
Proceedings Paper:
Paterson, Colin Alexander and Calinescu, Radu Constantin (Accepted: 2017) Accurate Analysis of Quality Properties of Software with Observation-Based Markov Chain Refinement. In: IEEE International Conference on Software Architecture (ICSA 2017). IEEE. (In Press)

Accurate Analysis of Quality Properties of Software with Observation-Based Markov Chain Refinement

Colin Paterson and Radu Calinescu
Department of Computer Science, University of York, UK

***Abstract*—We introduce a tool-supported method for the automated refinement of continuous-time Markov chains (CTMCs) used to assess quality properties of component-based software. Existing research focuses on improving the efficiency of CTMC analysis and on identifying new applications for this analysis. As such, ensuring that the analysis is accurate by using CTMCs that closely model the behaviour of the analysed software has received relatively little attention. Our new method addresses this gap by refining the high-level CTMC model of a component-based software system based on observations of the execution times of its components. Our refinement method reduced analysis errors by 77–90.3% for a service-based system implemented using six public web services from three different providers, improving the accuracy of the analysis and significantly reducing the risk of invalid software engineering decisions.**

I. Introduction

Software performance and reliability engineering [1] uses mathematical models to predict performance, reliability and other quality properties of software [2]. These models include Petri Nets [3], queuing networks [4] and Markov chains [5], [6], [7], and may be built manually or extracted automatically from more general software models such as UML-MARTE [8] or Palladio [9].

Despite major advances in this research area, it is still very challenging to ensure that such mathematical models are sufficiently accurate to support the design and verification of real systems. Our paper addresses this challenge for *continuous-time Markov chains* (CTMCs), a type of stochastic state-transition models used to analyse quality software properties both at design time [5], [10], [11] and at runtime [12], [13], [14], [15]. We introduce an observation-based Markov chain refinement (OMNI) method that significantly improves the accuracy with which quality software properties can be analysed. OMNI starts from a high-level abstract CTMC whose states correspond to operations performed by different components of the analysed system, and uses observed execution times for the components to refine this CTMC. The high-level CTMC can be generated (e.g., from annotated UML activity diagrams as in [10]) or can be provided by the software engineers. The execution time observations can be obtained by unit testing components individually before the software system is built or, for existing systems, from their logs.

OMNI comprises two stages. The first stage makes the CTMC more realistic through the addition of states and transitions that model the fact that software components have non-zero minimum execution times. We use additional states and transitions corresponding to Erlang distributions [16] for this purpose. The CTMC is then further refined in the second OMNI stage by using phase-type distributions [17] to model the variations in the execution time of the components. The refined CTMC supports the accurate analysis of a wide range of quality software properties expressed in continuous stochastic logic [18], preventing many invalid design decisions and verification conclusions associated with traditional CTMC analysis. Moreover, the refined CTMCs can be analysed using existing probabilistic model checkers such as PRISM [19].

To support the use of OMNI, we describe rigorous techniques for synthesising its Erlang and phase-type distributions. These distributions model the execution of software components with far greater accuracy than the exponential distributions from existing CTMC modelling, which match only the first moment of the unknown distributions of the execution time observations. In addition, we provide a tool that implements the OMNI method, producing refined CTMC models that can be analysed with the probabilistic model checker PRISM [19]. Finally, we present a case study that shows how software engineers can use OMNI to avoid multiple invalid design decisions suggested by traditional CTMC analysis. To ensure the reproducibility of our results, we provide the models, code and data from this work on our project webpage www-users.cs.york.ac.uk/cap/OMNI/.

The remainder of the paper is organised as follows. Sections II and III introduce the theoretical background for our work and a running example that we use to motivate and illustrate the refinement method, respectively. We then present the OMNI method in Section IV and the tool that we implemented to automate its use in Section V. Section VI evaluates the effectiveness of our method through a case study that uses OMNI to analyse the service-based system from the running example, and Section VII discusses related work. We analyse threats to validity in Section VIII, and present our conclusions and future work directions in Section IX.

II. Preliminaries

Continuous Time Markov Chains—Markov chains are mathematical models for stochastic processes evolving in time [20]. A continuous-time stochastic process $\{X(t) : t \geq 0\}$ with countable state space S is a continuous-time Markov chain (CTMC) if it has the *Markov property*

$$Prob\{X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n, X(t_{n-1}) = s_{n-1}, \ldots, X(t_1) = s_1\} = Prob\{X(t') = s' \mid X(t) = s\}, \quad (1)$$

TABLE I: Third-party web services used to develop the travel application

Label	Thid-party service	URL	rate (s^{-1})
location	Bing location service	http://dev.virtualearth.net/REST/v1/Locations/	9.62
arrivals	Thales rail arrival board	http://www.livedepartureboards.co.uk/ldbws/	19.88
departures	Thales rail departures board	http://www.livedepartureboards.co.uk/ldbws/	19.46
search	Bing web search	https://api.datamarket.azure.com/Bing/Search/v1/	1.85
weather	WebserviceX.net weather service	http://www.webservicex.net/globalweather.asmx?op=GetWeather	1.11
traffic	Bing traffic service	http://dev.virtualearth.net/REST/v1/Traffic/	2.51

where $s_1, \ldots, s_{n-1}, s_n, s_{n+1} \in S$ and $0 \leq t_1 \leq ... \leq t_{n-1} \leq t_n \leq t_{n+1}$, $n \geq 1$, is any sequence of $n+1$ times. For the work presented in this paper, we use the following CTMC definition adapted from [21].

Definition 1: A continuous-time Markov chain is a tuple

$$(S, \boldsymbol{\pi}, \mathbf{R}), \tag{2}$$

where S is a finite set of states, $\boldsymbol{\pi} : S \rightarrow [0,1]$ is an initial-state probability vector such that the probability that the CTMC is initially in state $s_i \in S$ is given by $\boldsymbol{\pi}(s_i)$ and $\sum_{s_i \in S} \boldsymbol{\pi}(s_i) = 1$, and $\mathbf{R} : S \times S \rightarrow \mathbb{R}$ is a transition rate matrix such that, for any states $s_i \neq s_j$ from S, $\mathbf{R}(s_i, s_j) \geq 0$ specifies the rate with which the CTMC transitions from state s_i to state s_j, and $\mathbf{R}(s_i, s_i) = -\sum_{s_j \in S \setminus \{s_i\}} \mathbf{R}(s_i, s_j)$.

We use the notation $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ for the CTMC (2). The probability that this CTMC will transition from state s_i to another state within t time units is $1 - e^{-t \cdot \sum_{s_k \in S \setminus \{s_i\}} \mathbf{R}(s_i, s_k)}$, and the probability that the new state is $s_j \in S \setminus \{s_i\}$ is

$$p_{ij} = \mathbf{R}(s_i, s_j) \; / \; \Sigma_{s_k \in S \setminus \{s_i\}} \mathbf{R}(s_i, s_k). \tag{3}$$

Continuous Stochastic Logic—Similar to the probabilistic model checkers used to analyse CTMCs in software performance engineering (e.g., PRISM [19]), OMNI uses continuous stochastic logic (CSL) extended with rewards to specify the quality properties analysed over CTMCs [18], [21].

Definition 2: Let AP be a set of atomic propositions, $a \in AP$, $p \in [0,1]$, I an interval in $\mathbb{R}$ and $\bowtie \in \{\geq, >, <, \leq\}$. Then a state formula Φ and a path formula Ψ in CSL are defined by the following grammar:

$$\begin{array}{l} \Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\bowtie p}[\Psi] \mid S_{\bowtie p}[\Psi] \\ \Psi ::= X\Phi \mid \Phi U^I \Phi \end{array} . \tag{4}$$

CSL formulae are interpreted over a CTMC whose states are *labelled* with atomic propositions from AP by a function L: $S \rightarrow 2^{AP}$. Path formulae only occur inside the probabilistic operator P and *steady-state* operator S, which define bounds on the probability of system evolution, e.g., a state s satisfies a formula $P_{\bowtie p}[\Phi]$ if the probability of the future evolution of the system meets the bound '$\bowtie p$'. For a path, the "next" formula $X\Phi$ holds if Φ is satisfied in the next state; the "bounded until" formula $\Phi_1 U^I \Phi_2$ holds if before Φ_2 becomes true at time $t \in I$, Φ_1 is satisfied continuously in the interval $[0, t)$. If $I = [0, \infty)$, the formula is termed "unbounded until". The notation $F^I \Phi \equiv \mathsf{true}\; U^I \Phi$ is used when the first part of an until formula is true, and $s \models \Phi$ and $M \models \Phi$ indicate that Φ is satisfied in state s and for the initial state distribution $\boldsymbol{\pi}$ of a CTMC $M = (S, \boldsymbol{\pi}, \mathbf{R})$, respectively. Probabilistic model checkers also support formulae in which the bound '$\bowtie p$' is replaced with '=?', to indicate that the computation of the actual bound is required. For a formal definition of the CSL semantics, see [21].

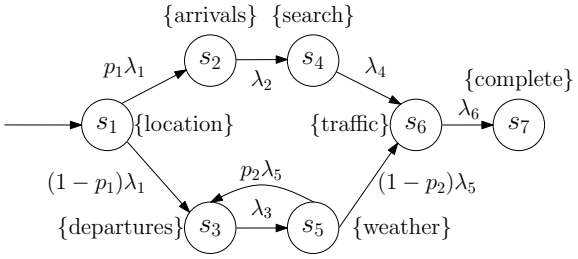


Fig. 1: High-level CTMC model of request handling

III. Motivating Example

To motivate our work, we consider a travel web application that needs to use the third-party services from Table I in order to handle two types of requests:

1. Requests from users who plan to meet and entertain a visitor arriving by train. These requests are expected to occur with probability p_1.
2. Requests from users looking for a possible destination for a day trip by train. These requests are expected to occur with probability $1 - p_1$.

The high-level abstract CTMC from Fig. 1 (derived from an activity diagram) models the handling of a request by the application. The initial state s_1 corresponds to finding the location of the train station. For the first request type, this is followed by finding the train arrival time (state s_2), identifying suitable restaurants in the area (s_4), obtaining a traffic report for the route from the user's location to the station (s_6), and returning the response to the user (s_7). For the second request type, state s_1 is followed by finding a possible destination (s_3), and obtaining a weather forecast for this destination (s_5). With a probability of p_2 the weather is unsuitable and a new destination is selected (back to s_3). Once a suitable destination is selected, the traffic report is obtained (s_6) and the response returned to the user (s_7).

The operation execution rates λ_1 to λ_6 depend on the components used for these operations, and the engineers want to decide if the real services from Table I are suitable for building the application. If they are, the engineers need: (i) to select appropriate request-handling times to be promised in the

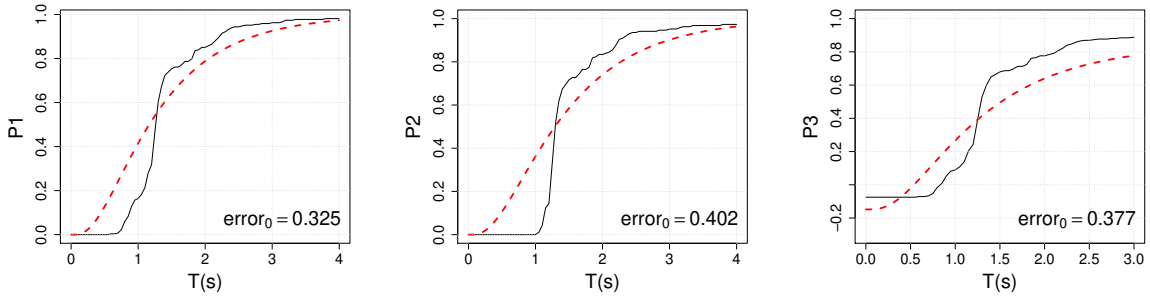


Fig. 2: Predicted (dashed lines) versus actual (continuous line) values for properties **P1**–**P3**

application service-level agreement (SLA); and (ii) to choose a pricing scheme for the application. Therefore, the engineers are interested to assess the following quality properties of the application variant built using these services:

P1 The fraction of user requests handled in under T seconds, for $0<T\leq 4$.

P2 The fraction of "day trip" requests handled in under T seconds, for $0<T\leq 4$.

P3 The expected profit per request handled, assuming that 1 cent is charged for requests handled within T seconds and a 2-cent penalty is paid for requests not handled within 3 seconds, for $0 < T \leq 3$.

Service response times are assumed exponentially distributed in CTMC modelling, so the engineers use observed response times $t_{i1}, \ldots, t_{in}$ for service i (taken from existing logs or obtained through testing) to estimate the service rate λ_i as

$$\lambda_i = \left(\frac{t_{i1} + t_{i2} + \cdots + t_{in}}{n} \right)^{-1}. \tag{5}$$

Finally, they use the CTMC to analyse the CSL-formalised properties **P1**–**P3**:

$$\begin{array}{ll} \textbf{P1} & P_{=?}[F^{[0,T]}\, complete] \\ \textbf{P2} & P_{=?}[\neg arrivals\ U^{[0,T]}\, complete]/(1-p_1) \\ \textbf{P3} & P_{=?}[F^{[0,T]}\, complete] - 2 \cdot P_{=?}[F^{(3,\infty)}\, complete] \end{array} \tag{6}$$

where $0 < T \leq 4$ for **P1** and **P2**, and $0 < T \leq 3$ for **P3**. To replicate this process, we built a prototype version of the application and used it to handle 270 randomly generated requests for $p_1 = 0.3$ and $p_2 = 0.1$. We obtained sample execution times for each service (between 81 for arrivals and search and 270 for location and traffic), and used (5) to calculate the estimate service rates in Table I. We then used the model checker PRISM [19] to analyse the CTMC for these rates, and thus to predict the values of properties (6). To assess the accuracy of the predictions, we calculated the actual values of these properties using detailed timing information logged by our application. The predictions obtained through CTMC analysis and the actual property values are compared in Fig. 2. The errors reported in the figure give the area difference between the actual and predicted property values:

$$error = \int_0^{T_{\max}} |actual(T) - predicted(T)|\ dT, \tag{7}$$

where $T_{\max} = 4$ for properties **P1** and **P2**, and $T_{\max} = 3$ for **P3**. We will later use these errors to measure the accuracy improvements due to our OMNI model refinement. For now, recall that the engineers must make decisions based only on the predicted property values; two such decisions could be:

- Implement the application with the services from Table I, with an SLA promising that 40% of the requests will be handled within 1s (property **P1**), 35% of the "day trip" requests will be handled also within 1s (property **P2**), and charge 1 cent for requests handled within 1s (property **P3**). This decision would be wrong, as both promises would be violated by a wide margin, and the actual profit would be under a third of the predicted profit (cf. Fig. 2).
- Look for alternative services for the application, because not even 80% of the requests or "day trip" requests are handled within 2s, and/or because the profit is below 0.7 cents per request when charging 1 cent for each request handled within 2s. This decision would also be wrong, since all the constraints it is based on would actually be satisfied by the application (Fig. 2).

Clearly, we chose the times and bounds above so as to show that the current practice of using idealised CTMC models *may* lead to blatantly invalid decisions. Using other times and bounds will yield valid decisions. However, we argue that engineering decisions must be consistently valid, and not down to chance.

IV. CTMC Refinement Method

Let $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ be a high-level CTMC model of a system such that:

- each state $s_i \in S$ corresponds to operation i of the system and $\boldsymbol{\pi}(s_i)$ is the probability that this is the initial operation;
- for any $s_i \neq s_j$ from S, $\mathbf{R}(s_i, s_j) = p_{ij}\lambda_i$, where p_{ij} is the (known or estimated as in [22], [23]) probability (3) that operation i is followed by operation j, and λ_i is calculated as in (5), using $n > 0$ observed execution times $t_{i1}, t_{i2}, \ldots, t_{in}$ of operation i.

This CTMC model makes the typical assumption that operation execution times are exponentially distributed. However, this assumption is almost always invalid for two reasons. First, each software operation i has a minimum execution time $\underline{t_i} > 0$ such that its probability of completion within $\underline{t_i}$ time units is zero. Second, even the "holding times"

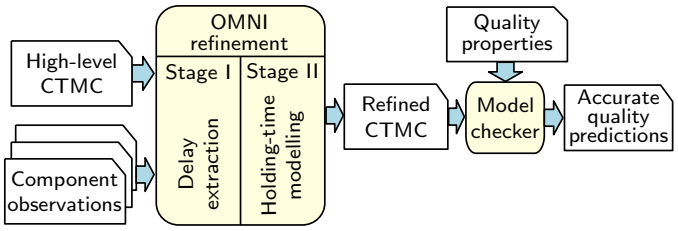


Fig. 3: Accurate analysis of quality properties with OMNI

$t_{i1} - \underline{t}_i, t_{i2} - \underline{t}_i, \ldots, t_{in} - \underline{t}_i$ of an operation i are rarely exponentially distributed. As we showed in Section III, these issues can lead to erroneous analyses of the quality properties of the modelled system.

OMNI overcomes these issues by producing a refined CTMC whose analysis with *existing* model checkers supports the accurate evaluation of quality properties of software (Fig. 3). This is achieved through the use of phase-type distributions (PHDs) that accurately model the variations in execution times for each component. The PHD fitting of distributions with deterministic delays is known to require extremely large numbers of states, and such delays are best fitted by Erlang distributions [24], as we confirm experimentally in Section VI. As such, the OMNI refinement comprises two stages. In the first stage, the high-level CTMC is refined to better model the minimum execution times of software operations. We term this stage *delay extraction*. In the second stage, the CTMC is further refined through the use of PHD fitting. We call this stage *holding-time modelling*. The two OMNI stages are described next.

A. Stage I: Delay Extraction

In this OMNI stage, the CTMC is extended with additional states and transitions that model the minimum execution times (i.e. *delays*) of the software operations by means of Erlang distributions, i.e., sums of several independent exponential distributions with the same rate [16]. With the notation above, state s_i and its $m_i \geq 1$ outgoing transitions with rates $p_{i1}\lambda_i$, $p_{i2}\lambda_i$, …, $p_{im_i}\lambda_i$ are replaced (Fig. 4) with a sequence of delay-modelling states $s_{i1}, s_{i2}, \ldots, s_{ik_i}$ that encode an Erlang-k_i distribution with rate λ_i^E, and a state s_i' with outgoing transitions of rates $p_{i1}\lambda_i'$, $p_{i2}\lambda_i'$, …, $p_{im_i}\lambda_i'$ to the same next states as s_i. However, delays are not modelled perfectly by Erlang distributions: for any *error* $\epsilon \in (0,1)$, there is a (small) probability p that the refined CTMC leaves state s_{ik_i} within $\underline{t}_i(1-\epsilon)$ time units of entering s_{i1}. Given specific values for ϵ and p, the theorem below supports the calculation of the parameters k_i, λ_i^E and λ_i' for our *delay extraction*.

Theorem 1: Given an error bound $\epsilon \in (0,1)$, if the delay-extraction refinement parameters k_i, λ_i^E and λ_i' satisfy

$$1 - \sum_{l=0}^{k_i-1} \frac{(k_i(1-\epsilon))^l \, e^{-k_i(1-\epsilon)}}{l!} = p, \tag{8}$$

$$\lambda_i^E = \frac{k_i}{\underline{t}_i} \quad \text{and} \quad \lambda_i' = \frac{\lambda_i}{1 - \lambda_i \underline{t}_i} \tag{9}$$

for some value $p \in (0,1)$ then the following properties hold for the refined CTMC:

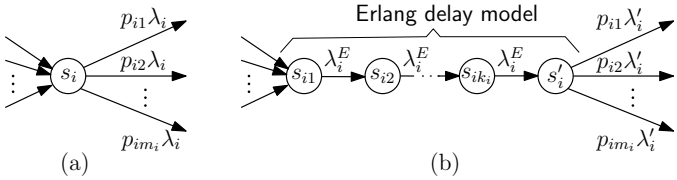


Fig. 4: Modelling operation i in the (a) abstract CTMC and (b) refined CTMC

(i) The probability that the CTMC leaves state s_{ik_i} within $\underline{t}_i(1-\epsilon)$ time units from entering state s_{i1} is p;

(ii) The expected time for the CTMC to leave s_i' after entering state s_{i1} is λ_i^{-1}.

Proof: To prove (i), we note that the cumulative distribution function of an Erlang-k distribution with rate λ is $F(k,\lambda,x) = 1 - \sum_{l=0}^{k-1} \frac{(\lambda x)^l e^{-\lambda x}}{l!}$, so (8) can be rewritten as $F(k_i, \lambda_i^E, \underline{t}_i(1-\epsilon)) = p$ since $k_i = \lambda_i^E \underline{t}_i$ according to (9). Therefore, the probability that the Erlang delay model from Fig. 4 will transition from entering state s_{i1} to exiting state s_{ik_i} within $\underline{t}_i(1-\epsilon)$ time units is p. For part (ii), the expected time for the CTMC to leave state s_i' after entering s_{i1} is the sum of the mean of the Erlang-k_i distribution with rate λ_i^E and the mean of the exponential distribution with rate λ_i', i.e. $\frac{k_i}{\lambda_i^E} + \frac{1}{\lambda_i'} = \underline{t}_i + \frac{1}{\lambda_i} - \underline{t}_i = {\lambda_i}^{-1}$. ■

Thus, we can calculate the delay model parameters for operation i as follows:

1. Approximate the minimum execution time for operation i as $\underline{t}_i = \min_{j=1}^{n} t_{ij}$;
2. Choose a small error $\epsilon \in (0,1)$ and a small probability p (e.g., $\epsilon = 0.1$ and $p = 0.05$), and solve (8) for k_i, e.g., by using a numeric solver and rounding the result up to an integer value or—since k_i only depends on ϵ and p, and is independent of $\underline{t}_i$—by using a table of precomputed k_i values as in Table II;
3. Calculate λ_i^E and λ_i' using (9).

The theorem below gives the format of the refined CTMC after the delay extraction stage. For convenience, we consider that the delay extraction procedure was applied to all states of the initial model $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$, which involves setting $k_i = 0$ and $\lambda_i' = \lambda_i$ in the Erlang delay model from Fig. 4(b) for states $s_i \in S$ that do not require delay extraction (e.g. state s_7 from our CTMC in Fig. 1).

Theorem 2: Applying the OMNI delay extraction procedure to a high-level model $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ yields the refined model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$, where:

$S' = \cup_{s_i \in S} \{s_{i1}, s_{i2}, \ldots, s_{ik_i}, s_i'\}$;

$\boldsymbol{\pi}'(s_i') = \boldsymbol{\pi}(s_i)$ and $\boldsymbol{\pi}(s_{i1}) = \cdots = \boldsymbol{\pi}(s_{ik_i}) = 0, \forall s_i \in S$;

$\boldsymbol{R}'(s_{ik}, s_{i,k+1}) = \boldsymbol{R}'(s_{ik_i}, s_i') = \lambda_i^E, 1 \leq k < k_i, \forall s_i \in S$;

$\boldsymbol{R}'(s_i', s_{j1}) = p_{ij}\lambda_i'$ for all $s_j \in S \setminus \{s_i\}, \forall s_i \in S$;

$\boldsymbol{R}'(s_{ik}', s_{jl}') = 0$ for all $s_j \in S \setminus \{s_i\}, 1 \leq k \leq k_i, 1 \leq l \leq k_j$;

$\boldsymbol{R}'(s', s') = -\Sigma_{z' \in S' \setminus \{s'\}} \boldsymbol{R}'(s', z'), \forall s' \in S'$.

Proof: The proof is by construction, cf. Fig. 4. ■

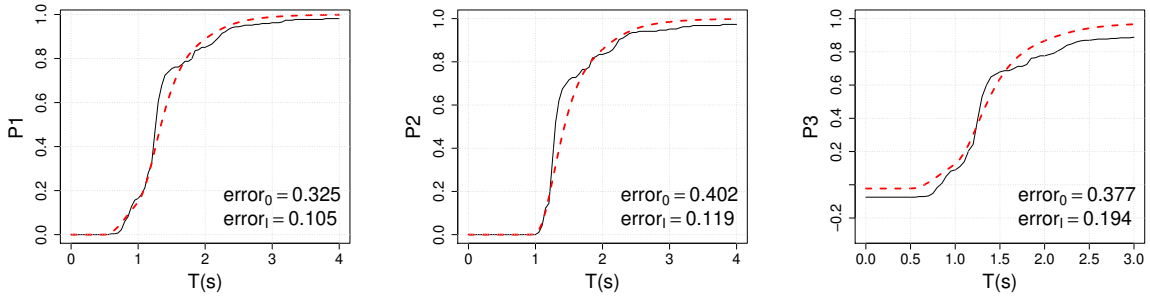


Fig. 5: Predicted after OMNI delay extraction (dashed lines) versus actual (continuous lines) values for properties **P1**–**P3**, where error_0 and error_I are calculated as in (7), before and after delay extraction

TABLE II: Precomputed k_i values used in the OMNI evaluation from Section VI

error ϵ	0.2	0.2		0.1	0.1
prob. p	0.29	0.08	$\cdots$	0.16	0.05
k_i	10	45		100	259

Example 1 Consider again the web application from our motivating example, and its three quality properties from (6). We applied the OMNI delay extraction to each state associated with a service in the CTMC from Fig. 1. We set $\epsilon = 0.1$ and $p = 0.05$, so state s_i, $1 \leq i \leq 6$, from the original CTMC was replaced with an Erlang delay model as in Fig. 4(b), where $k_i = 259$ (cf. Table II) and the values for λ_i^E and λ_i' are given in Table III. Accordingly, the refined CTMC had 6×259 more states and transitions than the original CTMC. Fig. 5 compares the actual values of properties **P1**–**P3** with the values predicted through the analysis of the refined CTMC. As anticipated, the error (7) decreased significantly (by between 49%–70%) for all three properties, reducing the margin for engineering decision errors. Nevertheless, there are still time points T where the actual and predicted property values differ noticeably, e.g., the predicted **P1** and **P2** values for $T = 1.4$s are 0.565 (instead of an actual value of 0.723) and 0.462 (instead of 0.674), respectively. The next OMNI stage addresses this difference.

B. Stage II: Holding-Time Modelling

We further refine the CTMC from the first OMNI stage by using *phase-type distributions* (PHDs) to model the "holding times" of the system operations. A PHD [17] is defined as the distribution of the time to absorption in a CTMC with one absorbing state, i.e. the time to reach the only state $s_i \in S$ of a CTMC (2) for which $\mathbf{R}(s_i, s_j) = 0$ for all $s_j \in S$. PHDs support efficient numerical and analytical evaluation [17] and can approximate arbitrarily close any continuous distribution with a strictly positive density in $(0, \infty)$ [25]. OMNI exploits these advantages of PHDs by synthesising a PHD that models the "holding times" sample

$$sample_i = (t'_{i1} = t_{i1} - \underline{t}_i, t'_{i2} = t_{i2} - \underline{t}_i, \ldots, t'_{in} = t_{in} - \underline{t}_i) \quad (10)$$

for each operation i, and replacing state s'_i of the Erlang delay model from Fig. 4(b) with the CTMC associated with this PHD. Before describing the OMNI PHD synthesis algorithm and replacement procedure for state s'_i, we need to introduce several basic concepts about PHDs. First, consider the transition rate matrix $\mathbf{R}$ of a CTMC $(S, \boldsymbol{\pi}, \mathbf{R})$ with one absorbing state and $N \geq 1$ *transient* (i.e., non-absorbing) states called the *phases* of the PHD. Without loss of generality, we will assume that the last row of $\mathbf{R}$ corresponds to the absorbing state, i.e.

$$\mathbf{R} = \left[\begin{array}{c|c} \mathbf{D}_0 & \mathbf{d_1} \\ \hline \mathbf{0} & 0 \end{array}\right], \quad (11)$$

where the $N \times N$ sub-matrix $\mathbf{D}_0$ specifies only transition rates between transient states, $\mathbf{0}$ is a $1 \times N$ row vector with zero elements, and $\mathbf{d}_1$ is an $N \times 1$ vector whose elements specify the transition rates from the transient states to the absorbing state. The elements from each row of $\mathbf{R}$ add up to zero (cf. Definition 1), so we additionally have $\mathbf{D}_0\mathbf{1} + \mathbf{d_1} = \mathbf{0}$, where $\mathbf{1}$ and $\mathbf{0}$ are column vectors of N ones and N zeros, respectively. Thus, $\mathbf{d_1} = -\mathbf{D}_0\mathbf{1}$ and the PHD associated with this CTMC is fully defined by the sub-matrix $\mathbf{D}_0$ and the row vector $\boldsymbol{\pi}_0$ containing the first N elements of the initial probability vector $\boldsymbol{\pi}$ (as in most practical applications, we are only interested in PHDs that are acyclic and that cannot start in the absorbing state). We use the notation $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ for this PHD.

The fitting of phase-type distributions to empirical data received considerable attention [17], [26], with effective PHD fitting algorithms developed based on techniques such as moment matching [27], expectation maximisation [28], [29] and Bayes estimation [30], [31]. Recently, these algorithms have been used within PHD fitting approaches that: (a) partition the dataset into segments [29] or clusters [32] of "similar" data points; (b) employ an established algorithm to fit a PHD with a simple structure to each data segment or cluster; and (c) use these simple PHDs as branches of a PHD that fits the whole dataset. These approaches achieve better trade-offs between the size, accuracy and complexity of the final PHD than direct algorithms applied to the entire dataset.

The function HTMODEL from Algorithm 1 performs the holding-time modelling for state s'_i of the $\mathsf{CTMC}(S', \boldsymbol{\pi}', \boldsymbol{R}')$ model from Theorem 2. HTMODEL applies Reinecke et al.'s *cluster-based PHD fitting* approach [32], [33], [34] to the holding times $sample_i$ (10) (lines 2–16), uses this PHD to derive the parameters of the refined $\mathsf{CTMC}(S'', \boldsymbol{\pi}'', \boldsymbol{R}'')$ (lines 17–20), and returns this CTMC in line 21.

The PHD fitting is carried out by the while loop in lines 5–16, which iteratively assesses the suitability of PHDs obtained when partitioning $sample_i$ into $c = MinC, MinC +$

TABLE III: Erlang delay model parameters for the states of the CTMC from Fig. 1

s_i	**label**	$\underline{t}_i$ (ms)	λ_i^E (s^{-1})	λ_i' (s^{-1})	s_i	**label**	$\underline{t}_i$ (ms)	λ_i^E (s^{-1})	λ_i' (s^{-1})
s_1	location	49	5285	18.21	s_4	search	209	1239	3.01
s_2	arrivals	45	5756	188.81	s_5	weather	706	367	5.14
s_3	departures	45	5756	156.76	s_6	traffic	179	1447	4.57

Algorithm 1 Holding-time modelling with parameters:
– $MinC$ (minimum number of PHD clusters)
– $MaxC$ (maximum number of PHD clusters)
– $MaxP$ (maximum number of cluster phases)
– $FittingAlg$ (basic PHD fitting algorithm)
– $MaxSteps$ (maximum steps without improvement)

1: **function** HTMODEL(CTMC$(S', \boldsymbol{\pi}', \boldsymbol{R}')$, s_i', $sample_i$, α)
2: PHD$(\boldsymbol{\pi}_0, \mathbf{D}_0) \leftarrow null$, $minErr = \infty$
3: $improvement \leftarrow 0$, $steps \leftarrow 0$
4: $c \leftarrow MinC$
5: **while** $c \leq MaxC \wedge steps \leq MaxSteps$ **do**
6: $phd \leftarrow$ CBFITTING$(sample, c, FittingAlg, MaxP)$
7: $err \leftarrow \Delta$CDF$(sample, phd)$
8: **if** $err < minErr$ **then**
9: $improvement \leftarrow improvement + (minErr - err)$
10: $minErr \leftarrow err$
11: PHD$(\boldsymbol{\pi}_0, \mathbf{D}_0) \leftarrow phd$
12: **if** $improvement \geq \alpha$ **then**
13: $improvement \leftarrow 0$, $steps \leftarrow 0$
14: **else**
15: $steps \leftarrow steps + 1$
16: $c \leftarrow c + 1$
17: $N_i \leftarrow$ SIZEOF$(\boldsymbol{D}_0)$
18: $S'' \leftarrow (S' \setminus \{s_i'\}) \cup \{s_{i1}', s_{i2}', \ldots, s_{iN_i}'\}$
19: $\boldsymbol{\pi}'' = \left[\begin{array}{c|c} \boldsymbol{\pi}' & \mathbf{0}_{1\times N_i} \end{array}\right]$
20: $\boldsymbol{R}'' \leftarrow \left[\begin{array}{c|c} \boldsymbol{R}'(S'\setminus\{s_i'\}, S'\setminus\{s_i'\}) & \begin{array}{c}\mathbf{0}\\ \lambda_i^E\boldsymbol{\pi}_0\end{array} \\ \hline p_{i1}\boldsymbol{d}_1 \;\; p_{i2}\boldsymbol{d}_1 \;\; \cdots & \boldsymbol{D}_0 \end{array}\right]$
21: **return** CTMC$(S'', \boldsymbol{\pi}'', \boldsymbol{R}'')$

$1, \ldots, MaxC$ clusters. Line 6 obtains a PHD with c branches (corresponding to partitioning $sample_i$ into c clusters) and up to $MaxP$ phases by using the function CBFITTING, which implements the cluster-based PHD fitting from [32]. The $FittingAlg$ argument of CBFITTING specifies the basic PHD fitting algorithm applied to each cluster as explained above, and can be any of the standard moment matching, expectation maximisation or Bayes estimation PHD fitting algorithms. The quality of the c-branch PHD is established in line 7 by using the CDF-difference metric [32] to compute the difference err between $sample_i$ and the PHD. The if statement in lines 8–11 identifies the PHD with the lowest err value so far, making a record of it in line 10. Any reductions in err (i.e., "improvements") are cumulated in $improvement$ (line 9), and the while loop terminates early if the iteration counter $steps$ exceeds $MaxSteps$ before $improvement$ reaches the threshold $\alpha \geq 0$ supplied as an argument to HTMODEL and the $steps$ counter is reset (line 13).

On exit from the while loop, the elements $\boldsymbol{D}_0$, $\boldsymbol{d_1} = -\boldsymbol{D}_0\mathbf{1}$ and $\boldsymbol{\pi}_0$ of the best-fit phase-type distribution PHD$(\boldsymbol{\pi}_0, \mathbf{D}_0)$ recorded in line 11 are used to calculate the parameters S'', $\boldsymbol{R}''$ and $\boldsymbol{\pi}''$ of the refined CTMC. These calculations are shown in lines 17–20, under the simplifying assumption that $\boldsymbol{R}'$ maintains the transition rates for the outgoing and incoming transitions of states s_{ik_i} and s_i' in the last two rows and columns, respectively. This assumption is easily satisfiable by reordering the rows and columns of the matrix, so that $\boldsymbol{R}'$ has the structure

$$\boldsymbol{R}' = \left[\begin{array}{c|c} \boldsymbol{R}'(S'\setminus\{s_i'\}, S'\setminus\{s_i'\}) & \begin{array}{c}\mathbf{0}\\ \lambda_i^E\end{array} \\ \hline \lambda_i'\,[\,p_{i1} \quad p_{i2} \quad \cdots\,] & -\lambda_i' \end{array}\right] \begin{array}{l}\leftarrow s_{ik_i}\\ \leftarrow s_i'\end{array} \qquad (12)$$

where $\boldsymbol{R}'(S'\setminus\{s_i'\}, S'\setminus\{s_i'\})$ denotes matrix $\boldsymbol{R}'$ without the last row and column. The following result justifies the construction of the refined CTMC.

Theorem 3: The tuple $(S'', \boldsymbol{\pi}'', \boldsymbol{R}'')$ synthesised by Algorithm 1 defines a valid CTMC in which the probability that s_{j1} is the first state from $S'' \cap S'$ reached after state s_{ik_i} is p_{ij}, and the mean time of reaching any state in $S'' \cap S'$ from s_{ik_i} is given by the mean time to reach the absorbing state of PHD$(\boldsymbol{\pi}_0, \mathbf{D}_0)$.

Proof: Since $\boldsymbol{\pi}'$ is the initial probability vector of CTMC$(S', \boldsymbol{\pi}', \boldsymbol{R}')$, its elements sum to 1.0 and so do the elements of π'', which is therefore a valid probability vector. It is immediate to show that the rows of $\mathbf{R}''$ contain non-negative elements outside the main diagonal and that the elements on each row add up to zero, so $\boldsymbol{R}''$ is a valid transition rate matrix. Accordingly, $(S'', \boldsymbol{\pi}'', \boldsymbol{R}'')$ satisfies Definition 1 and is a valid CTMC. For the second part of the theorem, assume that the N_i transient states of the (acyclic) PHD$(\boldsymbol{\pi}_0, \boldsymbol{D}_0)$ are reached with probabilities $x_1, x_2, \ldots, x_{N_i}$. We have $[\,x_1\; x_2\; \cdots\; x_{N_i}\,] \cdot \boldsymbol{d}_1 = 1$. In the CTMC returned by Algorithm 1, we note from $\mathbf{R}''$ that the next state after s_{ik_i} is necessarily one of $s_{i1}', s_{i2}', \ldots, s_{iN_i}'$, and that the probabilities of reaching these "PHD-based" states are given by the elements of $\boldsymbol{\pi}_0$. Thus, the probabilities of reaching the states $s_{i1}', s_{i2}', \ldots, s_{iN_i}'$ from s_{ik_i} are $x_1, x_2, \ldots, x_{N_i}$ and so the probability of reaching s_{j1} from s_{ik_i} is $[\,x_1\; x_2\; \cdots\; x_{N_i}\,]\,p_{ij}\boldsymbol{d}_1 = p_{ij}\,[\,x_1\; x_2\; \cdots\; x_{N_i}\,]\,\boldsymbol{d}_1 = p_{ij}$. Finally, since the next state after s_{ik_i} corresponds to a transient state of PHD$(\boldsymbol{\pi}_0, \mathbf{D}_0)$, the mean time to reach any other state (i.e., a state from $s_{i1}, s_{i2}, \ldots$) is the same as the mean time to reach the absorbing state of the PHD. ■

Example 2 We used our OMNI implementation to perform Stage II of the refinement on the CTMC from the motivating example. Algorithm 1 was executed for each of our six services, with $\alpha = 0.1$ and with the configuration parameters

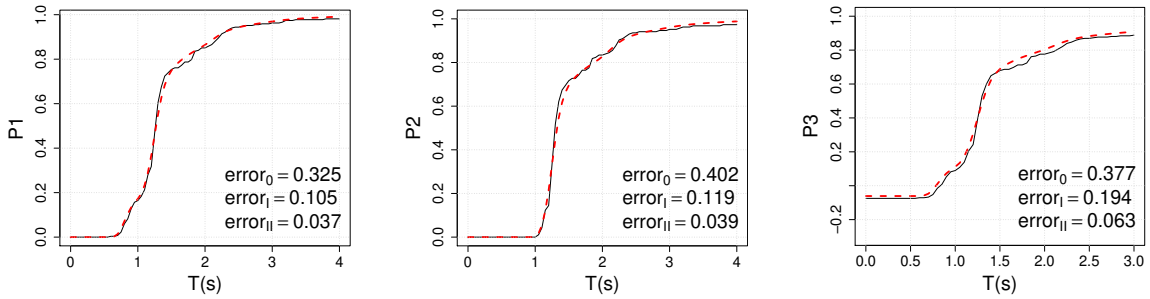


Fig. 6: Properties **P1**–**P3** predicted after holding-time modelling (dashed line) vs. actual (continuous line); error_0, error_I and error_II are the prediction errors before OMNI and after each OMNI stage, respectively

$MinC = 2$, $MaxC = 30$, $MaxP = 300$, $MaxSteps = 3$ and $FittingAlg$ an expectation-maximisation PHD fitting algorithm that produces hyper-Erlang distributions.[1] We obtained a refined CTMC with 1766 states and 1797 transitions, and Fig. 6 compares the actual values of properties **P1**–**P3** with the values predicted by the analysis of this CTMC. Both the visual assessment and the error_II error values associated with these predictions (which are significantly lower than the error values error_0 before the OMNI refinement and error_I after the first OMNI stage) show that this CTMC supports the accurate analysis of the three properties.

V. OMNI Refinement Tool

We implemented OMNI as a Java tool that accepts a CTMC model of a software system and an XML configuration file as input. The CTMC is expressed in the modelling language of PRISM [19]. The configuration file (i) maps datasets of observed execution times of the software operations to states within the CTMC; and (ii) defines all parameters that control the OMNI refinement. Our tool uses the HyperStar PHD fitting tool from [33] for the CBFitting function from Algorithm 1, and produces the refined CTMCs as a PRISM model. The OMNI tool is freely available from our project webpage www-users.cs.york.ac.uk/cap/OMNI.

VI. Evaluation

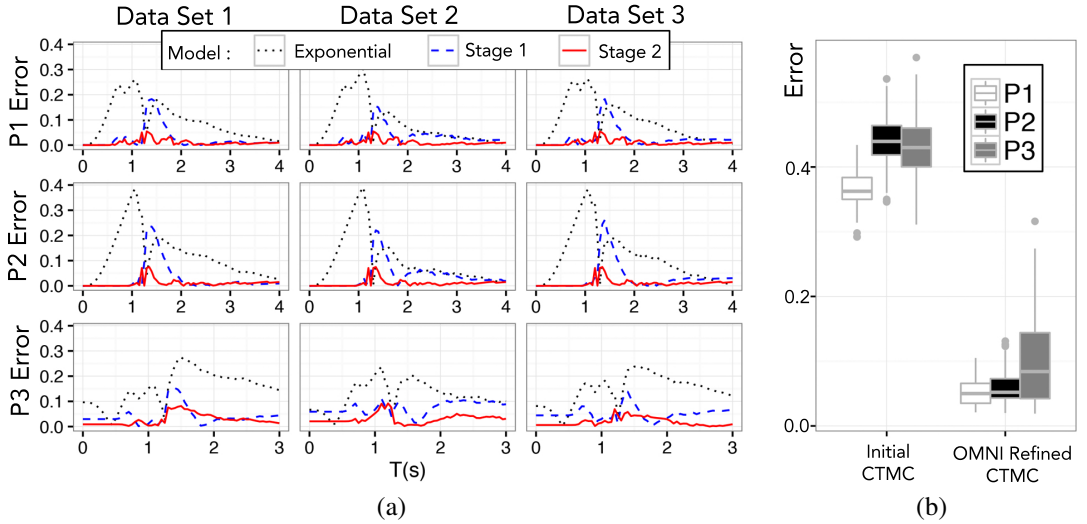
We evaluated OMNI experimentally to answer the following research questions (RQs).

RQ1 (Accuracy): How effective are OMNI models at predicting quality property values for other system runs than the one used to collect the datasets used for the refinement? For any stochastic process, a (sufficiently large) *training dataset* should be sufficient to capture the behaviour of the system. Models based on this dataset should be robust when evaluated against datasets from other system runs (i.e., no *overfitting*). To assess if OMNI models have this property, we used three four-hour runs of the application from Section III, carried out at different times of day over a period of two days, to collect three *testing datasets* of same size to the training dataset from Section IV. The differences between actual and predicted values for each additional dataset are presented in Fig. 7a. The models used in this analysis are the initial CTMC from Fig. 1 (labelled "exponential" in the diagrams) and the refined CTMCs obtained after each OMNI stage for the training dataset and the OMNI parameters from Examples 1 and 2. A visual inspection of the results shows that each OMNI stage significantly improves the accuracy of the analysis. The reduction in the cumulated prediction error (7) across the three testing datasets was in the ranges 82.5–88.6% for property **P1**, 77–90.3% for **P2** and 83.3–89.8% for **P3**.

To eliminate the risk that these improvements were due to an advantageous partition of the data into training and testing sets, we carried out 30 more experiments in which the observations were randomly partitioned into four new datasets, one used for training and the others for assessing fitting errors. The errors from these experiments, summarised as box plots in Fig. 7b, show that OMNI consistently outperforms traditional CTMC modelling irrespective of the choice of training data, confirming the accuracy and robustness of our method.

RQ2 (Refinement Granularity): What is the effect of varying the refinement granularity on the model accuracy, size and verification time? We ran experiments to evaluate the accuracy of the OMNI predictions when varying: (i) the size k_i of the Erlang delay model (cf. Table II); and (ii) the threshold α from Algorithm 1 (with the other OMNI parameters as in Examples 1 and 2). The values of k_i and α determine the granularity of the refinement from the two OMNI stages, with larger k_i and smaller α values corresponding to finer-grained refinement (and larger refined CTMCs). The experimental results (Table IV) show that when only Stage I of OMNI is used, increasing k_i initially reduces the analysis error for all properties, with significant improvements even for small k_i. However, increasing k_i beyond a certain value (approx. 100 for our system) has little effect on the model accuracy. In Stage II, the analysis error decreases when smaller α values are used. However, this decrease is much less significant when changing from $\alpha = 0.1$ to $\alpha = 0.05$ than when changing from $\alpha = 0.2$ to $\alpha = 0.1$. Both results show the presence of a point of "diminishing returns" in continuing the refinement beyond a certain level of granularity (which is necessarily application and training dataset dependent). As a further remark, a coupling effect between α and k_i is suggested by the results. This increases the limit at which k_i still provides improvements in model accuracy, with Stage I+II model accuracy showing

[1] A hyper-Erlang distribution [35], [17], [29] is a PHD in which the $c > 1$ branches of the PHD from Algorithm 1 are mutually independent Erlang distributions.



(a) (b)

Fig. 7: Prediction error for training & testing datasets from different runs (a), and for 30 random training & testing datasets (b)

TABLE IV: Effects of the OMNI refinement granularity

OMNI stage	k_i	α	#CTMC states	P1 Error	P1 verif. time (s)	P2 Error	P2 verif. time (s)	P3 Error	P3 verif. time (s)
Initial CTMC			7	0.325	< 0.1	0.402	< 0.1	0.377	< 0.1
I	10	-	67	0.113	< 0.1	0.15	< 0.1	0.190	< 0.1
I	100	-	607	0.105	0.1	0.121	0.1	0.195	0.1
I	259	-	1561	0.105	1.1	0.119	1.1	0.194	1.2
I+II	10	0.2	122	0.083	0.2	0.122	0.1	0.098	0.3
I+II	100	0.2	662	0.048	0.7	0.064	0.7	0.082	0.9
I+II	259	0.2	1616	0.044	2.8	0.057	2.7	0.079	3.2
I+II	10	0.1	272	0.076	0.6	0.109	0.6	0.086	0.7
I+II	100	0.1	812	0.041	1.4	0.047	1.3	0.066	1.6
I+II	259	0.1	1766	0.037	5.2	0.039	4.9	0.063	5.6
I+II	10	0.05	658	0.073	2.2	0.102	2.2	0.084	2.4
I+II	100	0.05	1198	0.040	4.0	0.042	3.5	0.064	4.1
I+II	259	0.05	2152	0.037	11.9	0.035	11.9	0.060	12.2

improvements as k_i is increased from 100 to 259.

Table IV also reports the number of CTMC states at each level of granularity, and the associated verification times—comprising the time to build the model and to verify each property at a single time point using PRISM on a MacBook Pro with 2.9 GHz Intel i5 processor and 16Gb of memory. As expected, the model size and verification time increase rapidly with the refinement granularity, up to over 2000 states and 12s for several models. Importantly, the results indicate that this increase can be limited by not refining the model beyond the granularity at which the *predicted* property values stabilise, which can be determined e.g. by applying OMNI iteratively at increasing levels of refinement granularity. Another positive insight is that when computational resources are at a premium, small OMNI refined models still provide considerable accuracy improvements.

The component-based system from our case study is realistic, but relatively simple. Therefore, it is worth noting that the increase in model size after the OMNI refinement is only linear in the number of system components. Moreover, as OMNI uses acyclic PHDs, the number of transitions also increases only linearly with refinement. As modern model checkers can analyse CTMCs with 10^5–10^6 states [36], we expect OMNI to scale well to much larger system sizes. We confirmed these hypotheses (Fig. 8) by running additional OMNI refinement

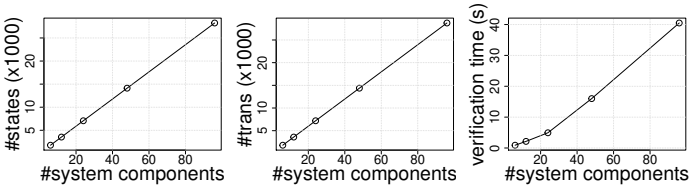


Fig. 8: Refined CTMC states, transitions and verification time for system sizes up to 16 times larger than the web application

experiments for models of systems with 12, 24, 48 and 96 components. These larger models (available on our project webpage) were obtained by combining 2, 4, 8 and 16 instances of our six-component high-level CTMC from Fig. 1, and OMNI was used with observation sets drawn randomly from those of our six web services, $k = 259$ and $\alpha = 0.1$. The verified property was only P1 from (6) for $T = 20$s (as P2 and P3 cannot be meaningfully extrapolated to the larger systems).

RQ3 (Training dataset size): What is the effect of the training dataset size on the refined model accuracy? We ran experiments to assess the accuracy of OMNI predictions when the refined CTMC was derived using subsets comprising only 20%, 40%, 60% and 80% of the elements from our 270-entry training dataset. For each subset size we used $k_i = 259$ and $\alpha = 0.1$, and ran 30 experiments with subset elements drawn randomly from the complete training dataset. As expected, the results (depicted in Table V) show that the OMNI prediction

TABLE V: Training dataset size effect on prediction accuracy, shown as average error and standard deviation over 30 runs

Dataset†	P1 Error	P2 Error	P3 Error
100%††	0.037 sd N/A*	0.039 sd N/A*	0.063 sd N/A*
80%	0.054 sd 0.019	0.054 sd 0.015	0.109 sd 0.053
60%	0.063 sd 0.023	0.066 sd 0.024	0.130 sd 0.066
40%	0.065 sd 0.021	0.063 sd 0.020	0.149 sd 0.068
20%	0.074 sd 0.029	0.075 sd 0.028	0.157 sd 0.068
Initial CTMC	0.325 sd N/A*	0.402 sd N/A*	0.377 sd N/A*

†Percentage of complete 270-element training dataset ††Single run using entire data set *Single run, so no standard deviation

errors increase as the training dataset becomes smaller (except for the P2 error for the 40% dataset, which is marginally smaller than that for the 60% dataset). Importantly, significant accuracy improvements are achieved even for the 20% dataset, so OMNI is effective even when relatively small observation sets can be obtained, e.g. due to time or budget constraints.

RQ4 (Comparison to PHD): How does OMNI compare to a single-stage PHD fitting approach? As PHDs can fit any positive continuous distribution, we assessed the benefit of having a delay extraction stage in OMNI. To this end, we switched off OMNI's delay extraction and produced PHD-only refined CTMCs for $\alpha \in \{0.1, 0.05, 0.01\}$. Table VI compares these models to OMNI-refined CMTCs of *equivalent size* (i.e. small, medium or large) that we constructed by using $\alpha = 0.1$, with enough delay-modelling states to not exceed the PHD-only model sizes. The experimental results show that for all three model sizes, the OMNI models yield more accurate results (with similar or better verification times, and much smaller refinement times) than the PHD models. This confirms existing theoretical results which show that PHD fitting of deterministic delays requires an excessive number of states, and that an Erlang distribution (as used in the first OMNI stage) is the best fit for a fixed delay [24].

VII. Related Work

PHD fitting to empirical data is an active research area, with numerous new fitting algorithms proposed in recent years, e.g. [27], [30], [29], [31]. OMNI leverages these results, and applies them to the refinement of CTMCs used in software engineering. The analysis of non-Markovian processes using PHDs is considered in [37], where a process algebra is proposed for use with the probabilistic model checker PRISM. However, [37] presents only the analysis of a simple system based on well-known distributions, and does not consider PHD fitting to real data nor how its results can be exploited in the scenarios tackled by OMNI.

Delays within a process present particular problems for PHD fitting, and probabilistic regions of zero density are considered in [24], where interval distributions are used to separate discrete and continuous features. Similar work [38] supports the synthesis of timeouts in fixed-delay CTMCs by using Markov decision processes. Unlike OMNI, [38], [24] do not consider essential non-Markovian features of real data such as multi-modal and long-tail distributions, and thus cannot handle empirical data that has these common characteristics.

Finally, the cluster-based PHD fitting method from [32] was used to implement the efficient PHD-fitting tool HyperStar [33]. However, the PHD fitting method and tool from [32], [33] generate a PHD only for a single dataset. OMNI uses this method and tool in its second stage, to refine the high-level CTMC model of a component-based software system.

Non-PHD-based approaches to combining Markov models with real data range from e.g. using Monte Carlo simulation to analyse properties of discrete-time Markov chains with uncertain parameters [39] to using semi-Markov chains to model holding times governed by general distributions [40]. However, none of these approaches can offer the guarantees and tool support provided by OMNI thanks to its exploitation of established CTMC model checking techniques.

VIII. Threats to validity

Construct validity threats may arise due to the assumptions made when implementing our prototype web application and collecting the datasets used for the model refinement experiments. To mitigate them, we implemented the web application using standard Java technologies, and we collected the datasets from six real web services from three different providers, at different times of day and on two different days. We also used different datasets for training and testing, and we analysed typical performance and cost properties of the application.

Internal validity threats can be due to the stochastic nature of the analysed component-based system or bias in interpreting the experimental results. To address these threats, we provided formal proofs for our method, and reported results from multiple independent experiments that use different values for the OMNI parameters, and analyse three system properties at several levels of refinement granularity.

External validity threats may exist if the stochastic features of the analysed system are not indicative of the features of other software systems. To reduce this threat, trace data was obtained from a range of real web services, such that the characteristics of the distributions used exhibited features which we would expect to see in a many real-world applications, e.g., regions of zero density, multi-modal response times and long tails. Additionally external threats exists if the refined CTMC model cannot be verified within the resources of a traditional computer. The OMNI approach allows for the refinement to be carried out at different levels of granularity and our evaluation suggests that significant improvements in prediction accuracy can be achieved with modest enlargement of the models.

IX. Conclusion

We introduced OMNI, an observation-based CTMC refinement method and tool that significantly improve the accuracy with which quality properties of software systems can be analysed using CTMC models. We evaluated OMNI using data obtained from real web services, demonstrating its robustness.

In future work, we will assess the sensitivity of OMNI to the size of the training datasets, and its effectiveness at estimating a wider range of execution time distributions for the system components. In addition, we plan to augment OMNI with

TABLE VI: Comparison of OMNI refinement with PHD-only refinement

Method	Experiment "size"	α	k_i	#CTMC states	Refinement time (m:s)	P1 verif. time (s)	P1 Error	P2 verif. time (s)	P2 Error	P3 verif. time (s)	P3 Error
PHD only	small	0.1	-	485	1:19	0.7	0.059	0.7	0.064	0.9	0.101
	medium	0.05	-	827	1:52	1.4	0.055	1.4	0.058	1.6	0.099
	large	0.01	-	2468	4:07	5.5	0.046	5.4	0.043	6.0	0.097
OMNI	small	0.1	45	482	0:24	0.8	0.049	0.8	0.058	0.9	0.072
	medium	0.1	100	812	0:24	1.4	0.041	1.3	0.047	1.6	0.066
	large	0.1	259	1766	0:24	5.2	0.037	4.9	0.039	5.6	0.063

the ability to intelligently vary the level of refinement across components, so that fine-grained refinement is only used for components that need it.

ACKNOWLEDGEMENTS

This paper presents research sponsored by the UK MOD. The information contained in it should not be interpreted as representing the views of the UK MOD, nor should it be assumed it reflects any current or future UK MOD policy.

REFERENCES

[1] A. B. Bondy, *Foundations of Software and System Performance Engineering*. Addison Wesley, 2014.

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

[3] D. Perez-Palacin, R. Mirandola, and J. Merseguer, "QoS and energy management with Petri nets: A self-adaptive framework," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2796–2811, 2012.

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

[5] N. Sato and K. S. Trivedi, "Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks," in *ICSOC'07*, 2007, pp. 107–118.

[6] R. Calinescu, K. Johnson, and Y. Rafiq, "Developing self-verifying service-based systems," in *ASE*, 2013, pp. 734–737.

[7] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzè, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals to establish quality of service properties of software systems," *IEEE Trans. Reliability*, vol. 65, no. 1, pp. 107–125, 2016.

[8] M. Woodside, D. Petriu, J. Merseguer, D. Petriu, and M. Alhaj, "Transformation challenges: from software models to performance models," *J. Softw. & Syst. Modeling*, vol. 13, no. 4, pp. 1529–1552, 2014.

[9] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, no. 1, pp. 3–22, 2009.

[10] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Quality prediction of service compositions through probabilistic model checking," in *QoSA'08*, 2008, pp. 119–134.

[11] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in *ASE*, 2015, pp. 319–330.

[12] R. Calinescu and M. Kwiatkowska, "CADS*: Computer-aided development of self-* systems," in *FASE*, 2009, pp. 421–424.

[13] S. Gerasimou, R. Calinescu, and A. Banks, "Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration," in *SEAMS*, 2014, pp. 115–124.

[14] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Asp. Comput.*, vol. 24, no. 2, pp. 163–186, 2012.

[15] R. Calinescu, S. Gerasimou, and A. Banks, "Self-adaptive software with decentralised control loops," in *FASE*, 2015, pp. 235–251.

[16] W. J. Stewart, *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[17] P. Buchholz, J. Kriege, and I. Felko, "Phase-type distributions," in *Input Modeling with Phase-Type Distributions and Markov Models*. Springer, 2014, pp. 5–28.

[18] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," in *CAV'96*, 1996, pp. 269–276.

[19] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV'11*, 2011, pp. 585–591.

[20] J. R. Norris, *Markov chains*. Cambridge University Press, 1998.

[21] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *SFM'07*, 2007, pp. 220–270.

[22] R. Calinescu, K. Johnson, and Y. Rafiq, "Using observation ageing to improve Markovian model learning in QoS engineering," in *ICPE'11*, 2011, pp. 505–510.

[23] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakir, "Adaptive model learning for continual verification of non-functional properties," in *ICPE'14*, 2014, pp. 87–98.

[24] L. Krčál, J. Krčál, and R. Vojtěch, "Dealing with zero density using piecewise phase-type approximation," in *EPEW'14*, 2014, pp. 119–134.

[25] C. A. O'Cinneide, "Phase-type distributions: open problems and a few properties," *Communications in Statistics. Stochastic Models*, vol. 15, no. 4, pp. 731–757, 1999.

[26] H. Okamura and T. Dohi, "Fitting phase-type distributions and Markovian Arrival Processes: Algorithms and tools," in *Principles of Performance and Reliability Modeling and Evaluation*. Springer, 2016, pp. 49–75.

[27] A. Horváth and M. Telek, "Matching more than three moments with acyclic phase type distributions," *Stochastic Models*, vol. 23, no. 2, pp. 167–194, 2007.

[28] A. Thummler, P. Buchholz, and M. Telek, "A novel approach for phase-type fitting with the EM algorithm," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 245–258, July 2006.

[29] J. Wang, J. Liu, and C. She, "Segment-based adaptive hyper-Erlang model for long-tailed network traffic approximation," *The Journal of Supercomputing*, vol. 45, no. 3, pp. 296–312, 2008.

[30] H. Okamura, R. Watanabe, and T. Dohi, "Variational bayes for phase-type distribution," *Communications in Statistics - Simulation and Computation*, vol. 43, no. 8, pp. 2031–2044, 2014.

[31] Y. Yamaguchi, H. Okamura, and T. Dohi, "A variational Bayesian approach for estimating parameters of a mixture of Erlang distribution," *Communications in Statistics - Theory and Methods*, vol. 39, no. 13, pp. 2333–2350, 2010.

[32] P. Reinecke, T. Krauß, and K. Wolter, "Cluster-based fitting of phase-type distributions to empirical data," *Computers & Mathematics with Applications*, vol. 64, no. 12, pp. 3840–3851, 2012.

[33] P. Reinecke, T. Krauss, and K. Wolter, "HyperStar: Phase-type fitting made easy," in *QEST'12*, 2012, pp. 201–202.

[34] P. Reinecke, T. Krauß, and K. Wolter, "Phase-type fitting using HyperStar," in *Computer Performance Engineering*. Springer, 2013, pp. 164–175.

[35] S. A. Anichkin, "Hyper-Erlang approximation of probability distributions on $(0, \infty)$ and its application," in *Stability Problems for Stochastic Models*, V. V. Kalashnikov and V. M. Zolotarev, Eds., 1983, pp. 1–16.

[36] D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelinga, and I. Zapreev, "How fast and fat is your probabilistic model checker? an experimental performance comparison," in *HVC'07*, 2008, pp. 69–85.

[37] G. Ciobanu and A. S. Rotaru, "PHASE: a stochastic formalism for phase-type distributions," in *ICFEM'14*, 2014, pp. 91–106.

[38] T. Brázdil, L. Korenčiak, J. Krčál, P. Novotnỳ, and V. Řehák, "Optimizing performance of continuous-time stochastic systems using timeout synthesis," in *QEST'15*, 2015, pp. 141–159.

[39] I. Meedeniya, I. Moser, A. Aleti, and L. Grunske, "Evaluating probabilistic models with uncertain model parameters," *Software & Systems Modeling*, vol. 13, no. 4, pp. 1395–1415, 2014.

[40] G. G. I. López, H. Hermanns, and J.-P. Katoen, "Beyond memoryless distributions: Model checking semi-Markov chains," in *Process Algebra and Probabilistic Methods*. Springer, 2001, pp. 57–70.