e_2, \ldots, e_n \rangle$: $append(V, e) = \langle e_1, \ldots, e_n, e \rangle$.

B. Experimental Data Collection

Building a representative model of the system also implies collecting data in a meaningful way. In particular, data related to the operation of the system in its conventional operational profile is not useful for evaluating resilience properties, so it must be discarded. Concretely, we collect information corresponding time intervals that start when an anomaly occurs, and end when the deadline to fulfill adaptation goals expires.

Algorithm 2 Data Collection

Input: Snapshot trace ST, set of anomalies A, vector of maximum deadlines for adaptations $MD = \langle md_1, \ldots md_p \rangle : md_i = max(d_j),\ i \in \{1, \ldots, n\},\ j \in \{1, \ldots, p\}$.
Output: Vector of sets of traces T^N

```
1: T^N := ⟨T_0, ..., T_n⟩;  ∀i ∈ {1,...,n} T^N[i] := ∅;
2: tr := ⟨tr_1, ..., tr_n⟩; ∀i ∈ {1,...,n} tr[i] := ⟨⟩;
3: while ST ≠ ⟨⟩ do
4:     current := extract(ST);
5:     for α_c ∈ A : (current ⊨ α_c) ∧ (clocks[c] = 0) do
6:         clocks[c] := MD[c];
7:         tr[c] := ⟨⟩;
8:     end for
9:     for i ∈ {1,...,n} : clocks[i] ≠ 0 do
10:        append(tr[i], current);
11:        clocks[i] := clocks[i] − 1;
12:        if clocks[i] = 0 then
13:            T^N := T^N[i] ∪ {tr[i]};
14:        end if
15:    end for
16: end while
17: return T^N;
```

Algorithm 2 obtains a set of snapshot traces for each one of the non-conventional operational profiles N_i, given a set of associated anomalies $A = \{\alpha_1, \ldots, \alpha_n\}$ expressed as PCTL state formulae, a set of adaptations associated with each of the anomalies $Ad_{i \in \{1,\ldots,n\}} = \{(c_1, g_1, d_1), \ldots, (c_p, g_p, d_p)\}$, $(s.t.\ \forall c_{j \in \{1,\ldots,p\}}, c_j = \alpha_i)$, and a scenario represented as a snapshot trace $S = \langle s_1, \ldots, s_q \rangle$. The algorithm returns a set of snapshot traces $T_i \in T^N$, maintains a current snapshot trace tr_i, and also a clock associated with each of the non-conventional operational profiles. For every snapshot in the input snapshot trace ST, the algorithm evaluates whether any of the anomalies hold (line 5). If this is the case and the clock for the corresponding anomaly is zero, the clock is set to the maximum deadline for any of the adaptations in Ad_i (line 6) and a new current trace for N_i is allocated (line 7). Next, the current snapshot is added to all current traces with non-zero clocks (line 10). These clocks are decremented after the processing of each subsequent snapshot (line 11), and when the value of a clock becomes zero, the algorithm stops collecting snapshots for the current trace and adds it to T_i (line 13). Function *extract* (line 4) returns and removes the first element of a vector $V = \langle e_1, e_2, \ldots, e_n \rangle$: $extract(V) = e_1,\ with\ V = \langle e_2, \ldots, e_n \rangle$.

C. Model Generation

Once snapshot traces for a non-conventional operational profile have been obtained, we use them to build its corresponding probabilistic model.

Algorithm 3 DTMC Synthesis.

Input: Set of traces T associated with a non-conventional operational profile N, set of atomic propositions AP.
Output: DTMC (S, S_0, P, L) associated with N.
Auxiliary variables: Transition observation matrix O.

```
1:  S := ∅; S_0 := ∅; P := 0; L := ∅;
2:  O := 0;
3:  for tr ∈ T do
4:      current := ∅;
5:      first := true;
6:      while tr ≠ ⟨⟩ do
7:          prev := current;
8:          current := extract(t);
9:          if first ∧ current ∉ S_0 then
10:             S_0 := S_0 ∪ {current};
11:             first := false;
12:         end if
13:         if current ∉ S then
14:             S := S ∪ {current};
15:             L := L ∪ {(current, {α ∈ AP | current ⊨ α})};
16:         end if
17:         if prev ≠ ∅ then
18:             O(prev, current) := O(prev, current) + 1;
19:             SC := {s' ∈ S | O(prev, s') ≠ 0};
20:             for s' ∈ SC do
21:                 P(prev, s') := O(prev, s')/ ∑_{sc∈SC} O(prev, sc);
22:             end for
23:         end if
24:     end while
25: end for
26: return (S, S_0, P, L);
```

In particular, Algorithm 3 synthesizes a DTMC from a set of snapshot traces T and the set of atomic propositions of interest AP. The algorithm starts by incrementally building the sets of states (initial and global, lines 10, 14), and their corresponding labelling (line 15). Moreover, for the second part of the algorithm where the transition probability matrix

is built, we employ an auxiliary matrix O that is used to keep information about the number of times that a transition between any two states *prev* and *current* is observed (line 18). Hence, values in the transition probability matrix are updated according to this information by assigning to each of the successors of state *prev* (*i.e.*, all states $s' \in S$ *s.t.* $O(prev, s') \neq 0$) a value proportional to the number of times that the transition $prev \longrightarrow current$ has been observed *w.r.t.* the total number of transitions observed from source state *prev* (line 21).

D. Property Verification

To verify properties, we need to translate the probabilistic models obtained for each of the non-conventional operational profiles into an appropriate language to be used as input to a probabilistic model checker. Concretely, we use PRISM [16], a widely used tool where DTMCs can be written as a set of commands of the form:

```
[] guard -> prob_1 : update_1 + ... + prob_n : update_n;
```

Where the guard is a predicate over all the variables in the model and each update describes a valid transition if the guard is true. A transition is specified by giving the new values of the variables considered. Furthermore, each update occurs with an assigned probability. Translation of our DTMC models results in a set of commands, each of them associated to a state (snapshot) $s = \langle v_1, \ldots, v_n \rangle$ and its set of successor states $S' = \{s'_1, \ldots, s'_m\}$, $s'_j = (v'_{j1}, \ldots, v'_{jn}), j \in \{1, \ldots, m\}, i \in \{1, \ldots, n\}$, according to the following pattern:

$$[] \bigwedge_{i=1}^{n} v_i \text{ -> } P(s, s'_1) : \bigwedge_{i=1}^{n} v'_{1i} + \ldots + P(s, s'_m) : \bigwedge_{i=1}^{n} v'_{mi}$$

Finally, to assess the reliability of the probabilities obtained from the model checker, we need to estimate how good is the resulting model. This is achieved by comparing representative indicators measured directly from the execution of the system against the same measures estimated in the synthesized model. In particular, in the case of resilience properties, we use as an indicator the *Mean Time To Recovery* (MTTR), that is, the average time that the system takes to recover from an experienced anomaly. Comparing the real MTTR of the system against the one obtained from a probabilistic model helps us to estimate how accurate the model is. In order to evaluate the MTTR in models, DTMCs can be easily extended into *Discrete Markov Reward Models* (DMRM) [14], where rewards (or costs) can be quantified.

V. Experimental Validation

The aim of our validation is to determine the accuracy of probability estimations, and how different parameters in model construction (*i.e.*, resolution and time) affect this accuracy. For this purpose, we compare probability estimations against indicators that can be measured directly in the running system. In particular, we have evaluated the resilience of Znn.com to a phenomenon known as *slashdot effect* when it gets flooded with visits within a period of time (from a few hours to a couple of days [19]), causing a range of problems, including total system shut down in the worst cases. Concretely, we estimate the levels of confidence expected *w.r.t.* a particular set of resilience properties when the system adapts to a sudden rise in requests received. To obtain our models, we identified a changeload characteristic of a slashdot-type effect, based on a sample collected by Juric [13] (Figure 4), previously used for a general evaluation of the effectiveness of Rainbow in Znn.com [7].

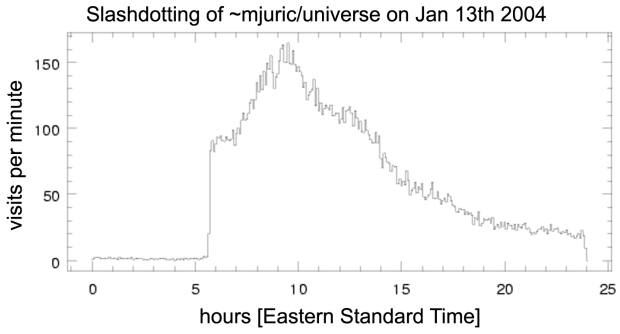

Figure 4. Graph of traffic for a website experiencing a Slashdot-like effect

Scenarios in the changeload conform to the pattern: (i) initial period of low activity; (ii) period of sharp rise in requests; and (iii) sustained period of peak in requests.

Our data sets were obtained from 100 separate runs of Rainbow running on top of the Znn.com simulator included in the Rainbow SDK. Information was collected according to Algorithm 2, and using Algorithm 3 we synthesized three different abstractions for the behavior of the system under the non-conventional operational profile $N_{cViolation}$, using different quantization parameters for the experienced response time $\eta_{expRspTime} \in \{10, 50, 100\}$ for high ($\mathcal{M}_h$), medium ($\mathcal{M}_m$), and low-level ($\mathcal{M}_l$) resolution models, respectively. A fixed quantization parameter was chosen for the cost of operation $\eta_{totalCost} = 100$ (cost/active server). The time sampling parameter used for all models is $\tau = 1s$. Models $\mathcal{M}_x$ are *time-homogeneous* DTMCs, where one-step transition probabilities are always the same, independently of time [14]. Moreover, in order to compare results against their time-homogeneous counterparts, we synthesized three additional abstractions ($\mathcal{M}^\tau_h$, $\mathcal{M}^\tau_m$, $\mathcal{M}^\tau_l$) of similar characteristics, but including time in the model. Adaptations that do not quickly counteract the sudden rise in demand are not effective, so the maximum time bound for the properties checked is $100s$. (length of collected traces).

Table II details the experimental results obtained for the different models. As expected, the size of the models with coarser quantization parameters is remarkably smaller (up to 79% and 46% reduction between $\mathcal{M}_h - \mathcal{M}_l$ and $\mathcal{M}^\tau_h - \mathcal{M}^\tau_l$, respectively), due to the higher ratio of observations grouped

per abstract state of the model. This is also the reason for the drastic reduction in size (about 1 order of magnitude) of time-homogeneous models, against those including time. However, this contrasts with the accuracy of the MTTR in the different models, which indicates a high representativity of the system's behavior and only presents slight variations ($\approx 95-96\%\mathcal{M}_x, 96-99\%\mathcal{M}_x^\tau$) independently of the size and type of model. It is worth observing that the best accuracy in MTTR is not achieved by the models with the smallest quantization parameters, but when a medium quantization parameter (*i.e.*, $\eta_{expRspTime} = 50$) is chosen in both $\mathcal{M}_x$ and $\mathcal{M}_x^\tau$ models. This is because moderately coarse resolutions in quantization parameters yield more representative information about the dynamics of the system than finer resolutions if the number of system observations available is not particularly high. However, if the quantization parameter chosen is too high *w.r.t.* the range of possible values of the variable, the loss in resolution will also distort the representativeness of the system dynamics. This puts forward the importance of striking a balance between required resolution and representativeness of the system's dynamics to choose appropriate quantization parameters for the models.

Table II
MODEL METRICS FOR $N_{cViolation}$

$N_{cViolation}$ $\tau = 1s$	$\mathcal{M}_h$ / $\mathcal{M}_h^\tau$ $\eta_{expRspTime}=10$ $\eta_{totalCost}=100$	$\mathcal{M}_m$ / $\mathcal{M}_m^\tau$ $\eta_{expRspTime}=50$ $\eta_{totalCost}=100$	$\mathcal{M}_l$ / $\mathcal{M}_l^\tau$ $\eta_{expRspTime}=100$ $\eta_{totalCost}=100$
Model Size (#States-#Transitions)	582-977 / 3691-4344	221-435 / 2507-3154	120-218 / 1995-2559
MTTR/MaxTTR (s)	19.74 / 43		
Estimated MTTR (s)	20.74 / 20.41	20.43 / 19.72	20.46 / 19.57
Accuracy MTTR (%)	95.17 / 96.71	96.66 / 99.89	96.48 / 99.13

Regarding property verification, Table III presents information about the probability of satisfaction for two PCTL formulas specified according to the patterns in Table I, with the implicit premise cViolation. These probabilities are computed for the subformula enclosed by the probabilisic operator $\mathcal{P}_{\bowtie p}(.)$ with the mean (MTTR) and the maximum (MaxTTR) times to recovery as time bounds. In general, the accuracy of the measures obtained in $\mathcal{M}_x^\tau$ models is higher. This can be observed in property P1, which can be used as an indicator of accuracy, since a perfect probability estimation with t = MaxTTR should be 1 (by definition, the system always recovers before MaxTTR), and should be very close to 0.5 with t = MTTR. While the error range in the estimation of P1 with t = MaxTTR for $\mathcal{M}_x^\tau$ models is 1.5-2%, in the case of $\mathcal{M}_x$ it is 9-10.4%. Moreover, accuracy in $\mathcal{M}_x^\tau$ experiences less degradation with lower resolutions. This can be observed in probabilities for the same property and time bound experiencing a maximum variation of 0.5% in $\mathcal{M}_x^\tau$ models, against 5% in $\mathcal{M}_x$ models. Regarding property P2, we can observe that its probability of satisfaction is low (≈ 0.3), and does not vary with different time bounds, since in all paths where hiCost holds in any of its states, this always happens before time MTTR. Let us remark that, unlike in P1, the probability of satisfaction for P2 with time bound MaxTTR is unknown and can only be estimated using a model.

Table III
SATISFACTION PROBABILITIES FOR PROPERTIES IN $N_{cViolation}$

PCTL Formula	Model	Satisfaction Probability	
		t = MTTR	t = MaxTTR
P1 $F^{\leq t}\neg$cViolation	$\mathcal{M}_h$ / $\mathcal{M}_h^\tau$	0.594 / 0.517	0.910 / 0.985
	$\mathcal{M}_m$ / $\mathcal{M}_m^\tau$	0.626 / 0.517	0.898 / 0.983
	$\mathcal{M}_l$ / $\mathcal{M}_l^\tau$	0.644 / 0.517	0.896 / 0.980
P2 $\neg$hiCost$U^{\leq t}\neg$cViolation	$\mathcal{M}_h$ / $\mathcal{M}_h^\tau$	0.303 / 0.276	0.303 / 0.276
	$\mathcal{M}_m$ / $\mathcal{M}_m^\tau$	0.331 / 0.276	0.334 / 0.276
	$\mathcal{M}_l$ / $\mathcal{M}_l^\tau$	0.328 / 0.276	0.334 / 0.276

VI. RELATED WORK

Many methodologies support the analysis of non-functional properties, based either on modeling, or direct measurement using an existing system implementation.

While modeling is a useful approach to help in the development of a system for which there is no available implementation yet, it heavily relies on model parameter estimations obtained either from domain experts, or from other similar existing systems. In particular, if we focus on the intersection between self-adaptive systems and modeling using probabilistic models, Calinescu and Kwiatkowska [4] introduce an autonomic architecture that uses Markov-chain quantitative analysis to dynamically adjust the parameters of an IT system according to its environment and objectives. However, this work assumes that Markov chains describing the different components in the system are readily available and are used as input to the problem.

Regarding direct measurement, Epifani *et al.* [9] present a methodology and framework to keep models alive by feeding them with run-time data to update their internal parameters, that should otherwise be provided by domain experts. The framework focuses on reliability and performance, and uses DTMCs and Queuing Networks as models to reason about non-functional properties. Moreover, Metzger *et al.* [18] describe a testing and monitoring framework oriented towards quality prediction in *Service-Based Applications*. Specifically, the framework is focused on dynamic selection of service bindings. To achieve this goal, the main activity of the framework is test case selection for online testing, which is combined with monitoring to predict the quality of the services and proactively determine the need for adaptation.

A complementary approach is taken by Calinescu *et al.* [3], who extend and combine [4] and [9], describing a tool-supported framework for the development of adaptive service-based systems. QoS requirements are translated into probabilistic temporal logic formulae used for analysis to identify and enforce optimal system configurations.

Our approach focuses on quantitative analysis using measurement, not assuming the existence of a Markov-chain for

the system nor its components. Moreover, most proposals deal with service-based applications that rely on estimates of the future behavior of the system to optimize its operation by adjusting parameters, trying to enforce policies, or proactively determining the need for adaptation. In contrast, our approach focuses on providing levels of confidence *w.r.t.* behavior of the system related with adaptations which have already taken place. In order to manage the complexity, information is selectively collected and aggregated into models specific to the particularities of resilience properties.

VII. Conclusions and Future Work

In this paper, we have defined an approach for the verification of self-adaptive systems that relies on probabilistic model-checking for obtaining levels of confidence regarding trustworthy service delivery when a system undergoes adaptation. Moreover, we have studied how different parameters in model construction, such as resolution, or the inclusion of time, affect the accuracy of the results. Our approach has been validated by assessing probabilistic response properties in Znn.com, facing a slashdot-like effect. Experimental results have shown that including time in models yields more accurate results regarding the estimation of satisfaction probability for the properties, although at the cost of increasing model size one order of magnitude *w.r.t.* time-homogeneous models. However, a moderate reduction in resolution always results in a remarkable reduction in the size of models without compromising accuracy in probability estimations, especially in models including time. Anyhow, time-homogeneous models still yield reasonable results, making them useful in run-time applications that require a compromise between accuracy and resources.

Regarding future work, we plan to extend our approach to enable comparison of alternative adaptations and incorporate feedback about probabilities in the selection of the best course of action at run-time in a self-adaptive system.

Acknowledgements

Co-financed by the Foundation for Science and Technology via project CMU-PT/ELE/0030/2009 and by FEDER via the "Programa Operacional Factores de Competitividade" of QREN with COMPETE reference: FCOMP-01-0124-FEDER-012983. The authors would like to thank Paulo Casanova and Bradley Schmerl from CMU, Shang-Wen Cheng at NASA JPL, and Rafael Ventura at University of Coimbra for their support with the deployment of Rainbow.

References

[1] J. Andersson, R. de Lemos, S. Malek, and D. Weyns. Modeling dimensions of self-adaptive software systems. In *SEfSAS*, volume 5525 of *LNCS*, pages 27–47. Springer, 2009.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.

[4] R. Calinescu and M. Z. Kwiatkowska. Using Quantitative Analysis to Implement Autonomic IT Systems. In *ICSE*, pages 100–110. IEEE, 2009.

[5] B. H. Cheng, R. de Lemos, et al. Software Engineering for Self-Adaptive Systems: a Research Roadmap. In *SEfSAS*, volume 5525 of *LNCS*, pages 1–26. Springer, 2009.

[6] S.-W. Cheng. *Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation*. PhD thesis, CMU, 2008.

[7] S.-W. Cheng, D. Garlan, and B. R. Schmerl. Evaluating the Effectiveness of the Rainbow Self-Adaptive System. In *SEAMS*, pages 132–141. IEEE, 2009.

[8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *ICSE*, pages 411–420, 1999.

[9] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model Evolution by Run-Time Parameter Adaptation. In *ICSE*, pages 111–121. IEEE CS, 2009.

[10] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37(10):46–54, 2004.

[11] L. Grunske. Specification Patterns for Probabilistic Quality Properties. In *ICSE*, pages 31–40. ACM, 2008.

[12] S. Hart, M. Sharir, and A. Pnueli. Termination of Probabilistic Concurrent Programs. In *POPL*, pages 1–6. ACM, 1982.

[13] M. Juric. Slashdotting of mjuric/universe. http://www.astro.princeton.edu/~mjuric/universe/slashdotting/, 2004.

[14] V. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman and Hall, 1995.

[15] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic Model Checking. In *SFM*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007.

[16] M. Z. Kwiatkowska, G. Norman, and D. Parker. Quantitative Analysis With the Probabilistic Model Checker PRISM. *Electr. Notes Theor. Comput. Sci*, 153(2):5–31, 2006.

[17] J.-C. Laprie. From Dependability to Resilience. In *DSN Fast Abstracts*. IEEE CS, 2008.

[18] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl. Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications. In *COMPSAC*, pages 582–587. IEEE CS, 2011.

[19] D. Terdiman. Solution for Slashdot Effect? http://www.wired.com/science/discoveries/news/2004/10/65165, 2004.